Design and Evaluation of High-Performance Processing Elements for Reconfigurable Systems

Sohan S. Purohit, *Member, IEEE*, Sai Rahul Chalamalasetti, *Student Member, IEEE*, Martin Margala, *Senior Member, IEEE*, and Wim A. Vanderbauwhede, *Member, IEEE*

Abstract— **In this paper, we present the design and evaluation of two new processing elements for reconfigurable computing. We also present a circuit-level implementation of the data paths in static and dynamic design styles to explore the various performance-power tradeoffs involved. When implemented in IBM 90-nm CMOS process, the 8-b data paths achieve operating frequencies ranging over 1 GHz both for static and dynamic implementations, with each data path supporting single-cycle computational capability. A novel single-precision floating point processing element (FPPE) using a 24-b variant of the proposed data paths is also presented. The full dynamic implementation of the FPPE shows that it operates at a frequency of 1 GHz with 6.5-mW average power consumption. Comparison with competing architectures shows that the FPPE provides two orders of magnitude higher throughput. Furthermore, to evaluate its feasibility as a soft-processing solution, we also map the floating point unit onto the Virtex 4 and 5 devices, and observe that the unit requires less than 1% of the total logic slices, while utilizing only around 4% of the DSP blocks available. When compared against popular field-programmable-gate-array-based floating point units, our design on Virtex 5 showed significantly lower resource utilization, while achieving comparable peak operating frequency.**

Index Terms— **Computer arithmetic, data path design, reconfigurable computing.**

I. Introduction

DIGITAL signal processing and multimedia applications require large amounts of data, real-time processing ability, and very high computational power. As a result, adaptable architectures with run-time reconfiguration capabilities have received increased attention. As new applications and algorithms continue to evolve, traditional field-programmable gate array (FPGA)-based reconfigurable solutions cease to be viable options due to their bit-level granularity and large amount of routing overhead, which result in a drop in the overall silicon efficiency of the architecture. As a result, in the past few years, the focus of research has shifted to coarse-grained reconfigurable architectures which offer control over 4/8/16/32 b at a time. Several of these architectures, such as RAW [1], MATRIX [2], MorphoSys [3], DAPDNA [4], AsAP [5], Ambric [6], MORA [7], and so on employ regular arrays of processing elements (PEs) connected through multiple levels of interconnection network and working with local or shared memory resources.

At a hardware level, the overall system performance of these architectures depends on: 1) the top-level array and interconnection scheme and 2) the individual processing cell. While factors such as organization of the cells, interconnection network, and memory hierarchy (in case of shared memory architectures) are critical to system throughput, it should be noted that the individual processing cells are the main workhorses of the system and hence are perhaps equally critical to the total processing throughput. It is therefore important to develop extendable arithmetic processing units which allow modular system design in order to guarantee maximum throughput from a reconfigurable array-based architecture. Another important requirement of modern DSP and media processing applications is to provide floating-point capability. This ability, if achieved by reusing or extending integer data paths, allows faster development time, low-cost system implementation, as well as possible FPGA implementation of the data paths.

In this paper, we present the architectures of two integer reconfigurable data paths. The proposed data paths can perform single-cycle addition, subtraction, multiplication, and accumulation operations. They can be used in multicore platforms to perform more complex arithmetic and logical operations. The data paths have a short and uniform critical path across the range of operations. Each of the data paths is extendable and can be parameterized to support higher precision arithmetic, and software-assisted variable-precision reconfigurable systems. Eight-bit versions of the integer data paths were implemented using the IBM 90-nm process using static, domino, and data-driven dynamic logic (D3L) [8]. Simulation results show that the data paths can achieve operating frequencies in the range of 1 GHz. Using the findings from the architectural and circuit analysis on the integer data paths, a new single-precision floating point processing element (FPPE) using the 24-b extension of the data paths is also presented. The full dynamic implementation of the FPPE operates at a frequency of 1 GHz with 6.5-mW average power consumption.

Manuscript received December 10, 2011; revised August 25, 2012; accepted August 31, 2012. Date of publication October 23, 2012; date of current version September 9, 2013.

S. S. Purohit was with the Department of Electrical and Computer Engineering, University of Massachusetts Lowell, Lowell, MA 01854 USA. He is now with Intel Corporation, Austin, TX 78746 USA (e-mail: sohan.s.purohit@intel.com).

S. R. Chalamalasetti and M. Margala are with the Department of Electrical and Computer Engineering, University of Massachusetts Lowell, Lowell, MA 01854 USA (e-mail: sairahul_chalamalasetti@student.uml.edu; martin_margala@uml.edu).

W. A. Vanderbauwhede is with the School of Computing Science, University of Glasgow, Scotland G12 8QQ, U.K. (e-mail: wim@dcs.ac.gla.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2012.2220868

To understand the feasibility of the proposed data paths for FPGA applications, we also performed synthesis experiments using Xilinx Virtex 4 and 5 FPGAs. These experiments helped to understand the tradeoffs associated with choosing optimum granularities and the impact of modularizing large-width operations on system throughput. The FPPE was also synthesized to evaluate its potential as a soft floating point PE. Comparative analysis with competing architectures shows that the proposed FPPE achieves comparable performance at significantly lower resource utilization.

The remainder of this paper is organized as follows. Section II presents a brief background of previous work on data path design. Section III describes the architecture of the integer data paths. Section IV presents the architecture of the floating point PE. Section V presents the VLSI implementation of the integer and floating point PEs. Section VI presents the FPGA-based evaluation of the proposed PEs. Finally, concluding remarks are presented in Section VII.

II. Background of Previous Work

The performance, flexibility, and cost of arithmetic PEs strongly impact the characteristics of the entire system. The value of a high-performance PE is enhanced even more if it adopts an algorithm which can be easily extended, since it allows design reuse and results in a massive reduction in development time as well as cost. Realizing this, several groups have proposed the concept of extendable and reusable arithmetic units. Xydis *et al.* [9] discussed the importance of developing efficient programmable arithmetic algorithms to implement flexible reconfigurable architectures. Their approach focuses on developing a stable interconnect scheme between multiple components, which allows for inline flexibility into the architecture and achieves computational efficiency. However, this approach results in an increased complexity in the interconnection network, which is likely to be the power–performance bottleneck in large array-based systems. In contrast to this, our approach incorporates flexibility into the core computational algorithm itself, allowing large arrays to have the necessary flexibility with a simpler interconnection scheme. Mohammad *et al.* [10] also studied the need to develop digital arithmetic structures and their impact on image processing systems. They achieved this by using a combination of algorithm and circuit development. However, they achieved flexibility by designing microprocessor architectures around their custom algorithm and circuit implementations. Thus, in order for the techniques to work, several architectural and circuit constraints should be placed on the system. Our proposed data paths allow modular use and can be used to replace arithmetic data paths in any architecture, without requiring major system/processor-level architectural optimizations. Gierenz *et al.* [11] and Shanthala *et al.* [12] presented work on generating parameterized arithmetic units for media processing systems.

Floating point capability has been another point of focus of several research projects. Solutions like DAPDNA [4] and Ambric [6] provide floating point support inside each processing core. The MORA [7] architecture, for instance, supports comparison and shifting operations, making it possible for multiple cells to work collaboratively and execute a floating point operation. To balance floating point capability with resource efficiency, hybrid architectures like Garp [13] and Element CXI [14] employ heterogeneous arrays of elements which employ a varying mix of integer as well as floating point resources, with each resource catering to a specific task. An important distinction between these solutions and the FPPE proposed here is the accuracy–performance tradeoff. All the floating point architectures place high importance on achieving maximum computational accuracy within the core. Our proposed FPPE, which is intended to work in a floating point extended version of MORA, prioritizes performance-power and area over accuracy. Our approach allows the floating point cores themselves to be relatively smaller and more performance-power and area efficient. Accuracy and error offsetting can be handled at the array level. For instance, the truncation algorithm used by our FPPE core is likely to result in a worst-case error of the order of 10^{-8}. It may be argued that these errors may accumulate over a series of operations. However, in array-based systems, the idle resources of the array can be easily used to offset these errors. Thus, overall computational accuracy can be achieved at relatively low area and power costs by utilizing the collaboration between multiple array components.

Building on these lines of thinking, the data paths proposed in this paper were implemented to achieve the following key goals.

1) Achieve high-performance computation with low area and power costs.
2) Use an extendable arithmetic algorithm to allow easier expansion in future-generation architectures.
3) Allow easy integration into full processing cores, without requiring major architectural modifications.
4) Modular designs to allow multiple small units to collaboratively handle larger arithmetic computations.
5) Encourage design reuse for application-specified integrated circuit (ASIC) as well as soft-processing applications.

III. Adaptable Architectures for Integer Arithmetic

Multiplication often forms the bottleneck in performance efficiency of arithmetic processing units. The multiplication algorithm used, often dictates the ability of an arithmetic unit to achieve optimum resource utilization and performance efficiency. As a result, the data paths proposed in this paper were designed after a careful choice of multiplication algorithm. Section III-A details the algorithm used by the proposed data paths.

A. Algorithm for N-Bit Multiplication

In its simplest form, an $N \times N$ multiplication requires the generation of $N \times N$ 1-b partial products, which are then added up to deliver the final product. As the size of the multiplier grows, the number of partial products to be processed also increases. For instance, a 4×4 multiplication requires sixteen

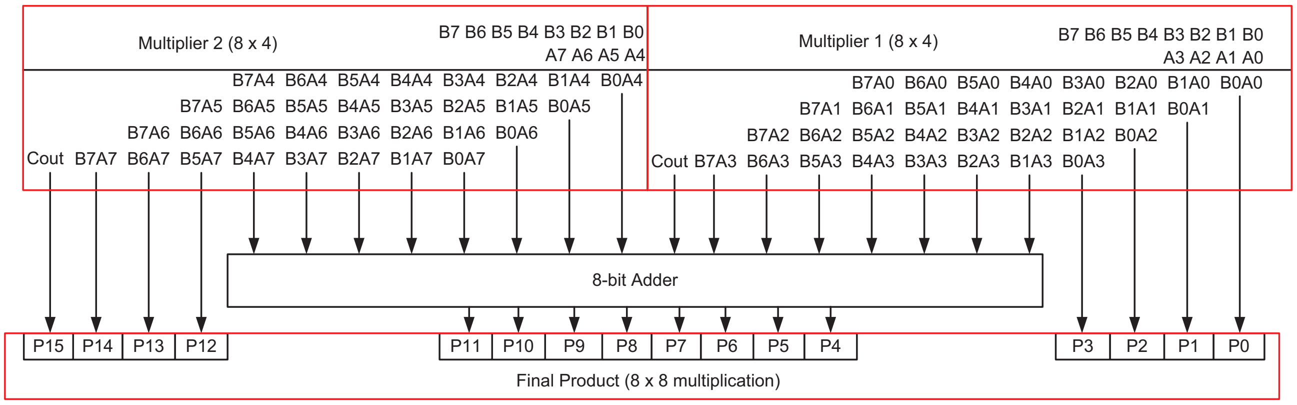
Multiplier 2 (8 x 4)
B7 B6 B5 B4 B3 B2 B1 B0
A7 A6 A5 A4
B7A4 B6A4 B5A4 B4A4 B3A4 B2A4 B1A4 B0A4
B7A5 B6A5 B5A5 B4A5 B3A5 B2A5 B1A5 B0A5
B7A6 B6A6 B5A6 B4A6 B3A6 B2A6 B1A6 B0A6
Cout B7A7 B6A7 B5A7 B4A7 B3A7 B2A7 B1A7 B0A7
Multiplier 1 (8 x 4)
B7 B6 B5 B4 B3 B2 B1 B0
A3 A2 A1 A0
B7A0 B6A0 B5A0 B4A0 B3A0 B2A0 B1A0 B0A0
B7A1 B6A1 B5A1 B4A1 B3A1 B2A1 B1A1 B0A1
B7A2 B6A2 B5A2 B4A2 B3A2 B2A2 B1A2 B0A2
Cout B7A3 B6A3 B5A3 B4A3 B3A3 B2A3 B1A3 B0A3
8-bit Adder
P15 P14 P13 P12
P11 P10 P9 P8 P7 P6 P5 P4
P3 P2 P1 P0
Final Product (8 x 8 multiplication)

Fig. 1. Divide and conquer approach used for implementing large multiplication.

1-b partial products to be added, while a simple increase to an 8×8 multiplication raises this number to 64. In general, an increase in size of the operands by a factor of f results in an f^2 increase in the number of partial products to be processed. Naturally, as this number increases, it becomes necessary to reduce the number of partial products as well as the number of stages of addition. Booth encoding forms one of the most commonly available technique of reducing the partial product terms. The efficient addition of these reduced terms can be implemented through the traditional Wallace tree [15] or array-based multiplier approaches. Techniques such as the use of n:2 compressors have also been proposed to reduce the number of stages of addition of partial products. In [16], Mora-Mora *et al.* present a technique to reduce partial products and speed up multiplication using lookup tables.

While these techniques work well for multiplication-only scenarios, the situation changes when implementing high = performance arithmetic units which are also required to maintain resource efficiency when supporting addition, subtraction, and accumulation operations. In such scenarios, although multiplication forms the most computation-intensive operation, it should balance its resource needs with the other operations to be performed. To solve this issue, we propose breaking up a large multiplication into two smaller ones and adding up the partial products generated. The smaller multipliers as well as the auxiliary circuits can be reused for other operations as per processing demands. Fig. 1 demonstrates our approach for splitting large multiplications, using an 8×8 multiplication as an example. Consider the multiplication of two N-bit numbers A and B. Instead of implementing one large 8×8 multiplier, the operand A can be partitioned into $N/2$-b suboperands A1 and A0, respectively.

Notice that now instead of processing N^2 partial products, we are now required to process in parallel two individual sets of $N^2/2$ partial products. A single implementation of an $N \times N$ multiplier would require a higher depth in the partial product addition, resulting in increased delay in the addition of partial products. It would also increase the timing difference between the shortest and longest partial product summation paths in the tree, thereby resulting in a loss of performance as well as possibilities of glitching. Moreover, a single multiplier would also require more complex placement and wiring in the layout tree which could introduce more area and power inefficiencies.

Another approach would be to use the standard divide and conquer approach of splitting up both the operands into smaller operands of $N/2$ width, as shown in Fig. 2. This approach also allows the computation of inner products, as shown by Lin [17], Van *et al.* [18], and Hong *et al.* [19], thus allowing increased flexibility. However, for computation of the complete $N \times N$ multiplication, which is the target computation in our scheme, this technique requires the use of a reconfigurable switching network, along with the design of complex adders, or adds an extra clock cycle in the addition of intermediate operands. It is worth mentioning however that decomposed multiplication schemes such as that proposed in [17]–[19] would be a valuable addition for multiple-issue architectures that require parallel computation of more than one operation, or for variable-precision arithmetic units. However, our proposed data paths were targeted for the MORA processor [7] which uses a hard-coded granularity of 8 b and supports only single-operation issue per core per cycle. Hence, the proposed technique was selected to act as the best tradeoff between the two multiplication approaches.

B. Proposed Data Path Architectures

Fig. 3 shows the generalized scheme for the first proposed data path. As shown, the data path accepts two N-bit operands A and B through the two N-bit registers. The operand A is then split into $A[N-1{:}N/2]$ and $A[(N/2)-1{:}0]$. The multiplexers controlled by signals $S0$ through $S3$ direct the appropriate operands to the two Wallace tree multipliers. The multipliers generate the intermediate products which are then added up in the compressor stage. The final output is generated by the $2N$-bit carry-linked adder stage. The data path performs N-bit addition, subtraction, and multiplication operations. In case of an accumulation operation, the results of the addition are sent back to the data path through multiplexers controlled by signal $S6$. The operation of the data path for a sample 8-b granularity can be explained as follows.

1) Addition/Subtraction: In case of an addition operation, the multiplier on the left performs $A[7{:}4] \times 1$, while the right multiplier performs $B[7{:}0] \times 1$. The multiplexer controlled

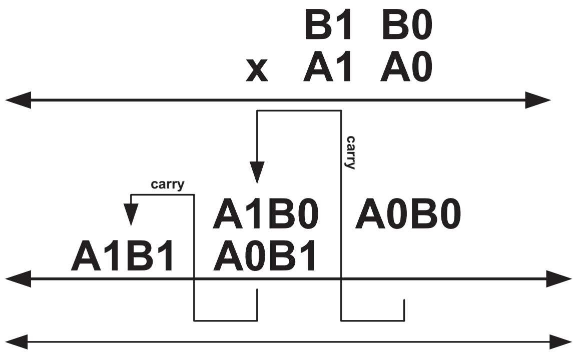

Fig. 2. Summation of partial products and carry propagation in fully decomposed N-bit multiplication.

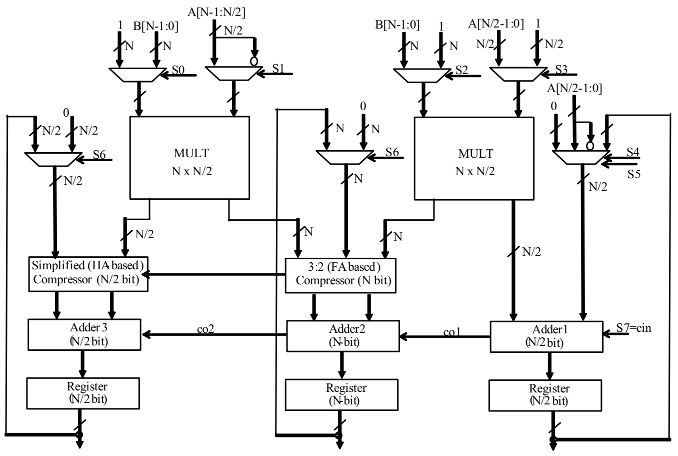

Fig. 3. Generalized N-bit architecture for the proposed data path I.

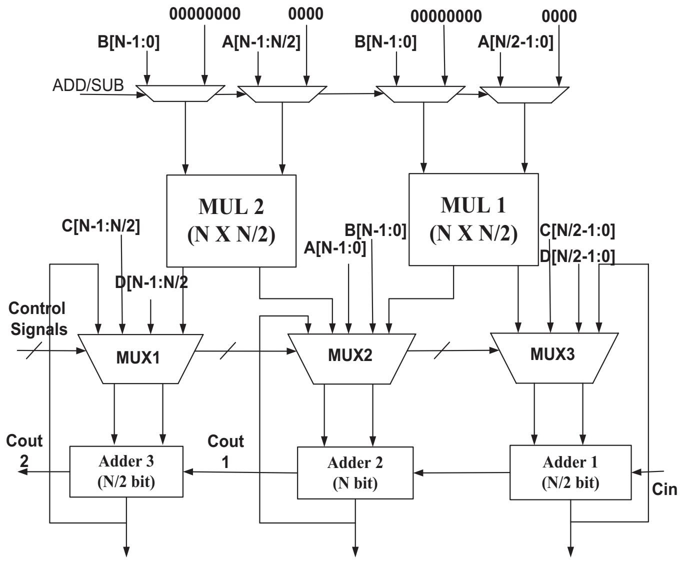

Fig. 4. Generalized N-bit architecture for the proposed data path II.

through $S4$ and $S5$ sends the LSB of operand A (A[3:0]) to the next stage. The partial products are then added up in the compressor and adder stages to generate the final computation $A+B$. Subtraction proceeds in a similar fashion, with the right multiplier now multiplying the complement of A[7:4] with 1 and the control signals $S4$ and $S5$ sending the complement of the LSB of operand A. The signal $S7$, which is the carry input of the adder stage, is set to 1, thus enabling 2s complement subtraction operation.

2) Multiplication: The two multipliers perform $B[7:0] \times A[3:0]$ and $B[7:0] \times A[7:4]$. As explained in Section III-A, the intermediate products of the two multiplications get added up in the 3:2 compressor stage (third input to the compressor is zero in this case). The four LSB bits of $B[7:0] \times A[3:0]$ and the four MSB bits of $B[7:0] \times A$ [7:4] form the LSB and MSB bits of the final product, respectively. The eight intermediate bits of the final product are obtained by adding up the eight MSB bits of $B[7:0] \times A[3:0]$ to the eight LSB bits of $B[7:0] \times A[7:4]$. The data paths employ a combination of compressor and adder stage (an extra increase in gate delay) to perform this addition.

3) Accumulation: To perform an accumulation operation, the output of the adders from the previous execution is sent back through the accumulation control multiplexers controlled through signal $S6$ and added up through the compressors and adders to the next operand in the accumulation.

The data path structure can be repeated for different levels of granularity by varying the value of N (which represents the bit width). This is of particular importance when using the proposed design as an FPGA-synthesizable soft-processing unit. The data path can be parameterized and easily extended to handle wider operands. This feature is also useful in soft-processor solutions to implement data paths of varying widths, thereby guaranteeing optimum performance–resource utilization tradeoffs on the FPGA platform. A toolchain to generate processor structures employing this data path, to handle the operands of required widths, has been proposed in [20]. The data path is said to be "easily" parameterizable since the architecture allows expansion to higher granularity, without any significant changes to the architecture, functioning, or programming model.

The second proposed data path structure is shown in Fig. 4. It can be observed from the figure, that the data path also relies on a divide and conquer approach for multiplication, by following the same operand splitting technique described earlier. However, an advantage over the previously proposed design is that this architecture eliminates the intermediate compressor stage by transmitting the partial products directly to the $2N$-bit carry-linked adders. Multiplexers placed after the multipliers impart additional flexibility and increase the range of operations performed by the data path. These multiplexers are controlled by one-hot select signals ADD, MUL, and ACC, and send the appropriate signals to the inputs of the adders. For a multiplication operation, the multiplexers send the outputs of the two multipliers to the adders. For an addition/subtraction operation, the two operands are selected to be sent to the adders, while for an accumulation operation, the multiplexers send the accumulated result along with a string of zeroes to the adders. Thus, they are effectively 6:2 multiplexers, implemented as two parallel 3:1 multiplex operations. In the ASIC implementation for data path II, these multiplexers were implemented using multioutput design styles, resulting in a total area and power cost lower than the combined cost of multiplexers and compressors in data path I. For an addition or subtraction operation, the operands are directly sent to the adders bypassing the multiplexers. A multiplexer controlled by the addition/subtraction signal, transmits all zeroes to the multipliers, effectively shutting down their dynamic power consumption. Thus, in scenarios where the PE performs a series of addition/subtraction operations, the switching in the multiplier is effectively shut off. Now, the only power consumed in the multiplier is the leakage power. This enables the design to save valuable power consumption by effectively switching off the multipliers during addition or subtraction operations. It should be noted that the real power-saving in this scheme is in scenarios where the data path executes long streams of additions or subtractions. In full functional processors like the MORA processor [7] more fine-grained power gating can be achieved by gating the power supply to the multipliers, when performing addition or subtraction operations. The time needed to wake up the multiplier from this power-gated state can be shadowed by a predictive reading of the operand field from the instruction pointer. This will allow the multiplier to be woken up couple of cycles in advance when a multiplication operation is expected. Added flexibility through the multiplexers also allows the data path to perform two kinds of subtractions $A-B$ and $B-A$. Although this looks simple, it translates into performance savings, by simplifying the memory read and data routing operations when switching between the two different kinds of subtraction. On the accumulation side, the data path allows addition of the $2N$-bit result to two operands A and B read during the next clock cycle.

The proposed data path thus exhibits additional flexibility over the first design, at the cost of increased complexity in the multiplexers. This data path employs complex multiplexers which select from large numbers of inputs. However, the penalty paid in terms of delay, power, and wiring complexity is small enough, not to offset the advantages over the first data path.

Another point worth mentioning is the modular construction of the data paths. It can be observed that both the data paths rely on the two multipliers and the additional multiplexers to deliver the output. To extend the operating range of the structures to signed arithmetic, only the multiplexers need to be redesigned to perform signed arithmetic. We used a hybrid Baugh–Wooley [21] multiplier to accomplish signed multiplication without a loss in the integer range of the operands processed. The operating range was extended to accommodate signed integers by using two Baugh–Wooley hybrid multipliers and an additional control signal to indicate signed operation. Once configured for signed arithmetic, the multipliers generate only signed results, while the compressors and the adders remain blind to the nature of operation being performed. Thus, by following a modular approach, the operating range of the data path was also easily extended to include unsigned as well as signed integer arithmetic.

IV. Single-Precision FPPE

In this section, we present the organization of the proposed FPPE based on the generalized data path architectures

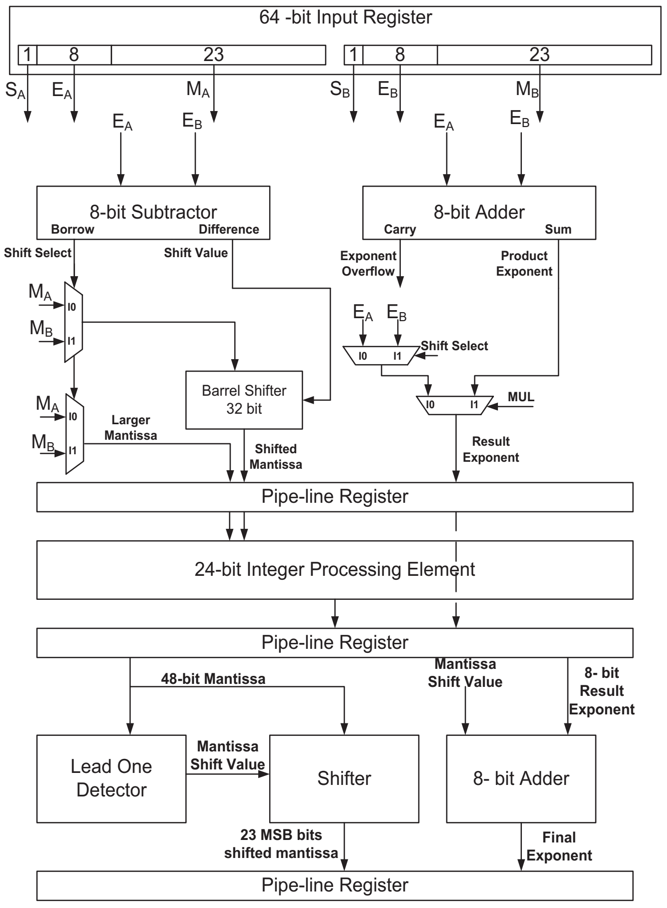

Fig. 5. Organization of the proposed FPPE.

described in Section IV. The proposed FPPE accepts 32-b single-precision floating point operands A and B at the input stage. The operands go through a data conditioning stage which involves aligning the two mantissas M_A and M_B and adjusting the exponents E_A and E_B. These adjusted operands then go through the arithmetic unit which performs the addition, subtraction, and multiplication operations. The result is then normalized and rounded before the output stage. Fig. 5 shows a detailed organization of the proposed FPPE.

The operands A and B are compared for exponent values. The comparison operation involves an 8-b subtraction $E_A - E_B$. Depending on whether $E_A > E_B$ or $E_A < E_B$, the output borrow of the subtractor is set at 1 or reset at 0. This borrow bit is used to control multiplexers which send the mantissa of the smaller number to the shifting unit. The 32-b barrel shifter is controlled through the difference between the two exponents $E_A - E_B$ and shifts the mantissa of the smaller number. This subtraction-based comparison technique thus serves as a common hardware block for exponent comparison as well as shifter control, and eliminates the need for separate blocks to do the same. The output of this unit is thus the larger of the two exponents and the aligned mantissas. The MSB bits of the exponent comparator are used to indicate whether the difference between the two exponents is large enough to shift out the smaller number's mantissa completely. For instance, if exponent of A is greater than that of B, the mantissa of

B is adjusted by shifting it by a value of $E_A - E_B$. Note that both the mantissas are 23-b wide. Thus, a difference of more than $(10111)_2$ between E_A and E_B, that is, the two MSB bits of the exponent comparator being $(11)_2$ means the entire mantissa of the smaller number will be shifted out. Hence, these two bits are used to detect if the difference between the two numbers is large enough for the smaller mantissa to be completely shifted out. In such a case, these bits are used to inhibit the operation of the alignment circuitry, and the final mantissa of the operation is the same as the larger number. The exponent of the result is the larger exponent in case of addition/subtraction or the sum of the two exponents in case of multiplication.

After alignment, the mantissas of the two numbers are sent to a 24-b integer PE. This PE is a 24-b extension of the two data path structures proposed in Section II. A bulk of the area of this data path is occupied by the two 24 × 12 multipliers. Pipelining stages are often required in large data path or multiplier structures, to ensure a high throughput and high frequency of operation. The split-operand approach used in building the multipliers also means that all the product terms become available simultaneously as opposed to a large array-based structure. This multiplier structure thus allows for easy placement of pipeline registers in the data path as well as leaves open the option of pipelining within the multiplier itself. The data path also operates on the result of the exponent. After exponent comparison and mantissa alignment in the earlier stage, the data path retains this value of exponent for addition and subtraction operations while simply doubling the exponent value in case of a multiplication operation.

The mantissa and exponent of the result obtained from the integer PE now need to be normalized and rounded so as to be represented back in the IEEE 754 floating point format. For this purpose, a copy of the mantissa of the result is fed to a modified leading one detector (LOD). This LOD also works as a priority encoder and automatically generates the value by which the mantissa needs to be adjusted so as to satisfy the IEEE 754 notation. This same value is used to adjust the exponent accordingly. Thus at the output we obtain the normalized floating point result. It should be noted that the mantissa rotated through the normalization unit is the full 48-b output of the 24-b integer PE. Once, the mantissa and exponent have been adjusted to the IEEE 754 single-precision format, the 24 LSBs of the rotated mantissa are dropped. That is the 48-b mantissa is truncated. This approach compromises on the accuracy of the result, but maintains the result accuracy within the acceptable limits for most media processing algorithms. Since the proposed FPPEs are intended for array-based systems, it will be possible to employ the idle resources from the array as lookup tables to offset the loss in accuracy. However, this is still under exploration and hence accuracy loss has not been quantitatively addressed in this paper.

As shown in the figure, the mantissa alignment, 24-b integer data path, and the normalization unit form the three pipeline stages in the data path. The 24-b integer data path forms the critical path in the design, and hence determines the peak operating frequency of the proposed FPPE. The pipelining allows the FPPE to work at a higher frequency (dominated by the integer data path), with a latency of three clock cycles. Three stages of pipelining were selected here, so as to maintain a tradeoff between latency and throughput. However, it should be noted that the 24-b data path itself can be fully pipelined in order to further improve the FPPE performance. The partitioning of the FPPE during pipelining also results in the separation of the logic and arithmetic sections of the data path. This allows a potential for the data path to be also extended to perform integer arithmetic and logic operations.

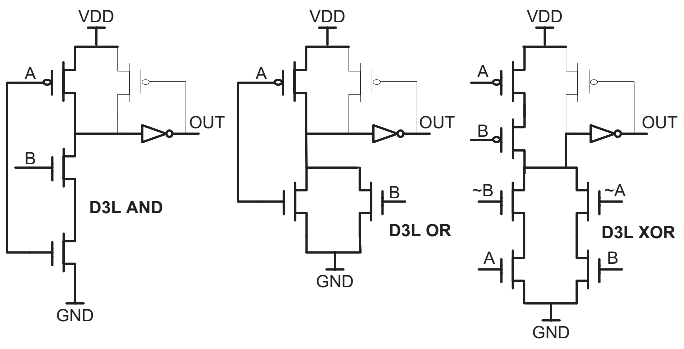

Fig. 6. Basic gates in D3L. (a) AND. (b) OR. (c) XOR.

V. ASIC Implementation and Performance Analysis

A. Static and Dynamic Circuit Implementation

As we continued to explore different PE topologies for efficient arithmetic and logic operation, a separate study involving various design styles was performed in order to have a high-speed area-efficient and power-efficient PE architecture. As a part of this paper, we implemented the proposed data path architectures using standard CMOS, domino, and D3L implementations. The evaluation provided an estimate of the data path performance using static as well as dynamic design styles.

Apart from static CMOS and domino logic styles, another high-speed dynamic style explored was the D3L [8]. Unlike conventional dynamic circuits, which use a clock signal to precharge, D3L uses combinations of input signals to achieve the alternate precharge–evaluate cadence required for dynamic operation. Fig. 6 shows the basic AND/OR/XOR gates implemented using the D3L methodology. It can be observed that the inputs are carefully ordered so that the pull-down network is fully cut-off during the precharge phase, while the pull-up network is ON. This allows the output node to be precharged. Then, depending on the values of inputs during evaluation, the node discharges to zero or remains precharged. Thus, these circuits achieve dynamic circuit operation, but without the use of a clock signal. However, they still offer the high-speed advantages of dynamic circuits, due to a shorter precharge path. This performance however comes without the use of a clock signal. D3L thus allows us to retain the high-speed advantages of conventional dynamic circuits, without the extra power associated with the clock-distribution network. A key point to note in the design of data path II is the design of the complex multiplexers. The total area/power cost of the

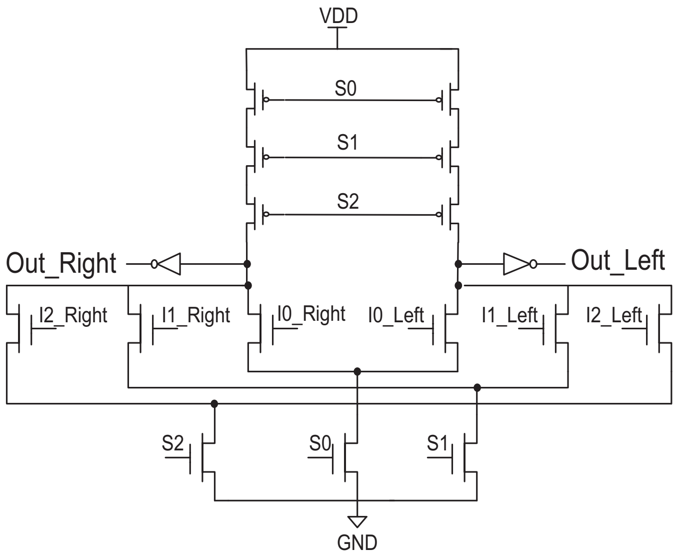

Fig. 7. D3L implementation of complex MUX in data path II.

multiplexers had to be equal or better than the combined cost of multiplexers and compressors of data path I. Since dynamic implementations use small precharge networks, it was possible to implement the multiplexers using multioutput versions of the design styles, resulting in area and power savings. Fig. 7 shows the implementation of a single bit of the multiplexer using D3L methodology. During the precharge phase, all the inputs are brought down to zero. During this phase, the pMOS pull-up network formed by the series combination of select signals S0, S1, and S2 is turned on, while the pull-down network is completely turned off. Hence, both the output nodes are precharged to zero. During the evaluation phase, one of the three select lines gets turned ON, resulting in the pull-up network being turned off. Now depending on the select lines that are turned ON, one input out of I0, I1, and I2 from the left and one from the right are sent to the output. Thus, the multiplexer performs two 3:1 multiplex operations in parallel and generates inputs to the adders. Each bit slice of the D3L multiplexer costs 19 transistors (total $19 \times 16 = 304$). When compared to the D3L version of data path I which uses 8 full adders (32 transistors each), 4 half adders (18 transistors each), and 20 multiplexers (static multiplexers using 6 transistors each), this thus translates into a transistor count reduction of 32% over data path I. Moreover, the precharge and evaluate networks in each of these implementations are smaller than those found in full adders, resulting in faster performance as well as lower power consumption.

We implemented 8-b variants of the proposed data paths in IBM's 90-nm CMOS process. Table I shows the performance comparison of the two data paths designed in static, domino, and D3L topologies. It can be observed that data path II always outperforms data path I. For the nominal 1.2-V operating voltage, it can be seen that data path II is 2%–14% faster and consumes 27%–45% lower power than data path I. This results in around 37%–50% better PDP than data path I. This is primarily due to the decrease in the number of multiplexers and inverters, as well as the elimination of a complete compressor stage. Besides the reduction in the number of gates, the sizing requirements on individual stages are also relaxed. The output from the multiplier now has to drive multiplexers instead of full adders (from the compressor), thus allowing for relaxed sizing. This ripples back throughout the multiplier, thus allowing for smaller sizes on the addition tree, partial product generating AND gates, and hence consequently, the inputs to the multiplier. Cumulatively, they result in the large power saving obtained in data path II. This makes data path II an ideal candidate to implement the arithmetic unit inside the proposed FPPE.

An important objective of this paper was to analyze the best possible implementation style for the data path designs so as to optimize their performance according to the application domain as well as field of use. Table I also shows the performance-power tradeoffs in using each of the three design styles, with the supply voltage swept from the nominal 1.2 to 0.7 V. It was observed, that at higher supply voltages (1.2–1.1 V), compared to D3L and static versions, the domino data path performs 8%–12% and 14%–24% faster in data paths I and II, respectively, making it the best implementation strategy for purely high-performance systems. Due to the large amount of power dissipated in the clock tree, however, the domino version also shows maximum power consumption at every operating point, for each data path.

Fig. 8 shows the power delay product (PDP) plots for the two data paths when simulated for the supply voltage sweep. It can be observed that scaling down the supply voltages yields a significant decrease in the PDP for both the data paths. This implies that for both the data paths, the power reduction due to supply voltage scaling sufficiently offsets the degradation in performance, thereby allowing the data paths to operate with increased power efficiency in strictly power-constrained environments. An exception to this is observed around the 0.8-V range, where all the implementations show an increase in PDP. Based on our simulation results, as well as previous experiments [22]–[26], we concluded that, for this particular technology, 0.8–0.7 V is the region where both the P and N devices are partially ON, resulting in a large increase in delay. Especially, the dynamic topologies, which rely heavily on fast switching of the P and N devices, are more susceptible to delay degradation at this point, resulting in an increase in PDP, as seen from Fig. 8. It can also be observed that at almost all supply voltages, the D3L version offers better PDP compared to domino and static versions. Beyond 0.8 V however, the static version shows approximately 6% lower PDP than the D3L version. An exception to this is the 0.7-V operating point for data path I, where the static implementation shows 18% better PDP. It is worth considering however, that out of the 18 data points for PDP comparison for each data path, the D3L comes in second in only two-third cases. Thus, D3L shows PDP superiority over the widest range of operating conditions, and makes itself the ideal candidate for implementing power-performance efficient systems. In [24] and [25], we have shown evaluations of static, domino, D3L, as well as pass-transistor-based adder chains and multiplier units. In each case, the D3L circuits outperformed the others due to superior PDP. An improved version of D3L, which achieves

TABLE I
PERFORMANCE EVALUATION OF PROPOSED DATA PATHS IN DIFFERENT DESIGN STYLES

Supply Voltage (V)	Design Style	Data Path I				Data Path II			
		Delay (ps)	Power (mW)	Area (mm^2)	PDP (10^{-18}W.s)	Delay (ps)	Power (mW)	Area (mm^2)	PDP (10^{-18}W.s)
1.2 V	Static	1012	4.192	0.0209	4242	988	2.412	0.0241	2383
	Domino	887	5.371	0.0181	4764	759	3.925	0.0176	2979
	D3L	961	3.955	0.0199	3800	879	2.148	0.0194	1888
1.1 V	Static	1198	3.184	0.0209	3814.432	1169	1.832	0.0241	2141.608
	Domino	1139	4.033	0.0181	4593.587	974	2.948	0.0176	2871.352
	D3L	1249	2.725	0.0199	3403.525	1143	1.495	0.0194	1708.785
1.0 V	Static	1282	2.752	0.0209	3528.064	1252	1.583	0.0241	1981.916
	Domino	1355	3.352	0.0181	4541.96	1160	2.449	0.0176	2840.84
	D3L	1279	2.547	0.0199	3257.613	1170	1.383	0.0194	1618.11
0.9 V	Static	1349	2.329	0.0209	3141.821	1317	1.34	0.0241	1764.78
	Domino	1579	2.907	0.0181	4590.153	1351	1.967	0.0176	2657.417
	D3L	1388	2.298	0.0199	3189.624	1270	1.247	0.0194	1583.69
0.8 V	Static	1669	1.761	0.0209	2939.109	1629	0.965	0.0241	1571.985
	Domino	2160	2.271	0.0181	4905.36	1848	1.3	0.0176	2402.4
	D3L	1760	1.783	0.0199	3138.08	1609	0.977	0.0194	1571.993
0.7 V	Static	2386	1.274	0.0209	3039.764	2329	0.681	0.0241	1586.049
	Domino	3303	1.682	0.0181	5555.646	2826	0.82	0.0176	2317.32
0.7 V	D3L	2691	1.339	0.0199	3603.249	2462	0.685	0.0194	1686.47

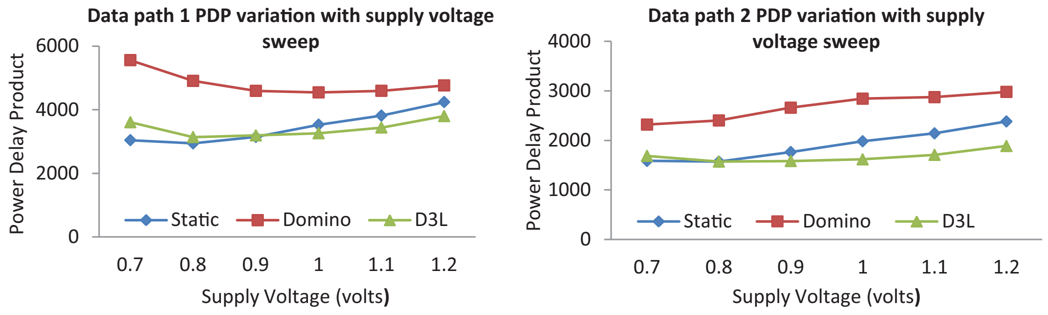

Fig. 8. Variation in the PDP of the proposed data paths with supply voltage sweep.

faster precharge by splitting the precharge path, was proposed in [26]. While static and D3L implementations show slower carry propagation chains than domino, the power consumed by the clock tree in a domino circuit completely offsets any performance improvement gained.

In a previous study focusing on D3L implementation of data path, we verified the D3L methodology across 8-, 16-, and 32-b data path widths [22]. In each case, the D3L implementation was found to be superior to domino and static data paths. Architecturally, data path II eliminates the compressors and multiplexers found in data path I. For 16- and 32-b variants of data paths, eliminating these units results in even more area, power, and performance advantages for data path I over data path II. Thus, the performance results are expected to show a similar trend even for wider sets of operands.

The proposed data paths are intended to be used in the MORA reconfigurable processor. We recently implemented the MORA processor in IBM 90-nm technology node using data path II. Using an in-house compiler and cycle accurate simulator [27], we evaluated the performance of the 64-processor MORA array when performing 8×8 2D-DCT, DWT, and H.264 integer transform algorithms. Table II presents the results of benchmarking the MORA processor implemented using the proposed data paths. The processor as well as the programming model has been described in detail in [27], for the interested reader. The data paths form the main PEs of each MORA processor in the array, and also determine the peak operating frequency of each processor. The D3L implementations of data paths I and II operate at peak frequencies of 1.04 and 1.13 GHz, respectively, at 1.2-V supply voltage. These results have been included to show the

TABLE II
ESTIMATED THROUGHPUT RESULTS FOR THE MORA RC WHEN IMPLEMENTING POPULAR BENCHMARK ALGORITHMS

Data-Paths Operating at 1.2 V		I	II
Algorithm	Cycles	Delay (ns)	Delay (ns)
8 × 8 2D-DCT, min delay	72	69.192	63.288
8 × 8 2D-DCT, max throughput	256	246.01	225.024
4 × 4 H.264, min delay	18	17.298	15.822
4 × 4 H.264, max throughput	32	30.752	28.128
32 × 32 DWT LeGall (5, 3)	2432	2336.1	2137.728

TABLE III
PERFORMANCE VARIATION OF THE PROPOSED FPPE WITH POWER SUPPLY SWEEP

Voltage (V)	F_{MAX}	Worst Delay (ps)	Power (mW)	PDP (10^{-15}W.s)
1.2	1 GHz	1000	6.53	6530
1.1	769 MHz	1300	4.78	6220
1	751 MHz	1331	4.09	5450
0.9	692 MHz	1445	3.44	4964
0.8	546 MHz	1831	2.68	4917
0.7	357 MHz	2801	1.83	5140

application of the proposed data paths in a real processor array. More details on the implementation of the MORA processor as well as other extensions of the proposed data paths have been published in [28].

B. Implementation of FPPE

Based on the study presented in Section V, we implemented the FPPE using a 24-b variant of data path II in D3L design methodology. Table III shows the performance variation of the proposed FPPE with supply voltage scaling. The table reports the peak operating frequency achieved by the FPPE. As mentioned in Section V, the FPPE was implemented as a pipeline design. Within the pipeline, the 24 × 12 multipliers inside the 24-b version of data path II, forms the critical path. The delay through this critical path was used to determine the peak operating frequency and is listed in Table III. The delays through the rest of the pipeline stages are very small compared to that of the 24 × 12 multiplier and hence do not contribute to the critical delay. Fig. 9 shows the PDP variation curves for the FPPE with power supply scaling. From Fig. 9, it is clear that this trend continues until we hit the 0.8- to 0.9-V region. As mentioned earlier, beyond this is the region where the delay degrades more than the saving obtained in power, resulting in a slight upward trend in the PDP. The lowest PDP was calculated at 0.8 V, making it the most performance-power-efficient point for operating the FPPE.

Finally, to quantify our choice of architecture and implementation strategy we compared our floating point implementation with a few recently proposed floating point units targeted for reconfigurable architectures. One of the architectures included for comparison was a floating point unit by Liou *et al.* [29], which was designed to work with an ALU cluster as a part of a reconfigurable accelerator. The design is based on a stream processing model similar to the target reconfigurable array for which our FPPE is intended, which makes it an interesting comparison. However, a major differentiator is the fact that this FPU relies on an array environment even for basic floating point operability. In contrast to this, the proposed FPPE works well in array-based environments, but can also be used in isolated processing cells. Another architecture included in the comparison was the floating point unit for the CELL processor [30] which is an interesting design considering that the architecture, logic design, implementation, and final integration were simultaneously planned and executed to target the

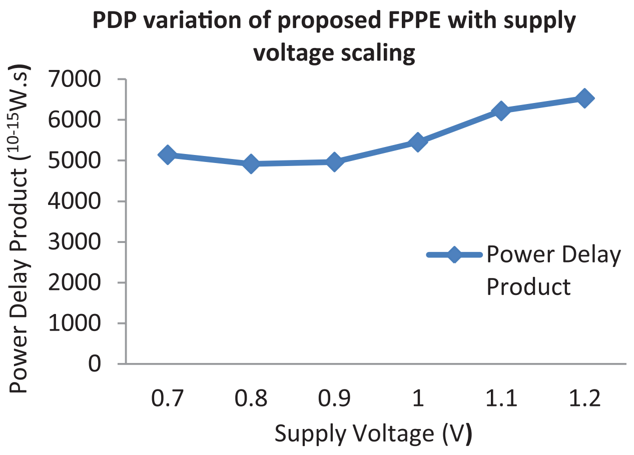

Fig. 9. PDP variation of the proposed FPPE with power supply sweep.

optimum area, power, and performance balance, an approach similar to the one behind MORA, the target implementation platform for our FPPE. We also compared our results with the fully pipelined unit proposed by Huang *et al.* [31] which follows a cost optimization strategy identical to ours. While making the comparison, we have included the implementation technology node as well as the reported area, power, and performance parameters. The architectures were then compared for power, area, performance as well as throughput (MOPS), throughput per milliwatt (MOPS/mw) and throughput per milliwatt per square millimeter (MOPS/mW/mm^2). This factor is beneficial since it takes into account the architecture as well as implementation strategies. The results are presented in Table IV. It can be observed that the proposed architecture offers almost 30% improvement in power and 50% area improvement while working at 1 GHZ with a latency of three clock cycles per instruction. In terms of throughput efficiency, the proposed design shows up to three orders of magnitude higher throughput/power/area efficiency compared to the other architectures. Table IV shows, that for throughput purely based on clock frequency, the design by Oh *et al.* [30] outperforms all other designs. However, after considering power and area in the comparison, it is observed, that the design by Oh *et al.* consumes significantly larger amount of power compared to the proposed design. As a result, the proposed design shows the best throughput efficiency among all the designs considered here.

TABLE IV
PERFORMANCE COMPARISON OF THE PROPOSED FPPE WITH COMPETING ARCHITECTURES

Design	Performance Characterization							
	Process	F_{MAX}	Latency	Power (mW)	Area (mm^2)	Throughput (MOPS)	Throughput (MOPS/mw)	Throughput (MOPS/mW/mm^2)
Proposed	IBM 90 nm	1 GHz	3	6.526 mW	0.2493	1000	153.2332	614.653
Liou *et al.* [29]	TSMC 150 nm	75 MHz	-	10.85 mW	0.415	75	6.912442	16.69
Oh *et al.* [30]	90 nm SOI	4 GHz	6–7	1.4 W	1.3	4000	2.857143	2.197
Huang *et al.* [31]	TSMC 180 nm	390 MHz			1.934	390		-

TABLE V
NORMALIZED PERFORMANCE COMPARISON OF THE PROPOSED FPPE WITH COMPETING ARCHITECTURES

Design	Normalized Performance			
	Max. Frequency	Power (mW)	Area (mm^2)	Throughput (MOPS/mW/mm^2)
Proposed	1 GHz	6.526 mW	0.2493	614.653
Liou *et al.* [29]	125 MHz	6.51 mW	0.249	77.11
Oh *et al.* [30]	4 GHz	1.4 W	1.3	2.197
Huang *et al.* [31]	780 MHz	Not reported	0.967	-

Despite the fact that the choice of implementation technology is a decision made at the beginning of the design process and to a large extent governs the architecture and underlying circuit design decisions, some may argue that a straightforward comparison across different technology nodes cannot be considered a fair comparison. Hence for the interested reader, we have also normalized the numbers to the 90-nm technology node using standard scaling rules and compared the architectures again for area, power, and performance efficiency. The results are tabulated in Table V. It can be observed that after normalization to the 90-nm node, our architecture continues to fare significantly better than the others in terms of performance, power, and area efficiency. The floating point unit proposed by Liou *et al.* [29] uses a finite-state machine approach to control the FPU operation. This FSM design itself takes up around 40% overhead. This reduces the overall power and area efficiency of the architecture. The CELL processor achieves a very high operating frequency of 4 GHz. However, this has been achieved by the insertion of extra latency in the data path. The design also uses static–dynamic converters and dynamic multiplexers, resulting in very high power consumption. Thus our performance margin is due to the choice of architecture as well as implementation strategy. Moreover, the significant margin of advantage means that even if additional processing needs to be performed to offset the inaccuracy of the rounding process, the architecture will still continue to deliver higher performance per unit power per unit area.

VI. PROPOSED DATA PATHS FOR SOFT-PROCESSING APPLICATIONS

Recently, several soft-processing cores have been introduced to map onto the FPGAs and function as complete 8/16/32 b RISC processors. As described in Sections III–V, the architecture of the proposed data path units can be extended to process any N-bit data. From an FPGA implementation perspective, this means that the width of the data paths can be easily parameterized. This property is useful when considering the applications of the proposed data path architectures as soft-processing solutions as it allows synthesizing data paths of appropriate width depending on the requirement of the application being serviced. This section describes our experiments on exploring the feasibility of the proposed data paths and the FPPE as solutions for FPGA-based soft-processing applications.

A. Exploring Optimal Granularity for Soft Processor

We implemented 8-, 16-, and 32-b versions of data path I on Xilinx FPGAs to understand the tradeoffs involved. Table VI shows the performance and resource utilization of the data path units when implemented on a series of Xilinx FPGAs. All the three versions of data path I were constructed using parameterized VHDL code for data path structure I. To understand the implications of choice of granularity on resource utilization, performance, and functional flexibility, we implemented the 32-b structure using six 8-b and three 16-b data paths. Table VII shows the results of the implementation of 32-b data path using the smaller units. These units have been compared against stand-alone 32-b unit using the general data path structure. It can be observed that for 32-b granularity, the single 32-b block outperforms its modularized implementations significantly. This can be attributed due to the large overhead incurred in routing and control structures on the FPGA. When implementing soft-processing systems on the FPGAs, a similar study can be used to find an optimum

TABLE VI
PERFORMANCE –GRANULARITY TRADE-OFFS IN FPGA IMPLEMENTATION OF DATA-PATH I

Device	Operand Size	Slices Occupied	Frequency (MHz)	Throughput (MOPS)
Spartan 3 (XC3S200)	8-b	89	134	103.08
	16-b	188	83	63.85
	32-b	358	60	46.15
Virtex 2P (XC2VP20)	8-b	90	168	129.23
	16-b	163	136	104.62
	32-b	350	97	74.62
Virtex 4 (XC4VKX15)	8-b	107	253	194.62
	16-b	170	158	121.54
	32-b	342	115	88.46
Virtex 5 (XC5VLX30)	8-b	62	232	178.46
	16-b	172	145	111.54
	32-b	610	101	77.69

TABLE VII
COMPARISON OF MODULARIZED 32-B DATA-PATH IMPLEMENTATIONS ON VIRTEX 5 FPGA

Design	Slices Occupied	No. of Building Blocks	Frequency Supported (MHz)	Throughput 32-b Applications	Throughput 16-b Applications	Throughput 8-b Applications
32-b	610	1	101	77.69 MOPS	77.69 MOPS	77.69 MOPS
32-b using 16-b	876	3	163	65.46 MOPS	196.38 MOPS	196.38 MOPS
32-b using 8-b	2080	6	178	55.68 MOPS	167.04 MOPS	334.08 MOPS

TABLE VIII
COMPARISON OF PROPOSED FPPE AGAINST OTHER FPGA FLOATING POINT IMPLEMENTATIONS

Architecture	Device	Slices	Utilization	Peak Operating Frequency
Proposed	Virtex 5	339/51840	0.653%	79 MHz
Xilinx APU	Virtex 5	2620/51840	5.054%	200 MHz
Hockert *et al.*	STRATIX III	2601/56800	4.579%	-
Liang *et al.* (A)	Xilinx XCV300E-6	900/82944	1.085%	75 MHz
Liang *et al.* (B)	Xilinx XCV300E-6	790/82944	0.952%	20 MHz
Nallatech core	Xilinx XCV300E-6	1600/82944	1.929%	120 MHz

tradeoff point between performance, resource utilization, and routing overhead.

B. Proposed FPPE as a Soft Processor Data Path

To evaluate its feasibility in FPGA-based soft-processing solutions, the FPPE was implemented in VHDL and synthesized on the Virtex family of FPGAs to demonstrate the performance, cost, and portability of the proposed architecture. The results of the mapping on Virtex 4 and Virtex 5 FPGA devices from Xilinx are presented in Table VI. As shown in the table, the proposed data path maps well on both the FPGA devices, occupying only a small percentage of the total resources, with a relatively good operating frequency. We also compared the synthesized version of our FPPE against several FPGA-based single-precision floating point units to estimate the effectiveness of our FPPE as a soft-processing solution. The Xilinx auxiliary processor unit (APU) [32] implemented on Virtex 5 is a dedicated floating point coprocessor capable of single-precision and double-precision units. Since we are designing a single-precision unit, we have used the single-precision data from the data sheet for our comparison. Since we are targeting a soft-processor implementation, we included in our comparison results, the fractured floating point unit by Hockart *et al.* [33] targeted at the Nios soft-processor platform from Altera. This design is similar to ours considering the fact that resource and design cost were the primary constraints of the designers. We also looked at floating point cores developed by Liang *et al.* [34] using an optimized floating point core generator as well as floating point cores from Xilinx and Nallatech [35] to get a more in-depth comparison. All the designs were compared based on number of slices utilized and frequency of operation achieved. Considering the fact that each of these designs was implemented on a different platform, we have also calculated the estimated resource utilization of each design on its respective FPGA device. Table VIII compares the designs for area, speed, and resource utilization. It can be observed that while our design does not synthesize to the fastest solution on the FPGA, it offers the best area–speed tradeoff when contrasted against the other competing designs.

VII. Conclusion

In this paper, we presented our recent efforts in the design of high-throughput and low-area, data path elements for reconfigurable media processing architectures. When implemented using the static, domino, and D3L methodologies, over a wide range of operating voltages, the D3L version was found to be superior over most of the operating range of both the data paths. It was observed that data path II was around 14% faster and consumed 27%–45% lower power than data path I. Over the entire operating range from 1.2 to 0.7 V, data path II showed around 37%–50% better PDP than data path I, and hence it was selected to build the FPPE.

The data paths are scalable and parameterizable. This was demonstrated through the implementation of a new FPPE. The generalized structure of the data paths makes them ideal implementation platforms for soft-processing-based systems. When implemented on the Virtex FPGA devices, the FPPE soft core occupied only a fraction of the resources, thus allowing for a large number of cores to be mapped for a high-performance FPGA-based solution. Our future efforts in this area will involve integrating these data path structures into a hybrid, multigranular ASIC as well as soft-processing reconfigurable array for low-cost, high-throughput multimedia processing.

References

[1] M. Taylor, J. Psota, A. Saraf, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, A. Agarwal, W. Lee, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. Johnson, and J. Kim, “Evaluation of the raw microprocessor: An exposed-wire-delay architecture for ILP and streams,” in *Proc. 31st Annu. Int. Symp. Comput. Arch.*, 2004, pp. 2–13.

[2] E. Mirsky and A. DeHon, “MATRIX: A reconfigurable computing architecture with configurable instruction distribution and deployable resources,” in *Proc. IEEE Symp. FPGAs Custom Comput. Mach.*, 1996, pp. 157–166.

[3] H. Singh, M. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and C. Filho, “MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications,” *IEEE Trans. Comput.*, vol. 49, no. 5, pp. 465–481, May 2000.

[4] *DAPDNA-2 Product Brochure*. (2010) [Online]. Available: http://www.ipflex.com

[5] D. Troung, “A 167-processor 65 nm computational platform with per-processor dynamic supply voltage,” in *Proc. Symp. VLSI Circuits*, Jun. 2008, no. C 3.1, pp. 22–23.

[6] M. Butts, “Synchronization through communication in a massively parallel processor array,” *IEEE Micro*, vol. 27, no. 5, pp. 32–40, Sep.–Oct. 2007.

[7] S. Chalamalasetti, S. Purohit, M. Margala, and W. Vanderbauwhede, “MORA-an architecture and programming model for a resource efficient coarse grained reconfigurable processor,” in *Proc. 4th NASA/ESA Conf. Adapt. Hardw. Syst.*, San Francisco, CA, 2009, pp. 389–396.

[8] R. Rafati, S. M. Fakhraie, and K. C. Smith, “ A 16 bit barrel shifter implemented in data-driven dynamic logic,” *IEEE Trans. Circuits Syst.*, vol. 53, no. 10, pp. 2194–2202, Oct. 2006.

[9] S. Xydis, G. Economakos, and K. Pekmestzi, “Designing coarse-grain reconfigurable architectures by inlining flexibility into custom arithmetic data-paths,” *Integration, VLSI J.*, vol. 42, pp. 486–503, Mar. 2009.

[10] K. Mohammad, S. Agaian, and F. Hudson, “Implementation of digital electronic arithmetics and its application in image processing,” *Comput. Electr. Eng.*, vol. 36, pp. 424–434, Jan. 2010.

[11] V. Gierenz, C. Panis, and J. Nurmi, “Parameterized MAC unit generation for a scalable embedded DSP core,” *Microprocess. Microsyst.*, vol. 34, pp. 138–150, Nov. 2010.

[12] S. Shanthala and S. Kulkarni, “VLSI design and implementation of low power MAC unit with block enabling technique,” *Eur. J. Sci. Res.*, vol. 30, no. 4, pp. 620–630, 2009.

[13] J. Hauser and J. Wawrzynek, “Garp: A MIPS processor with reconfigurable co-processor,” in *Proc. Int. Conf. FPGA Custom Comput.*, 1997, pp. 24–33.

[14] P. Athanas, “Element CXI: Exploring elemental computing in academia,” in *Proc. Int. Conf. Eng. Reconfig. Syst. Appl.*, Jul. 2009, pp. 1–8.

[15] C. Wallace, “A suggestion for a fast multiplier,” *IEEE Trans. Electron. Comput.*, vol. 13, no. 1, pp. 14–17, Feb. 1964.

[16] H. Mora-Mora, J. Pascual, J. Sanchez-Romero, and J. Garcia-Chamizo, “Partial product reduction by using look-up tables for MXN multiplier,” *Integr. VLSI J.*, vol. 41, pp. 557–571, Mar. 2008.

[17] R. Lin, “Reconfigurable parallel inner product processor architectures,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 9, no. 2, pp. 261–272, Apr. 2001.

[18] L. Van and J. H. Tu, “Power-efficient pipelined reconfigurable fixed-width Baugh–Wooley multipliers,” *IEEE Trans. Comput.*, vol. 58, no. 10, pp. 1346–1355, Oct. 2009.

[19] S. Hong, K. S. Park, and J. H. Mun, “ Design and implementation of a high-speed matrix multiplier based on word-width decomposition,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 4, pp. 380–392, Apr. 2006.

[20] S. Chalamalasetti, W. Vanderbauwhede, S. Purohit, and M. Margala, “A low cost reconfigurable soft processor for multimedia applications: Design synthesis and programming model,” in *Proc. Int. Conf. Field Program. Logic Devices*, 2009, pp. 534–538.

[21] C. Baugh and B. Wooley, “A 2s complement parallel array multiplication algorithm,” *IEEE Trans. Comput.*, vol. 22, no. 2, pp. 1045–1047, Dec. 1973.

[22] S. Purohit, M. Lanuzza, S. Perri, P. Corsonello, and M. Margala, “Design and evaluation of an energy-delay-area efficient data-path for coarse-grain reconfigurable computing systems,” *J. Low Power Electron.*, vol. 5, no. 3, pp. 326–338, 2009.

[23] S. Purohit, M. Lanuzza, S. Perri, P. Corsonello, and M. Margala, “Design-space exploration of energy-delay-area efficient coarse-grain reconfigurable data-path,” in *Proc. IEEE Int. Conf. VLSI Design*, Jan. 2009, pp. 45–50.

[24] S. Purohit, M. Lanuzza, and M. Margala, “Design space exploration of split-path data driven dynamic full adder,” *J. Low Power Electron.*, vol. 6, no. 4, pp. 469–481, 2010.

[25] S. Purohit and M. Margala, “Investigating the impact of logic and circuit implementation on full-adder performance,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 7, pp. 1327–1331, Jul. 2012.

[26] F. Frustaci, M. Lanuzza, P. Zicari, S. Perri, and P. Corsonello, “Low-power split-path data-driven dynamic logic,” *IET Circuits, Devices Syst.*, vol. 3, no. 6, pp. 303–312, Dec. 2009.

[27] W. Vanderbauwhede, S. Chalamalasetti, M. Margala, and S. Purohit, “A C++-embedded domain-specific language for programming the MORA soft processor array,” in *Proc. IEEE Conf. Appl.-Specific Syst. Arch. Process.*, 2010, pp. 141–148.

[28] S. Purohit, “Hardware software co-design of a resource efficient coarse grained reconfigurable architecture for high throughput media processing applications,” Ph.D. thesis, Univ. Massachusetts Lowell, Lowell, MA, 2011, pp. 1–179.

[29] C. Liou and H. Chiueh, “An ALU cluster with floating point unit for media streaming architecture with homogeneous processor cores,” in *Proc. IEEE 13th Asia-Pacific Comput. Syst. Arch. Conf.*, Aug. 2008, pp. 1–7.

[30] H. Oh, S. Mueller, C. Jacobi, K. Tran, S. Cottier, B. Micheal, H. Nishikawa, Y. Totsuka, T. Namatame, N. Yano, T. Machida, and S. Dhong, “A fully pipelined single-precision floating point unit in the synergistic processor element of a CELL processor,” *IEEE J. Solid-State Circuits*, vol. 41, no. 4, pp. 759–771, Apr. 2006.

[31] L. Huang, L. Shen, K. Dai, and Z. Wang, “A new architecture for multiple-precision floating point multiply-add fused unit design,” in *Proc. 18th IEEE Symp. Comput. Arith.*, 2007, pp. 69–76.

[32] *Xilinx FPU Documentation* [Online]. Available: http://www.xilinx.com/support/documentation/ip_document-ation/apu_fpu_virtex5.pdf

[33] N. Hockert and K. Compton, "FFPU: Fractured floating point unit for FPGA soft processors," in *Proc. Int. Conf. Field-Program. Technol.*, Dec. 2009, pp. 143–150.

[34] S. Liang, R. Tessier, and O. Mencer, "Floating point unit generation and evaluation for FPGAs," in *Proc. 11th Annu. IEEE Symp. Field-Program. Custom Comput. Mach.*, Apr. 2003, pp. 185–194.

[35] Nallatech Inc. (2001). *IEEE754 Floating Point Core*, Eldersburg, MD [Online]. Available: http://www.nallatech.com/products/ip/floating point virtex1/index.asp

Sohan S. Purohit (S'08–M'11) received the M.S. degree in electrical and computer engineering from the University of Rochester, Rochester, NY, and the Ph.D. degree in computer engineering from the University of Massachusetts Lowell, Lowell, in 2008 and 2011, respectively.

He is currently a Circuit Design Engineer with Intel Corporation, Austin, TX. He has authored or co-authored over 30 papers in peer-reviewed journals and international conferences and workshop proceedings, and one invited book chapter. His current research interests include high-speed digital circuit design, low-overhead radiation-hardened circuits and systems, high-performance arithmetic circuits and data-paths, reconfigurable computing, and THz circuits and systems design using novel non-CMOS devices.

Dr. Purohit is a reviewer for several journals and international conferences.

Sai Rahul Chalamalasetti (S'08) received the B.Tech. degree from the Koneru Lakshmaiah College of Engineering, Vaddeswaram, India, and the M.S. degree from the University of Massachusetts Lowell, Lowell, in 2007 and 2009, respectively, both in computer engineering.

He is a Research Associate Intern for HP Labs, Palo Alto, CA, and HP Houston, where he is involved in next generation low-power server-based architectures. He has authored or co-authored over ten publications in peer-reviewed journals and conference proceedings. His current research interests include reconfigurable architectures, novel data-path and processor architectures for low-power consumption, design of high-level programming tool chains for field-programmable gate arrays (FPGAs), and mapping massive data processing applications on FPGAs.

Martin Margala (S'92–M'98–SM'04) received the M.S. degree in microelectronics from Slovak Technical University, Bratislava, Slovakia, and the Ph.D. degree in electrical and computer engineering from the University of Alberta, Edmonton, AB, Canada, in 1990 and 1998, respectively.

He was with the University of Rochester, Rochester, NY, and the University of Alberta. From 1998 to 2003, he was an Adjunct Scientist with the Telecommunications Research Laboratory, Edmonton. He is currently a Professor and the Chair of the Electrical and Computer Engineering Department, University of Massachusetts Lowell, Lowell. He has authored or co-authored more than 160 publications in peer-reviewed journals and conference proceedings. He holds three patents. His current research interests include ballistic high-frequency devices and circuits, high-bandwidth data-processing architectures, and adaptive built-in-self-test systems.

Dr. Margala is a member of several program committees of many conferences and symposia in design and test.

Wim A. Vanderbauwhede (M'02) received the Ph.D. degree in electrotechnical engineering from the University of Gent, Gent, Belgium, in 1996.

He was a Mixed-Mode Design Engineer and Senior Technology R&D Engineer for Alcatel Microelectronics. He is currently a Lecturer with the School of Computing Science, University of Glasgow, Glasgow, U.K., where he joined in 2004. He has authored or co-authored over 90 papers in refereed journals and conferences. His current research interests include novel architectures for heterogeneous multicore systems and coarse-grained dynamically reconfigurable systems, and hardware acceleration for high-performance computing.