Variable Precision Arithmetic Circuits for FPGA-Based Multimedia Processors

Stefania Perri, Pasquale Corsonello, Maria Antonia Iachino, Marco Lanuzza, and Giuseppe Cocorullo

***Abstract*—This brief describes new efficient variable precision arithmetic circuits for field programmable gate array (FPGA)-based processors. The proposed circuits can adapt themselves to different data word lengths, avoiding time and power consuming reconfiguration. This is made possible thanks to the introduction of on purpose designed auxiliary logic, which enables the new circuits to operate in single instruction multiple data (SIMD) fashion and allows high parallelism levels to be guaranteed when operations on lower precisions are executed. The new SIMD structures have been designed to optimally exploit the resources of a widely used family of SRAM-based FPGAs, but their architectures can be easily adapted to any either SRAM-based or antifuse-based FPGA chips.**

***Index Terms*—Multimedia processors, multipliers, single instruction multiple data (SIMD).**

I. Introduction

Today multimedia applications are one of the major drivers for electronics products. Most multimedia tasks have high computational requirements, real-time throughput constraints and complex data-flows which vary significantly versus the specific application. Nowadays, systems-on-chip (SOCs) realized using the ASIC approach represent the performance optimized solution. Usually, one (or more) embedded processor (e.g., IP Core) is used inside these SOCs to produce a flexible computing platform able to support a large class of existing and future multimedia algorithms. However, the sequential instruction decoding and execution, and the fixed control architecture often limit the performance of such embedded processors [1]. Thus, to accomplish speed performance requirements the embedded processor has to be endowed with special units purposely designed to accelerate a certain class of applications [1], [2]. Just as an example, this is the case of the ARCtangent family of IP Core processors for which digital-signal processing (DSP) extensions (i.e., with SIMD instructions) are available for the design of multimedia SOCs. The ARCtangent-A4 reaches the maximum frequency of 200 MHz when synthesized using $0.18\square$ m libraries, whereas, the more recent and extensible ARCtangent-A5 runs up to 170 MHz using the same technology. MIPS, ARM, and OPENRISC soft-processors show similar characteristics and comparable speed performances.

Looking at the ASIC space in purely economic terms, the costs for a typical mask set for a modern technology run into the $ 500–700 K range. Thus, the ASIC design approach seems limited in viability to high-volume applications and big industries.

Modern field-programmable gate array (FPGA) devices are fast and offer a level of integration comparable to current application specific integrated circuits (ASICs). FPGAs are today largely used in multimedia applications [2]–[7], where a good tradeoff between costs, flexibility, performance and time-to-market plays a crucial role.

Manuscript received January 4, 2003; revised June 2, 2003; February 16, 2004.

S. Perri, M. Lanuzza, and G. Cocorullo are with the Dipartimento de Elettronica, Informatica E Sistemistica (D.E.I.S.), University of Calabria, Rende 87036, Italy (e-mail: perri@deis.unical.it).

P. Corsonello and M. A. Iachino are with the Department of Computer Science, Mathematics, Electronics and Transportation, University of Reggio Calabria, Reggio Calabria 89060, Italy (e-mail: corsonello@ing.unirc.it).

Digital Object Identifier 10.1109/TVLSI.2004.833400

Moreover, several embedded processors can be efficiently synthesized on FPGAs running up to 150 MHz and custom instructions can be added to these microprocessors in order to match algorithms requirements. As an example, in [8] custom logic has been added to the ALU of an ALTERA Nios to realize an MP3 decoder. ARCInternational, TENSILICA and XILINX embedded processors can exploit similar techniques using dedicated or fast-link busses.

The performance of multimedia processors is usually improved with the single instruction multiple data (SIMD) processing [9]–[13]. SIMD circuits should have the ability to dynamically vary their precision. However, often they are realized by trivially replicating computational processing elements with fixed functionality. This approach leads to a consistent waste of resources and power.

In this brief, efficient SIMD architectures for FPGA-based multimedia processors are presented. The new circuits here proposed for realizing variable precision arithmetic basic blocks run-time adapt their structures to different precisions at instruction level avoiding the need for FPGA reconfiguration. Obtained results demonstrate that this property not only saves consistent time and power due to reconfiguration, but also guarantees very high computational capabilities and an easy integration into field programmable systems-on-chip (FP-SOCs).

II. Motivation and Related Works

Several works exist in literature in which field programmable gate array (FPGA) devices either constitute or participate to multimedia processing units [3]–[5], [14]. Very recently, in [15] a first attempt to evaluate the impact of the introduction of SIMD instructions on the ALTERA Nios embedded processor has been carried out. There, it is clearly stated that to produce an effective advantage, such extensions have to be strictly additive and implemented in such a way that the extended processor has the same Instruction Set as the original one. Furthermore, in this way, original compiler and development tools can be used for the extended architecture.

For these reasons, variable precision modules that adapt their structures to different precisions avoiding the need for bit-stream downloading are very attractive. In fact, the alternative approach that exploits chip reconfiguration in order to change the processor functionality is very hard. As outlined in [14], it is well known that for initiating the reconfiguration, the need for it has to be firstly identified. Thus, to trigger reconfiguration either multimedia instructions have to be hardware decoded or special reconfiguration instructions inserted into the code have to be software processed. In the first case, purpose-designed complex instruction decoders that can deteriorate the overall performance are needed[1]. On the other hand, the software approach requires a purposely designed compiler that detects the need of reconfigurations introducing into the code reconfiguration instructions. They must be well scheduled to ensure that the appropriate configuration bit-stream is downloaded onto the FPGA device well before the specialized multimedia instructions are executed. In any case, the above approaches imply a consistent performance penalty which further increases if the reconfiguration time (e.g., in the order of milliseconds, for practical cases) cannot be hidden [14].

The circuits described in the following do not require any action either on the processor decoder or on the compiler. The user can easily configure, send data to and read data from the coprocessing unit by means of the "move-to" and "move-from" instructions available in any instruction set.

[1]Note that for most embedded processors, the instruction decoder cannot be accessed by the user. Thus, reconfiguration instructions cannot be processed in hardware.

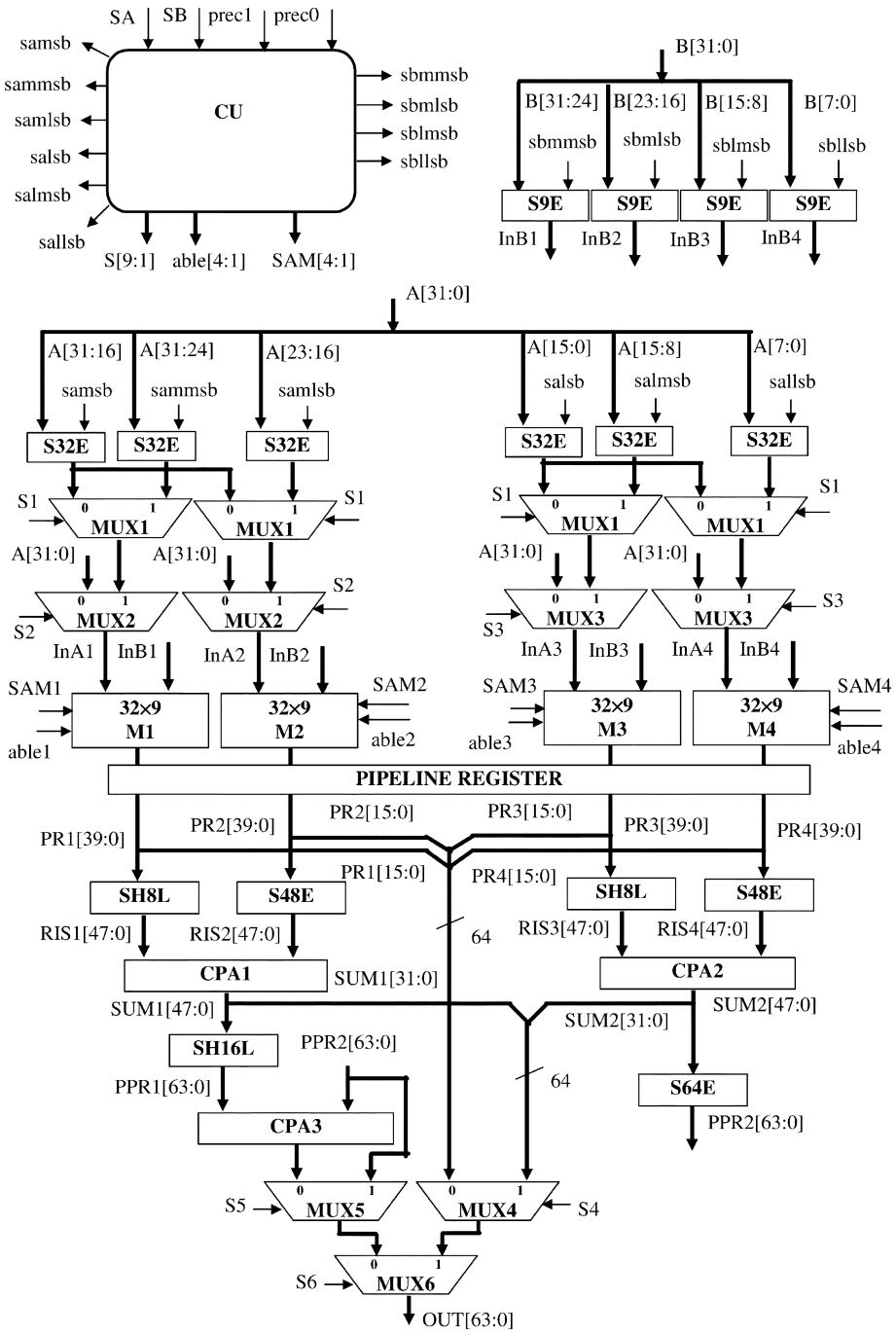

Fig. 1. Architecture of the new variable precision multiplier using four 32 × 9 multipliers.

III. New Variable Precision SIMD Signed/Unsigned Multipliers

In media signal processing (i.e., filtering, discrete cosine transforms, fast Fourier transforms, etc.), multiplication and multiply-accumulation are the most frequently required arithmetic operations. Moreover, multiplication often has the largest impact on the instruction cycle time of a whole system [1]. However, no efficient architectures for realizing variable precision SIMD multipliers have been yet proposed and optimized for FPGA-based processors. It is the goal of this Section.

Typically, a $N \times N$ variable precision SIMD multiplier is able to execute either one $N \times N$ multiplication or two $(N/2) \times (N/2)$ or four $(N/4) \times (N/4)$ multiplications and the obtained results are usually stored into a 2N-bit output register. The design of the new variable precision multipliers with $N = 32$ is based on the following considerations. $A_{[31:0]}$ and $B_{[31:0]}$ being two 32-bit operands, the following identities are easily verified:

$$A_{[31:0]} \times B_{[31:0]} = \mathrm{SH16L}\left(A_{[31:0]} \times B_{[31:16]}\right) + \mathrm{S64E}\left(A_{[31:0]} \times B_{[15:0]}\right) \quad (1)$$

$$A_{[31:0]} \times B_{[31:0]} = \left(A_{[31:16]} \times B_{[31:16]}\right) \mathrm{LINK} \left(A_{[15:0]} \times B_{[15:0]}\right) + \mathrm{SH16L}\left(\mathrm{S64E}\left(A_{[31:16]} \times B_{[15:0]}\right) + \mathrm{S64E}\left(A_{[15:0]} \times B_{[31:16]}\right)\right) \quad (2)$$

$$A_{[31:0]} \times B_{[31:0]} = \mathrm{SH16L}\left(\mathrm{SH8L}\left(A_{[31:0]} \times B_{[31:24]}\right) + \mathrm{S48E}\left(A_{[31:0]} \times B_{[23:16]}\right)\right) + \mathrm{S64E}\left(\mathrm{SH8L}\left(A_{[31:0]} \times B_{[15:8]}\right) + \mathrm{S48E}\left(A_{[31:0]} \times B_{[7:0]}\right)\right) \quad (3)$$

TABLE I
COMMANDS AND CONTROL SIGNALS FOR THE VARIABLE PRECISION ARCHITECTURE WITH FOUR 32 × 9 MULTIPLIERS

					sa		sb				SAM		able	s					
32×32	**prec1**	**prec0**	**SA**	**SB**	**msb=lsb**	**mmsb=mlsb=lmsb=llsb**	**mmsb**	**mlsb**	**lmsb**	**llsb**	**1=2**	**3=4**	**1 2**	**1**	**2**	**3**	**4**	**5**	**6**
u[1]×u	0	0	0	0	X[3]	X	0	0	0	0	0	0	1 1	X	0	0	X	0	0
u×s[2]	0	0	0	1	X	X	1	0	0	0	0	0	1 1	X	0	0	X	0	0
s×u	0	0	1	0	X	X	0	0	0	0	1	1	1 1	X	0	0	X	0	0
sxs	0	0	1	1	X	X	1	0	0	0	1	1	1 1	X	0	0	X	0	0
32×16																			
u×u	0	1	0	0	X	X	X	X	0	0	X	0	0 1	X	X	0	X	1	0
u×s	0	1	0	1	X	X	X	X	1	0	X	0	0 1	X	X	0	X	1	0
s×u	0	1	1	0	X	X	X	X	0	0	X	1	0 1	X	X	0	X	1	0
s×s	0	1	1	1	X	X	X	X	1	0	X	1	0 1	X	X	0	X	1	0
16×16																			
u×u	1	0	0	0	0	X	0	0	0	0	0	0	1 1	0	1	1	1	X	1
u×s	1	0	0	1	0	X	1	0	1	0	0	0	1 1	0	1	1	1	X	1
s×u	1	0	1	0	1	X	0	0	0	0	1	1	1 1	0	1	1	1	X	1
s×s	1	0	1	1	1	X	1	0	1	0	1	1	1 1	0	1	1	1	X	1
8×8																			
u×u	1	1	0	0	X	0	0	0	0	0	0	0	1 1	1	1	1	0	X	1
u×s	1	1	0	1	X	0	1	1	1	1	0	0	1 1	1	1	1	0	X	1
s×u	1	1	1	0	X	1	0	0	0	0	1	1	1 1	1	1	1	0	X	1
s×s	1	1	1	1	X	1	1	1	1	1	1	1	1 1	1	1	1	0	X	1

[1] unsigned; [2] signed; [3] don't care

where LINK, SHyL, and SzE indicate a concatenation action, a left shift by y bits, and an extension to z bits, respectively.

It is useful to recall that an n-bit number can be easily extended to $(n + h)$ bits considering that: 1) an n-bit unsigned number can be represented as a positive $(n + h)$-bit signed number having all the h additional bits set to 0 and 2) an n-bit signed number can be represented as a signed $(n + h)$-bit number having all the h additional bits equal to the nth.

Relations (1)–(3) indicate that a 32×32 multiplier generating a 64-bit output can be decomposed either into two 32×16 or into four 16×16 or into four 32×8 multipliers operating on 16 or 8-bit operands subwords. Architectures organized in these ways allow both high- and low-precision data to be handled and different parallelism levels to be obtained on the basis of the required precision.

In the following, new two-stage pipelined architectures are presented which allow variable precision SIMD multipliers to be realized efficiently, exploiting dedicated logic and fast routing resources available in an FPGA chip. The proposed schemes are able to perform signed–signed, signed–unsigned, and unsigned–unsigned SIMD multiplications on 32-, 16-, and 8-bit operands. Among the existing FPGA devices families the XILINX VIRTEX has been chosen as target. Thus, the proposed circuits have been optimized for it, but the described architectures can be easily optimized for many other FPGAs.

The first discussed architecture exploits (3) and it can execute one 32×32 or one 32×16 multiplication, or two 16×16 or four 8×8 operations, using four 32×9 multipliers. The latter have to be properly organized in order to manipulate both signed and unsigned operands also for the lower precisions. For the target FPGA devices family, an extremely efficient kind of automatically core generated macros exists which is useful to this purpose: the multiplier in which the data type of the operand A is controlled by pin whereas that of the operand B is fixed. In the new architectures, the operand B is always signed. Thus, in the following, this kind of multipliers is referenced as *a controlled by pin x signed* (CBPxS). An nxm CBPxS macro can be used to perform signed–signed, unsigned–unsigned, or signed–unsigned multiplications extending the operands to $(n+1)$ and to $(m+1)$ bits, respectively. The extension of the n-bit operand is internally executed by the macro itself on the basis of a control signal, which indicates whether that operand is a signed or an unsigned number. On the contrary, the m-bit operand has to be externally extended.

In Fig. 1, the block diagram of the new variable precision SIMD multiplier is depicted. It can be seen that its main building modules are: a control unit (CU) needed to properly set the computational flow on the basis of the required precision; four 32×9 CBPxS multipliers (M1, M2, M3, and M4) each generating 40-bit output;[2] two 48-bit and one 64-bit carry propagate adders (CPA1, CPA2, and CPA3).

The CU receives the signals prec1, prec0, SA, and SB as input and generates all the signals needed to orchestrate extensions and dataflow as summarized in Table I. The signal SA indicates if the operand A is a signed $(SA = 1)$ or an unsigned $(SA = 0)$ number. The same occurs for the operand B on the basis of SB. Conversely, the signals prec1 and prec0 serve to establish the required precision, as shown in Table I.

From Fig. 1, it can be seen that to correctly form the data InA_i and InB_i input to the multipliers M_i (for $i = 1, \ldots, 4$), the 16-bit and the 8-bit subwords of operand A are extended to 32 bits, whereas the 8-bit subwords of the operand B are extended to 9 bits. For any required precision, the extended subwords of operand *B* have to be input to the four multipliers. Conversely, for operand A, different conditions occur depending on the required precision. When operations on lower precision data have to be executed, the multiplexing blocks MUX1, MUX2, and MUX3 allow the extended subwords of operand A to be input to the multipliers. On the contrary, when the highest precision multiplication has to be executed those multiplexing blocks select the entire operand A as input for all the multipliers. In order to orchestrate the extension of the operands, subwords and the dataflow through the blocks MUX$_j$, with $j = 1, \ldots, 6$ the CU generates appropriate control signals. *samsb, sammsb, samlsb, salsb, salmsb*, and *sallsb* indicate if the modules for extension S32E have to treat the subwords $\mathrm{A}[31{:}16]$, $\mathrm{A}[31{:}24]$, $\mathrm{A}[23{:}16]$, $\mathrm{A}[15{:}0]$, $\mathrm{A}[15{:}8]$, and $\mathrm{A}[7{:}0]$, respectively, as signed or unsigned numbers. Analogously, *sbmmsb, sbmlsb, sblmsb*, and *sbllsb* indicate if the modules S9E must consider the subwords $\mathrm{B}[31{:}24]$, $\mathrm{B}[23{:}16]$, $\mathrm{B}[15{:}8]$, and $\mathrm{B}[7{:}0]$, respectively, as signed or unsigned numbers. The CU also generates the signals S_j and SAM$_i$. The former are used as the select input of the multiplexing blocks MUX$_j$, whereas the latter indicate if the data InA_i input to the multipliers M_i are signed or unsigned numbers.

[2]The additional bit of the sign-extended operand is internally used by the macro just as a control signal, thus, it does not increase the output wordlength [16].

TABLE II
CHARACTERISTICS OF THE PROPOSED VARIABLE-PRECISION CIRCUITS

	Delay [ns]	Power [mW/MHz]	Occupied Slices	Latency [cycles]	MOPS 8x8	MOPS 16x16	MOPS 32x16	MOPS 32x32
32x32 macro[4]	30.7	12.4	680	1	32.6	32.6	32.6	32.6
32x32 macro[5]	10.5	16.7	780	6	95	95	95	95
NVM 32x9[4]	23	19.5	1108	2	174	87	43.5	43.5
NVM 32x17	30.9	15.8	919	2	64.7	64.7	32.4	32.4
NVM 16x17	28.7	13.4	729	2	69.7	69.7	34.9	17.5
NVM 32x9[5]	9	25.5	1300	6, 7, 8	444.4	222.2	111.1	111.1

[4] Minimum pipeline; [5] Maximum pipeline;

The above control signals allow the required operation to be matched and the appropriate data flow to be set as summarized in Table I. When 32×32 multiplications are executed, the multipliers M1–M4 calculate in parallel the products between the operand A and the subwords $\mathrm{B}[31{:}24], \mathrm{B}[23{:}16], \mathrm{B}[15{:}8]$, and $\mathrm{B}[7{:}0]$ all extended to 9 bits. Then, the 40-bit results PR1 and PR3 are left shifted by 8-bit positions, whereas the 40-bit products PR2 and PR4 are extended to 48 bits. The two 48-bit words *Ris*1 and *Ris*2 obtained in this way are added by means of the carry-propagate adder CPA1, which generates the 48-bit word SUM1. At the same time, *Ris*3 and *Ris*4 are added by CPA2 forming the word SUM2. Then, SUM1 is left shifted by 16-bit positions, thus forming the 64-bit partial result *PPR*1, whereas SUM2 is extended to 64-bit, thus forming the partial result *PPR*2. To generate the final 64-bit result $\mathrm{OUT}[63:0]$, *PPR*1 and *PPR*2 are added by CPA3. It can be noted, that the hardware used could support the execution of two 32×16 multiplications. However, a 64-bit output register is used, thus only one 32×16 multiplication result can be accommodated in this register. When this operation is required just two multipliers must operate, whereas the others can be stopped. We have chosen to make the multipliers M3 and M4 able to go into action and to stop M1 and M2. To control this particular case the signals *able*1 and *able*2 are used. When *able*1 is low, all the 40 bits of PR1 and PR2 (and consequently, also the 48 bits of *Ris*1 and *Ris*2) are forced to zero. Therefore, the whole result $\mathrm{OUT}[63:0]$ is equal to *PPR*2, which is directly outputted by means of MUX5 and MUX6, without passing through CPA3.When operations on 16-bit data are executed, MUX4 and MUX6 allow the less significant 32-bit of the independent results SUM1 and SUM2 to be directly outputted without passing through CPA3. Analogously, when multiplications on 8-bit data are requested, MUX4 and MUX6 allow the less significant 16-bit of the independent results PR1, PR2, PR3, and PR4 to be directly outputted without passing through the addition modules CPA1, CPA2, and CPA3.

It is worth underlining that in order to optimally exploit logic and routing resources available in the target, FPGA appropriate placement constraints have been used. This allowed an extremely compact layout to be obtained. In the following, the SIMD multiplier described above will be named $\mathrm{NVM}_{32\times9}$ to indicate that it exploits four 32×9 multipliers.

Two alternative architectures of SIMD multipliers have also been implemented in this work. They are also able to execute one 32×32 or one 32×16 multiplication, or two 16×16, or two 8×8 multiplications. The first alternative structure is named $\mathrm{NVM}_{32\times17}$ to indicate that it uses two $32\times17\ \mathrm{CBP}\times\mathrm{S}$ multipliers, whereas the second one is named $\mathrm{NVM}_{16\times17}$ to indicate that it uses two $16 \times 17\ \mathrm{CBP} \times \mathrm{S}$ multipliers. It is worth pointing out that the latter version needs two clock cycles to complete the 32×32 multiplication.

IV. RESULTS

The new variable precision multiplier blocks described above have been implemented and characterized using a FPGA XILINX VIRTEX XCV400 device and the ISE Foundation Tool Version 4.2i. For purposes of comparison, we also realized a 32×32 conventional fixed-precision core-generated multiplier. It is worth noting that the core-generation tool does not allow the number of registers used in the multiplier pipeline to be arbitrarily chosen. Just minimum and maximum pipelining are allowed. To make an effective comparison both minimum and maximum pipelining have been considered. Table II summarizes simulation results obtained for the modules described in the previous sections. It can be seen that the architecture $\mathrm{NVM}_{32\times9}$ allows the number of MOPS on the lowest precision to be increased by $\sim434\%$ and $\sim84\%$, with respect its fixed precision counterpart with minimum and maximum pipelining, respectively. This is a nice property, since, as demonstrated in [5] in variable precision computations, the operations on lower precisions are the most frequently required. Moreover, performance of the new multipliers can be further improved by increasing the number of pipelining stages. The $\mathrm{NVM}_{32\times9}$ multiplier with maximum pipelining have been also realized and characterized. Obtained results are included in Table II. It can be seen that it operates at a running frequency higher than 110 MHz with 6–8 clock latency cycles when 8×8, 16×16, and 32×32 multiplications are executed, respectively.

Among the proposed SIMD architectures, the $\mathrm{NVM}_{32\times9}$ is the fastest solution, whereas the circuit $\mathrm{NMV}_{16\times17}$ represents the cheapest one. The latter allows the computational rate to be more than doubled for 8×8 and 16×16 multiplications with respect to the fixed-precision multiplier with minimum pipelining. But, the performance on 32×32 precision is reduced by $\sim46.3\%$. This is due to the fact that the circuit needs two clock cycles to complete that operation. Thus, capability at lower precisions is obtained at the expense of performance on the highest one.

The solution exploited in $\mathrm{NVM}_{32\times17}$ has characteristics falling between the two above-discussed circuits. It requires one clock cycle to complete multiplication on all precisions. Moreover, it doubles the number of MOPS on lower precision maintaining the performance on 32×32 multiplications. This circuit dissipates $\sim19\%$ less power and requires $\sim17\%$ fewer slices than $\mathrm{NVM}_{32\times9}$.

V. CONCLUSIONS

Efficient variable precision multipliers for FPGA-based multimedia processors have been presented. Three different structures for variable precision multiplication have been evaluated. Each one has shown its positive features that make it suitable for optimizing a particular parameter (i.e., delay, power, or area occupancy). The new circuits operate in SIMD fashion and are able to match computational demands for several precisions avoiding time and power consuming reconfiguration. This property makes the proposed circuits suitable also for antifuse-based FPGAs. It is worth pointing out that if more recent FPGA families (e.g., Virtex II) are chosen as target the $\mathrm{NVM}_{32\times9}$ multiplier runs well over 160 MHz.

ACKNOWLEDGMENT

The Authors are grateful to F. Pirrello for his invaluable help given during this project. They would also like to thank the anonymous reviewers for their precious suggestions.

REFERENCES

[1] K. Bondalapati and V. K. Prosanna, "Reconfigurable computing systems," *Proc. IEEE*, vol. 90, pp. 1201–1217, July 2002.

[2] J. Fritts, W. Wolf, and B. Liu, "Understanding multimedia application characteristics for designing programmable media processors," in *Proc. SPIE Photonics West, Media Processors*, San Jose, CA, Jan. 1999, pp. 2–13.

[3] S. McBader and P. Lee, "A programmable image signal processing architecture for embedded vision systems," in *Proc. 14th Int. Conf. Digital Signal Processing*, Santorini Island, Greece, July 2002.

[4] T. Miyamori and K. Olukotun, "REMARC: Reconfigurable multimedia array coprocessor," *IEICE Trans. Inform. Syst.*, vol. E82-D, no. 2, pp. 389–397, Feb. 2002.

[5] G. J. M. M. Bos, P. J. M. Havinga, S. J. Mullender, and J. Smit, "Chameleon—Reconfigurability in hand-held multimedia computers," in *Proc. 1st Int. Symp. Handheld Ubiquitous Computing*, Karlsruhe, Germany, Sept. 1999.

[6] S. Wong, J. Kester, M. Konstapel, R. Serra, and O. Visser, "Building blocks for MPEG stream processing," in *Proc. 13th Annu. Workshop on Circuits, Systems Signal Processing*, Veldhoven, The Netherlands, Nov. 2002.

[7] S. Wong, S. Vassiliadis, and S. Cotofana, "A sum of absolute differences implementation in FPGA hardware," in *Proc. IEEE Euromicro'02*, Dortmund, Germany, Sept. 2002.

[8] Author Anonymous, "Using the NIOS embedded processors, programmable logic, IP & SOC builder tools to construct custom microcontrollers," in *News & Views*, Altera Corporation, San Jose, CA, 1st quart. 2003.

[9] A. Peleg and U. Weiser, "MMX technology extension to the intel architecture," *IEEE Micro*, vol. 16, pp. 42–50, Aug. 1996.

[10] M. S. Schmookler, M. Putrino, C. Roth, M. Sharma, A. Mather, J. Tyler, H. Van Nguyen, M. N. Pham, and J. Lent, "A low-power, high-speed implementation of a powerPC™ microprocessor vector extension," in *Proc. 14th Arithmetic Conf.*, Adelaide, Australia, Apr. 1999.

[11] R. B. Lee, "Subword parallelism with MAX-2," *IEEE Micro*, vol. 16, pp. 51–59, Aug. 1996.

[12] A. Farooqui and V. G. Oklobdzija, "A programmable data-path for MPEG-4 and natural hybrid video coding," in *Proc. 34th Annu. Asilomar Conf. Signals, Systems, Computers*, Pacific Grove, CA, Oct. 29–Nov. 1 2000.

[13] M. Margala, J. X. Xu, and H. Wang, "Design verification and DFT for an embedded reconfigurable low-power multiplier in system-on-chip applications," in *Proc. 14th IEEE Application-Specific Integrated Circuit (ASIC)*, Arlington, TX, Sept. 2001.

[14] S. Wong, S. Cotofana, and S. Vassiliadis, "Coarse reconfigurable multimedia unit extension," in *Proc. 9th Euromicro Workshop on Parallel Distributed Processing*, Mantova, Italy, Feb. 2001.

[15] J. Probell, "Improving application performance with instruction set architecture extensions to embedded processors," in *Proc. DesignCon 2004*, Santa Clara, CA, Feb. 2004.

[16] K. Hwang, *Computer Arithmetic, Principles, Architecture, and Design*. New York: Wiley, 1979.