Flexible DSP Accelerator Architecture Exploiting Carry-Save Arithmetic

Kostas Tsoumanis, Sotirios Xydis, Georgios Zervakis, and Kiamal Pekmestzi

***Abstract*— Hardware acceleration has been proved an extremely promising implementation strategy for the digital signal processing (DSP) domain. Rather than adopting a monolithic application-specific integrated circuit design approach, in this brief, we present a novel accelerator architecture comprising flexible computational units that support the execution of a large set of operation templates found in DSP kernels. We differentiate from previous works on flexible accelerators by enabling computations to be aggressively performed with carry-save (CS) formatted data. Advanced arithmetic design concepts, i.e., recoding techniques, are utilized enabling CS optimizations to be performed in a larger scope than in previous approaches. Extensive experimental evaluations show that the proposed accelerator architecture delivers average gains of up to 61.91% in area-delay product and 54.43% in energy consumption compared with the state-of-art flexible datapaths.**

***Index Terms*— Arithmetic optimizations, carry-save (CS) form, datapath synthesis, flexible accelerator, operation chaining.**

I. Introduction

Modern embedded systems target high-end application domains requiring efficient implementations of computationally intensive digital signal processing (DSP) functions. The incorporation of heterogeneity through specialized hardware accelerators improves performance and reduces energy consumption [1]. Although application-specific integrated circuits (ASICs) form the ideal acceleration solution in terms of performance and power, their inflexibility leads to increased silicon complexity, as multiple instantiated ASICs are needed to accelerate various kernels. Many researchers have proposed the use of domain-specific coarse-grained reconfigurable accelerators [2]–[9] in order to increase ASICs' flexibility without significantly compromising their performance.

High-performance flexible datapaths [2], [4], [6], [7], [10] have been proposed to efficiently map primitive or chained operations found in the initial data-flow graph (DFG) of a kernel. The templates of complex chained operations are either extracted directly from the kernel's DFG [10] or specified in a predefined behavioral template library [4], [6], [7]. Design decisions on the accelerator's datapath highly impact its efficiency. Existing works on coarse-grained reconfigurable datapaths mainly exploit architecture-level optimizations, e.g., increased instruction-level parallelism (ILP) [2]–[5], [7]. The domain-specific architecture generation algorithms of [5] and [9] vary the type and number of computation units achieving a customized design structure. In [2] and [4], flexible architectures were proposed exploiting ILP and operation chaining. Recently, Ansaloni *et al.* [8] adopted aggressive operation chaining to enable the computation of entire subexpressions using multiple ALUs with heterogeneous arithmetic features.

Manuscript received July 25, 2014; revised October 16, 2014 and December 12, 2014; accepted January 8, 2015. Date of publication January 29, 2015; date of current version December 24, 2015.

The authors are with the Department of Electrical and Computer Engineering, National Technical University of Athens, Athens 106 82, Greece (e-mail: kostastsoumanis@microlab.ntua.gr; sxydis@microlab.ntua.gr; zervakis@microlab.ntua.gr; pekmes@microlab.ntua.gr).

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.2015.2390974

The aforementioned reconfigurable architectures exclude arithmetic optimizations during the architectural synthesis and consider them only at the internal circuit structure of primitive components, e.g., adders, during the logic synthesis [11]. However, research activities [12]–[14] have shown that the arithmetic optimizations at higher abstraction levels than the structural circuit one significantly impact on the datapath performance. In [12], timing-driven optimizations based on carry-save (CS) arithmetic were performed at the post-Register Transfer Level (RTL) design stage. In [13], common subexpression elimination in CS computations is used to optimize linear DSP circuits. Verma *et al.* [14] developed transformation techniques on the application's DFG to maximize the use of CS arithmetic prior the actual datapath synthesis. The aforementioned CS optimization approaches target inflexible datapath, i.e., ASIC, implementations. Recently, Xydis *et al.* [6], [7] proposed a flexible architecture combining the ILP and pipelining techniques with the CS-aware operation chaining. However, all the aforementioned solutions feature an inherent limitation, i.e., CS optimization is bounded to merging only additions/subtractions. A CS to binary conversion is inserted before each operation that differs from addition/subtraction, e.g., multiplication, thus, allocating multiple CS to binary conversions that heavily degrades performance due to time-consuming carry propagations.

In this brief, we propose a high-performance architectural scheme for the synthesis of flexible hardware DSP accelerators by combining optimization techniques from both the architecture and arithmetic levels of abstraction. We introduce a flexible datapath architecture that exploits CS optimized templates of chained operations. The proposed architecture comprises flexible computational units (FCUs), which enable the execution of a large set of operation templates found in DSP kernels. The proposed accelerator architecture delivers average gains of up to 61.91% in area-delay product and 54.43% in energy consumption compared to state-of-art flexible datapaths [4], [7], sustaining efficiency toward scaled technologies.

II. Carry-Save Arithmetic: Motivational Observations and Limitations

CS representation [15] has been widely used to design fast arithmetic circuits due to its inherent advantage of eliminating the large carry-propagation chains. CS arithmetic optimizations [12], [14] rearrange the application's DFG and reveal multiple input additive operations (i.e., chained additions in the initial DFG), which can be mapped onto CS compressors. The goal is to maximize the range that a CS computation is performed within the DFG. However, whenever a multiplication node is interleaved in the DFG, either a CS to binary conversion is invoked [12] or the DFG is transformed using the distributive property [14]. Thus, the aforementioned CS optimization approaches have limited impact on DFGs dominated by multiplications, e.g., filtering DSP applications.

In this brief, we tackle the aforementioned limitation by exploiting the CS to modified Booth (MB) [15] recoding each time a multiplication needs to be performed within a CS-optimized datapath. Thus, the

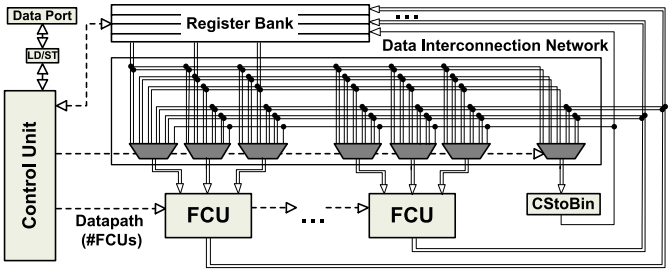

Fig. 1. Abstract form of the flexible datapath.

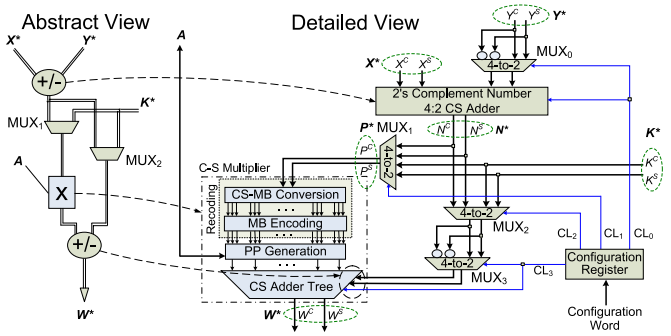

Fig. 2. FCU.

computations throughout the multiplications are processed using CS arithmetic and the operations in the targeted datapath are carried out without using any intermediate carry-propagate adder for CS to binary conversion, thus improving performance.

III. Proposed Flexible Accelerator

The proposed flexible accelerator architecture is shown in Fig. 1. Each FCU operates directly on CS operands and produces data in the same form[1] for direct reuse of intermediate results. Each FCU operates on 16-bit operands. Such a bit-length is adequate for the most DSP datapaths [16], but the architectural concept of the FCU can be straightforwardly adapted for smaller or larger bit-lengths. The number of FCUs is determined at design time based on the ILP and area constraints imposed by the designer. The CStoBin module is a ripple-carry adder and converts the CS form to the two's complement one. The register bank consists of scratch registers and is used for storing intermediate results and sharing operands among the FCUs. Different DSP kernels (i.e., different register allocation and data communication patterns per kernel) can be mapped onto the proposed architecture using post-RTL datapath interconnection sharing techniques [9], [17], [18]. The control unit drives the overall architecture (i.e., communication between the data port and the register bank, configuration words of the FCUs and selection signals for the multiplexers) in each clock cycle.

A. Structure of the Proposed Flexible Computational Unit

The structure of the FCU (Fig. 2) has been designed to enable high-performance flexible operation chaining based on a library of operation templates [4], [7]. Each FCU can be configured to any of the T1–T5 operation templates shown in Fig. 3. The proposed FCU enables intratemplate operation chaining by fusing the additions

[1]The FCU is able to operate on either CS or two's complement formatted operands, since a CS operand comprises two 2's complement binary numbers.

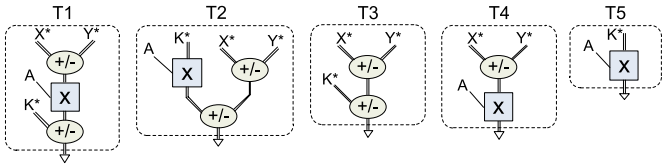

Fig. 3. FCU template library.

performed before/after the multiplication and performs any partial operation template of the following complex operations:

$$W^* = A \times (X^* + Y^*) + K^* \quad (1)$$

$$W^* = A \times K^* + (X^* + Y^*). \quad (2)$$

The following relation holds for all CS data: $X^* = \{X^C, X^S\} = X^C + X^S$. The operand A is a two's complement number. The alternative execution paths in each FCU are specified after properly setting the control signals of the multiplexers MUX_1 and MUX_2 (Fig. 2). The multiplexer MUX_0 outputs Y^* when $\text{CL}_0 = 0$ (i.e., $X^* + Y^*$ is carried out) or $\overline{Y^*}$ when $X^* - Y^*$ is required and $\text{CL}_0 = 1$. The two's complement 4:2 CS adder produces the $N^* = X^* + Y^*$ when the input carry equals 0 or the $N^* = X^* - Y^*$ when the input carry equals 1. The MUX_1 determines if N^* (1) or K^* (2) is multiplied with A. The MUX_2 specifies if K^* (1) or N^* (2) is added with the multiplication product. The multiplexer MUX_3 accepts the output of MUX_2 and its 1's complement and outputs the former one when an addition with the multiplication product is required (i.e., $\text{CL}_3 = 0$) or the later one when a subtraction is carried out (i.e., $\text{CL}_3 = 1$). The 1-bit ace for the subtraction is added in the CS adder tree.

The multiplier comprises a CS-to-MB module, which adopts a recently proposed technique [19] to recode the 17-bit P^* in its respective MB digits with minimal carry propagation. The multiplier's product consists of 17 bits. The multiplier includes a compensation method for reducing the error imposed at the product's accuracy by the truncation technique [20]. However, since all the FCU inputs consist of 16 bits and provided that there are no overflows, the 16 most significant bits of the 17-bit W^* (i.e., the output of the Carry-Save Adder (CSA) tree, and thus, of the FCU) are inserted in the appropriate FCU when requested.

B. DFG Mapping Onto the Proposed FCU-Based Architecture

In order to efficiently map DSP kernels onto the proposed FCU-based accelerator, the semiautomatic synthesis methodology presented in [7] has been adapted. At first, a CS-aware transformation is performed onto the original DFG, merging nodes of multiple chained additions/subtractions to 4:2 compressors. A pattern generation on the transformed DFG clusters the CS nodes with the multiplication operations to form FCU template operations (Fig. 3). The designer selects the FCU operations covering the DFG for minimized latency.

Given that the number of FCUs is fixed, a resource-constrained scheduling is considered with the available FCUs and CStoBin modules determining the resource constraint set. The clustered DFG is scheduled, so that each FCU operation is assigned to a specific control step. A list-based scheduler [21] has been adopted considering the mobility[2] of FCU operations. The FCU operations are scheduled according to descending mobility. The scheduled FCU operations are bound onto FCU instances and proper configuration bits are generated. After completing register allocation, a FSM is generated in order to implement the control unit of the overall architecture.

[2]Mobility: The ALAP-ASAP difference of the FCU operations.

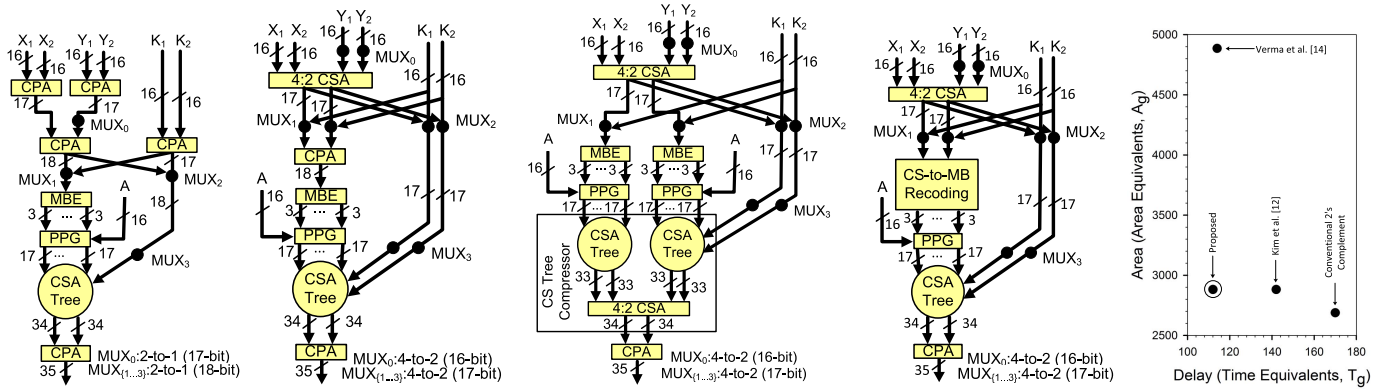

Fig. 4. Typical chaining of addition–multiplication–addition operations reflecting T1 template of Fig. 3. Its design is based on (a) two's complement arithmetic, (b) CS optimizations of [12], (c) CS optimizations with multiplication distribution [14], and (d) incorporating the CS-to-MB recoding concept. (e) Positioning of the proposed approach with respect to the two's complement one and the CS optimizations based on [12] and [14].

IV. Theoretical Analysis

In this section, we provide a theoretical analysis of the proposed approach based on the unit gate model[3] [22]. The critical template of the proposed FCU is the T1 of Fig. 3 and reflects an addition–multiplication–addition operation chaining (AMA_{DFG}). Fig. 4(a) shows the AMA_{DFG} when all operands are in two's complement form. Fig. 4(b) shows how [12] optimizes the AMA_{DFG}. Fig. 4(c) illustrates how [14] distributes the multiplication operation over the CS formatted data. The proposed approach in Fig. 4(d) incorporates the CS-to-MB recoding unit. We assume 16-bit input operands for all the designs and, without loss of generality, we do not consider any truncation concept during the multiplications. Fig. 4(e) shows the area-delay tradeoffs of all the alternative designs. As shown, the proposed design solution is the most effective among all the design alternatives.

V. Experimental Evaluation

A. Circuit-Level Exploration of the Proposed FCU With Respect to Technology Scaling

A circuit-level comparative study was conducted among the proposed FCU, the flexible computational component (FCC) of [4] and the reconfigurable arithmetic unit (RAU)[4] of [7] in scaled technology nodes. The CS representation requires twice the number of bits of the respective two's complement form, thus, increasing wiring and affecting performance in scaled technologies. This experimentation targets to show that the scaling impact on the performance does not eliminate the benefits of using CS arithmetic. The three units considered were described in RTL using Verilog. The CSA tree of the proposed FCU and the adders and multipliers of the FCC were imported from the Synopsys DesignWare library [11]. We used Synopsys Design Compiler [11] to synthesize the examined units and the TSMC 130, 90, and 65 nm standard cell libraries.[5] We synthesized each unit with the highest optimization degree at its critical clock period and 20 higher ones with a step interval of 0.10 ns.

Fig. 5 reports the area complexity of the evaluated units at 130, 90, and 65 nm of synthesis technology. At 130 nm, the proposed FCU, the FCC, and the RAU operate without timing violations starting at 2.98, 4.83, and 1.99 ns, respectively. At 90 nm, the proposed FCU, the FCC, and the RAU are timing functional starting at 1.66, 2.46, and 1.01 ns, respectively. In addition, at 65 nm, the proposed FCU, the FCC, and the RAU start operating without timing violations at 1.13, 1.68, and 0.67 ns, respectively. As the synthesis technology is scaled down, Fig. 5 shows that the proposed FCU outperforms the FCC in terms of critical delay and area complexity, but presents larger values for these metrics than the RAU in all the technology nodes. However, RAU's flexible pipeline stage (FPS) [7] features limited ability in carrying out heavy arithmetic operations as shown from the mega operations per second/watt (MOPS/W)[6] evaluation in Fig. 6.

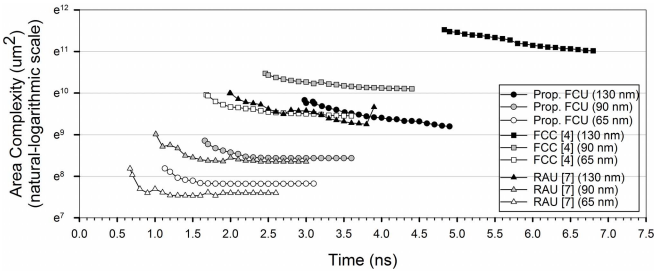

Fig. 5. Area-time diagram of FCUs.

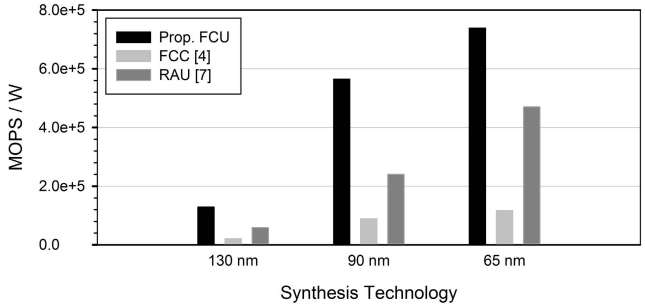

Fig. 6. MOPS/W values of FCUs at the lowest achievable clock periods with respect to the synthesis technology.

Fig. 6 shows the MOPS/W values for the proposed FCU, the FCC, and the RAU at their critical clock periods with respect to the synthesis technology. For each unit, we consider the templates

[3]The two-input gates NAND, AND, NOR, and OR count as one gate equivalent for area (A_g) and delay (T_g). The two-input XOR, XNOR gates count as two A_g and two T_g. The Full Adder (FA) and Half Adder (HA) area is seven and three A_g, respectively, while their critical delays are four and two T_g.

[4]In this section, we consider one flexible pipeline stage (FPS) of the RAU.

[5]The factor of technology scaling is ∼0.7 as the technology shrinks from 130 to 90 nm and from 90 to 65 nm.

[6]Mega Operations per Second/Watt: MOPS is defined as the number of operations multiplied by the clock frequency divided by the operation latency (=number of cycles), i.e., MOPS $= ((\#Ops)/(\#Cycles)) \times ClkFreq$.

TABLE I
EXECUTION LATENCY, AREA COMPLEXITY, AND ENERGY FOR DSP KERNELS MAPPED ONTO THE PROPOSED FCU, THE FCC, AND THE RAU AT 65 nm

Kernel	Prop. FCU			FCC [4]			RAU [7]			*A*×*L Gain*	*E Gain*	*A*×*L Gain*	*E Gain*
	L^a	A^b	E^c	*L*	*A*	*E*	*L*	*A*	*E*	*over FCC (%)*	*over FCC (%)*	*over RAU (%)*	*over RAU (%)*
ELLIPTIC	9.6	21636	198.72	13.2	41226	348.48	14.4	22042	234.72	61.83	42.98	34.56	15.34
FIR16	9.6	21150	182.40	17.6	17387	297.44	19.2	25975	391.68	33.65	38.68	59.29	53.43
VOLTERRA	8.0	19385	144.00	15.4	38299	264.88	25.2	21069	337.68	73.71	45.64	70.79	57.36
JPEGDCT	209.6	42803	5240.00	301.4	59897	7625.42	520.8	47588	14009.52	50.30	31.28	63.80	62.60
MPEG_IDCT	216.0	41525	3218.40	286.0	59382	6292.00	580.8	46136	11906.40	47.19	48.85	66.53	72.97
UDCT	212.8	17387	3553.76	281.6	40100	4111.36	625.2	25169	10128.24	67.23	13.56	76.49	64.91
Average	-	-	-	-	-	-	-	-	-	55.65	36.83	61.91	54.43

[a] Execution latency in ns. [b] Area complexity in um^2. [c] Energy consumption in pJ.

with the largest number of active operations without overlapping the one another and calculate the average ((*#Ops*)/(*#Cycles*)) factor. The *#Ops* derives from the two's complement arithmetic operations (additive or multiplicative). For CS-aware units, i.e., FCU and RAU, the CStoBin module runs in parallel with the FCU or RAU, thus counting as one additive operation. In particular, for the proposed FCU, we consider the templates T1 (or T2) with six operations (e.g., five operations from T1 and one operation from CStoBin) in one cycle and T3 with five operations in one cycle. Thus, $((\#Ops)/(\#Cycles))_{FCU} = 11/2$. For the FCC, we consider only the full load case of four operations in one cycle resulting in $((\#Ops)/(\#Cycles))_{FCC} = 4$. In addition, for the RAU, we consider the template of five additive operations that are carried out in one cycle, and the template of one multiplication that needs four cycles. Thus, $((\#Ops)/(\#Cycles))_{RAU} = 21/8$. The clock frequency ClkFreq is the highest one that each unit achieves (i.e., the critical clock period). The power values for computing the MOPS/W ones have been extracted by simulating each unit with Modelsim [23] for 2^{16} random inputs and using the Synopsys PrimeTime-PX [11] with the average calculation mode triggered. As shown, the proposed FCU delivers average MOPS/W gains across technology nodes of 6× and 2× over FCC and RAU, respectively. Thus, as the synthesis technology is scaled down, the benefits of the proposed FCU remain.

B. Mapping of DSP Kernels Onto the Proposed FCU-Based Architecture

We examined the efficiency of the proposed solution by mapping and accelerating DSP kernels onto the FCU datapath, that is: 1) a sixth-order ELLIPTIC filter [24]; 2) a 16-taps Finite Impulse Response filter (FIR16); 3) a nonlinear IIR VOLTERRA filter; 4) an 1-D unrolled Discrete Cosine Transform (DCT) kernel (UDCT); 5) the 2-D JPEG DCT (JPEGDCT) [25]; and 6) the 2-D Inverse MPEG DCT (MPEG_IDCT) [25]. The kernels were scheduled and mapped onto the architectures based on the proposed FCU, the FCC [4], and the RAU [7]. The proposed architecture comprises four FCUs and one CStoBin module, the FCC-based one contains two FCCs (= 8 ALUs + 8 Multipliers) [4], and the RAU-based architecture consists of four FPSs and one CStoBin module [7]. For each kernel mapped, the maximum memory bandwidth has been allocated between the local storage and the processing elements. All the datapaths were synthesized at 65 nm. The clock periods were specified at 1.60, 2.20, and 1.20 ns for the proposed FCU-, the FCC-, and the RAU-based architectures, respectively, considering the critical delays of Section V-A plus a slack of 0.50 ns for absorbing any delays inserted by the multiplexers and control units. Energy consumption values were calculated through PrimeTime-PX after simulating each datapath with Modelsim.

TABLE II
THEORETICALLY ESTIMATED EXECUTION LATENCY AND AREA COMPLEXITY FOR DSP KERNELS

Kernel	Prop. FCU		FCC [4]		RAU [7]	
	L^a	A^b	*L*	*A*	*L*	*A*
ELLIPTIC	258	14572	534	30056	360	27642
FIR16	258	14860	712	27176	480	26850
VOLTERRA	215	14092	623	28136	630	24666
JPEGDCT	5633	22420	12193	36806	13020	33912
MPEG_IDCT	5805	23572	11570	36806	14520	33768
UDCT	5719	14284	11392	29624	15630	28044

[a] Execution latency in time equivalents (T_g).
[b] Area complexity in area equivalents (A_g).

Table I reports the execution latency, area complexity, and energy values of the DSP kernels mapped onto the examined architectures. The execution latency is the total number of cycles multiplied by the clock period for the synthesis of each architecture. The average latency gains of the proposed FCU-based architecture over the ones built on the FCC and the RAU are 33.36% and 56.69%, respectively. Regarding the area complexity, the proposed FCU-based flexible datapath delivers average gains of 31.75% and 13.23% over the FCC- and RAU-based solutions, respectively. As shown, different kernels demand different area resources due to the differing needs regarding the number of scratch registers and multiplexers, as well as the complexity of the control unit (i.e., the more cycles a mapped kernel needs for execution, the more complex the control unit). Table I also reports the metrics of the area-delay product and the energy providing a clear view of the beneficial characteristics of the proposed approach and allowing us to safely conclude that the proposed architecture forms a very efficient solution for DSP acceleration.

Table II shows the theoretically estimated values for the execution latency and area complexity of the DSP kernels mapped onto the examined architectures. The analysis is based on the unit gate model as in Section IV. Regarding both the execution latency and the area complexity and considering all the DSP kernels, the proposed FCU-based architecture outperforms the ones built on the FCC and the RAU. As expected, the timing constraints and the effects of cell sizing implied by the Design Compiler synthesis tool, in some cases result in inconsistencies between the experimental and the theoretical studies, e.g., in Table I, the latency of ELLIPTIC kernel on FCC is more efficient than the one on RAU, but in Table II, the RAU-based ELLIPTIC kernel outperforms the one based on the FCC. In any case, both the experimental and theoretical analysis indicated that the proposed approach forms the most efficient architectural solution.

VI. Conclusion

In this brief, we introduced a flexible accelerator architecture that exploits the incorporation of CS arithmetic optimizations to enable fast chaining of additive and multiplicative operations. The proposed flexible accelerator architecture is able to operate on both conventional two's complement and CS-formatted data operands, thus enabling high degrees of computational density to be achieved. Theoretical and experimental analyses have shown that the proposed solution forms an efficient design tradeoff point delivering optimized latency/area and energy implementations.

References

[1] P. Ienne and R. Leupers, *Customizable Embedded Processors: Design Technologies and Applications*. San Francisco, CA, USA: Morgan Kaufmann, 2007.

[2] P. M. Heysters, G. J. M. Smit, and E. Molenkamp, "A flexible and energy-efficient coarse-grained reconfigurable architecture for mobile systems," *J. Supercomput.*, vol. 26, no. 3, pp. 283–308, 2003.

[3] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins, "ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix," in *Proc. 13th Int. Conf. Field Program. Logic Appl.*, vol. 2778. 2003, pp. 61–70.

[4] M. D. Galanis, G. Theodoridis, S. Tragoudas, and C. E. Goutis, "A high-performance data path for synthesizing DSP kernels," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 6, pp. 1154–1162, Jun. 2006.

[5] K. Compton and S. Hauck, "Automatic design of reconfigurable domain-specific flexible cores," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 5, pp. 493–503, May 2008.

[6] S. Xydis, G. Economakos, and K. Pekmestzi, "Designing coarse-grain reconfigurable architectures by inlining flexibility into custom arithmetic data-paths," *Integr., VLSI J.*, vol. 42, no. 4, pp. 486–503, Sep. 2009.

[7] S. Xydis, G. Economakos, D. Soudris, and K. Pekmestzi, "High performance and area efficient flexible DSP datapath synthesis," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 3, pp. 429–442, Mar. 2011.

[8] G. Ansaloni, P. Bonzini, and L. Pozzi, "EGRA: A coarse grained reconfigurable architectural template," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 6, pp. 1062–1074, Jun. 2011.

[9] M. Stojilovic, D. Novo, L. Saranovac, P. Brisk, and P. Ienne, "Selective flexibility: Creating domain-specific reconfigurable arrays," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 5, pp. 681–694, May 2013.

[10] R. Kastner, A. Kaplan, S. O. Memik, and E. Bozorgzadeh, "Instruction generation for hybrid reconfigurable systems," *ACM Trans. Design Autom. Electron. Syst.*, vol. 7, no. 4, pp. 605–627, Oct. 2002.

[11] [Online]. Available: http://www.synopsys.com, accessed 2013.

[12] T. Kim and J. Um, "A practical approach to the synthesis of arithmetic circuits using carry-save-adders," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 5, pp. 615–624, May 2000.

[13] A. Hosangadi, F. Fallah, and R. Kastner, "Optimizing high speed arithmetic circuits using three-term extraction," in *Proc. Design, Autom. Test Eur. (DATE)*, vol. 1. Mar. 2006, pp. 1–6.

[14] A. K. Verma, P. Brisk, and P. Ienne, "Data-flow transformations to maximize the use of carry-save representation in arithmetic circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 10, pp. 1761–1774, Oct. 2008.

[15] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford, U.K.: Oxford Univ. Press, 2000.

[16] G. Constantinides, P. Y. K. Cheung, and W. Luk, *Synthesis and Optimization of DSP Algorithms*. Norwell, MA, USA: Kluwer, 2004.

[17] N. Moreano, E. Borin, C. de Souza, and G. Araujo, "Efficient datapath merging for partially reconfigurable architectures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 7, pp. 969–980, Jul. 2005.

[18] S. Xydis, G. Palermo, and C. Silvano, "Thermal-aware datapath merging for coarse-grained reconfigurable processors," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2013, pp. 1649–1654.

[19] K. Tsoumanis, S. Xydis, C. Efstathiou, N. Moschopoulos, and K. Pekmestzi, "An optimized modified booth recoder for efficient design of the add-multiply operator," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 4, pp. 1133–1143, Apr. 2014.

[20] Y.-H. Chen and T.-Y. Chang, "A high-accuracy adaptive conditional-probability estimator for fixed-width booth multipliers," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 59, no. 3, pp. 594–603, Mar. 2012.

[21] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*, 1st ed. New York, NY, USA: McGraw-Hill, 1994.

[22] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Reading, MA, USA: Addison-Wesley, 2010.

[23] [Online]. Available: http://www.mentor.com/products/fv/modelsim/, accessed 2013.

[24] [Online]. Available: http://poppy.snu.ac.kr/CDFG/cdfg.html, accessed 2013.

[25] [Online]. Available: http://express.ece.ucsb.edu, accessed 2013.