A High-Speed, Energy-Efficient Two-Cycle Multiply-Accumulate (MAC) Architecture and Its Application to a Double-Throughput MAC Unit

Tung Thanh Hoang, *Student Member, IEEE*, Magnus Själander, *Member, IEEE*, and Per Larsson-Edefors, *Senior Member, IEEE*

***Abstract*—We propose a high-speed and energy-efficient two-cycle multiply-accumulate (MAC) architecture that supports two's complement numbers, and includes accumulation guard bits and saturation circuitry. The first MAC pipeline stage contains only partial-product generation circuitry and a reduction tree, while the second stage, thanks to a special sign-extension solution, implements all other functionality. Place-and-route evaluations using a 65-nm 1.1-V cell library show that the proposed architecture offers a 31% improvement in speed and a 32% reduction in energy per operation, averaged across operand sizes of 16, 32, 48, and 64 bits, over a reference two-cycle MAC architecture that employs a multiplier in the first stage and an accumulator in the second. When operating the proposed architecture at the lower frequency of the reference architecture the available timing slack can be used to downsize gates, resulting in a 52% reduction in energy compared to the reference. We extend the new architecture to create a versatile double-throughput MAC (DTMAC) unit that efficiently performs either multiply-accumulate or multiply operations for N-bit, $1 \times N/2$-bit, or $2 \times N/2$-bit operands. In comparison to a fixed-function 32-bit MAC unit, 16-bit multiply-accumulate operations can be executed with 67% higher energy efficiency on a 32-bit DTMAC unit.**

***Index Terms*—Arithmetic circuits, energy efficient, high speed, multiply-accumulate unit, variable wordlength.**

I. Introduction

THE multiply-accumulate (MAC) unit is a common digital block used extensively in microprocessors and digital signal processors for data-intensive applications. For example, many filters, orthogonal frequency-division multiplexing algorithms, and channel estimators require FIR or FFT/IFFT computations that MAC units can accelerate efficiently.

A basic MAC architecture consists of a multiplier and an accumulate adder organized as in Fig. 1. Inputs are fed to the multiplier, and successive products are summed by the accumulate adder. Multipliers are typically comprised of a partial-product unit (the PP unit) and a carry-propagate adder (the final adder).

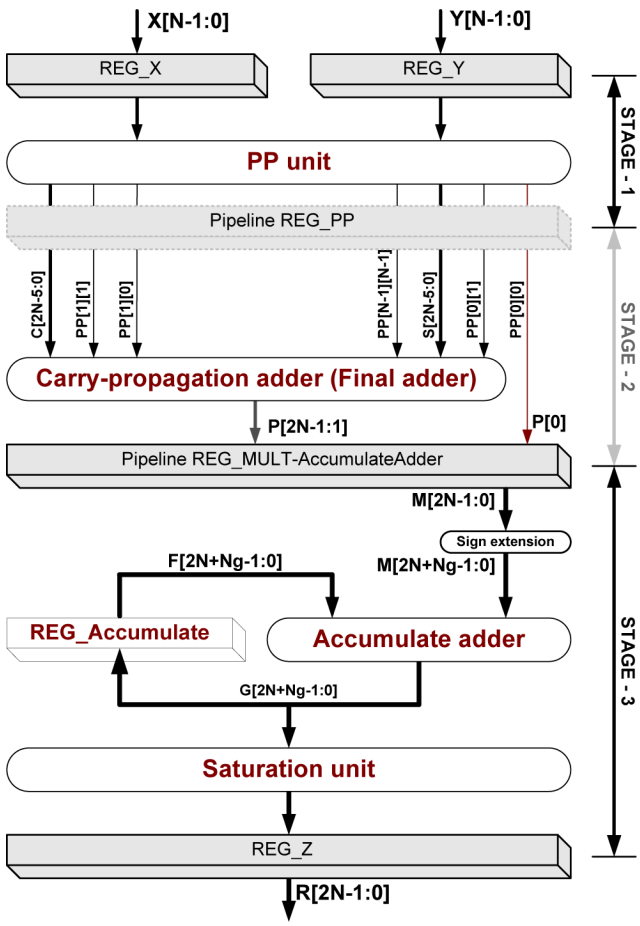

Fig. 1. Block diagram of a general MAC architecture. Here, the register between the PP unit and the final adder is removed/*included* to obtain a two/*three*-cycle MAC architecture.

To increase MAC performance, we can reduce the critical path delay by inserting an extra pipeline register, either inside the PP unit or between the PP unit and the final adder. This creates a three-cycle MAC architecture (Fig. 1), but increases overhead in terms of latency, energy, and area.

Much prior work focus on design techniques to reduce the multiplier delay, either in the PP unit or the final adder. Inside the PP unit, the partial-product circuitry might be implemented using the modified-Booth algorithm [1] or one of its successors [2]. The partial-product reduction tree of the PP unit can

Manuscript received April 02, 2010; revised August 04, 2010; accepted October 12, 2010. Date of current version December 15, 2010. This work was supported in part by VR, the Swedish Research Council, under Contract 2006-2927 and by the European Commission Framework Programme 7, Embedded Reconfigurable Architectures under Grant 249059.

The authors are with the Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Gothenburg, Sweden (e-mail: (hoangt@chalmers.se; hms@chalmers.se; perla@chalmers.se).

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/TCSI.2010.2091191

be implemented using high-speed compressors [3] or speed-optimized structures [4]. Mathew *et al.* propose a sparse-tree carry look-ahead adder for fast addition of the PP unit outputs [5], and Liu *et al.* introduce a hybrid adder [6] to reduce delay compared to a design that assumes equal arrival time on all adder inputs.

Performing two different carry propagations in the same MAC circuit is wasteful, since carry propagation is time consuming. Feeding the multiplier output back to the input of the PP unit reduction tree obviates the need for a conventional accumulate adder [7]–[9]. Accumulation is thus handled by the final adder of the multiplier, and only one carry-propagating stage is required. The problem is that this optimization only applies to one-cycle MACs, where the long critical delay is a limiting factor in most applications. If a pipeline register were to be inserted, the MAC output would no longer produce the correct result each cycle. In fact, to get the final result, we would have to add an extra, empty cycle after the final multiply-accumulate cycle of a loop. Furthermore, it is not obvious how guard bits can be accommodated in these designs. Guard bits are important for avoiding overflow when computing long sequences of multiply-accumulate operation. Ercegovac and Lang present a MAC architecture in which the multiplier's final adder is replaced by a stage of 4:2 compressors and guard bits are handled by an incrementer circuit [10]. However, this architecture only supports sign-magnitude numbers.

In general, two-cycle MAC architectures have a first (multiplication) stage that is significantly slower than the second (accumulation) stage. We propose a new two-cycle MAC architecture in which the second stage is somewhat slower, but the first stage is significantly faster, leading to a better delay balance between the two stages. The key to the new architecture is the implementation of product sign extension: the sign-extension circuitry is located in the second stage, together with the accumulate adder and the saturation unit. As a result, the feedback of the product is contained within the second pipeline stage.

The remainder of this paper is organized as follows: Section II describes our MAC architecture and contrasts it to a basic two-cycle architecture. Next, Section III provides an evaluation with respect to performance, power, energy, and area. The new MAC architecture has some features that enable us to design a unit that efficiently performs multiply-accumulate operations on different data operand sizes. Thus, as an extension to Section II, we introduce and evaluate the double-throughput MAC (DTMAC) architecture in Section IV. Finally, we conclude the paper in Section V.

II. Proposed MAC Architecture

The proposed two's complement MAC architecture [11] is shown in Fig. 2. Compared to the basic architecture in Fig. 1, the new design replaces the final adder in the first stage with a carry-save adder in the second. The MAC architecture's critical path delay still depends on the PP unit, but the delays of the two stages are now similar. The second stage remains faster, especially for larger operand sizes, which allows the accumulate adder to accommodate more guard bits.

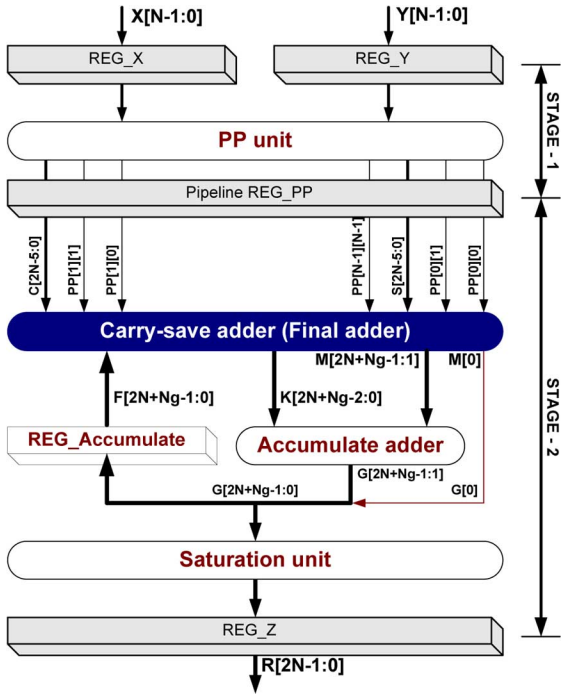

Fig. 2. Block diagram of the proposed MAC architecture.

Fig. 3 shows a basic multiply-accumulate operation using the Baugh–Wooley algorithm [12].[1] First we compute the product of the two inputs. Then this result is sign extended to have the same size as the accumulate adder. The accumulate adder is Ng bits wider than the multiplier to allow multiple (2^{Ng}) multiply-accumulate iterations without overflow. Finally, the sign-extended product is added to the stored accumulated value. The disadvantage here is that $P[2N-1]$ in Fig. 3 must be computed and used for sign extension in the accumulating addition.

In our architecture, we use a carry-save adder composed of full adders (3:2 counters). This adder (Fig. 4) sums the accumulated value with the two outputs from the PP unit's output registers. We avoid the complicated sign-extension procedure by adding the accumulated value to a row of $Ng+1$ bits of "1" (circled in Fig. 4) in the carry-save adder, much as in Hatamian and Cash's implementation of the Baugh–Wooley algorithm [14]. This removes the need to perform carry propagation inside the multiplication to obtain $P[2N-1]$ for sign extension. The product sign extension can instead be moved to the second stage.

Our MAC architecture offers a number of advantages in terms of latency, speed, area, power, and energy.

- If we compare to a two-cycle MAC (Fig. 1), the proposed MAC architecture needs no final adder.
- If we compare to a basic three-cycle MAC (Fig. 1), our architecture allows us to remove not only the final adder but also one pipeline register level—and the corresponding clock power—without degrading speed.
- Because our architecture is smaller, it uses shorter interconnects.

[1]A modified-Booth scheme such as that in Yeh and Jen [2] can be used, but offers no gain in terms of timing, and incurs significant power dissipation overhead [13].

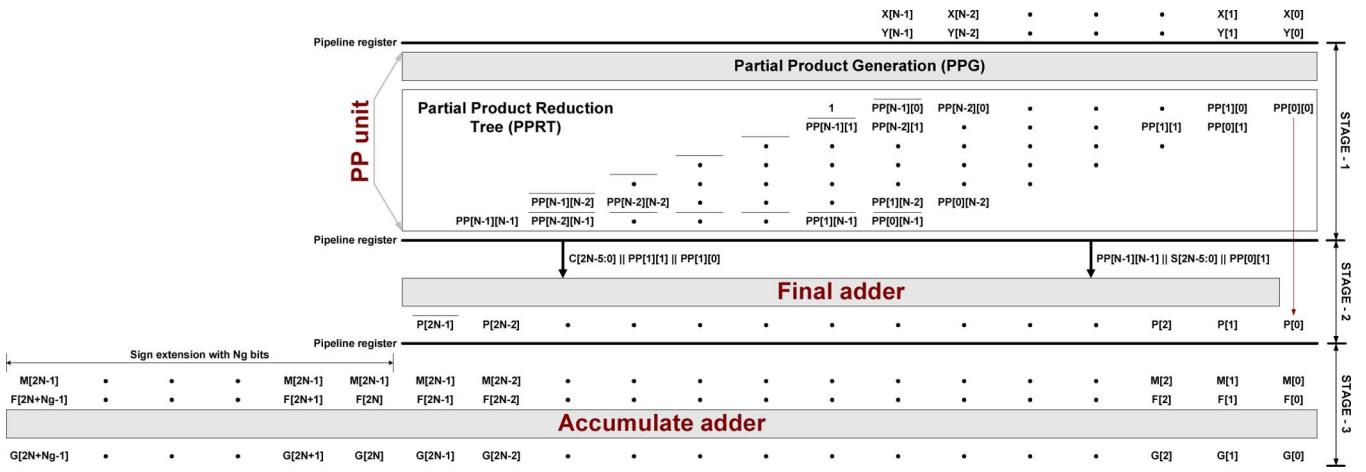

Fig. 3. A multiply-accumulate operation using inputs X and Y, assuming the three-cycle MAC architecture of Fig. 1. The multiply-accumulate operation starts with the generation (assuming the Baugh–Wooley algorithm) and reduction of partial products. The final adder performs carry propagation of the sums and carries produced by the PP unit. Finally, the accumulate adder sums the pipelined products (M) to the accumulated result (F), producing the new result (G).

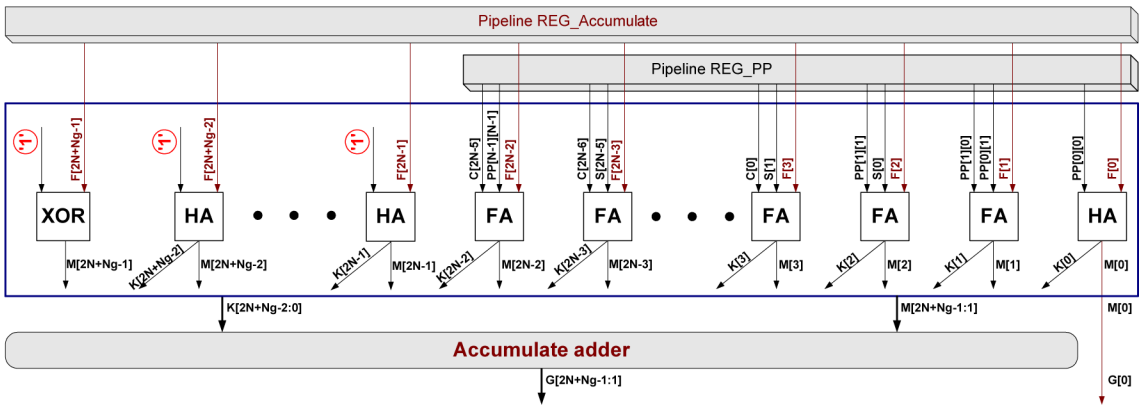

Fig. 4. Carry-save adder with product sign extension.

A saturation unit removes the guard bits (Ng) such that the final result is $2N$ bits wide (see Fig. 2). The saturation unit takes $G[2N+Ng-1:0]$ as input, where G is the output of the accumulate adder (as shown in Fig. 4), and implements the following algorithm:

```
IF G[2N+Ng-1:2N] = {Ng'bG[2N-1]} THEN
   # not saturated
   MAC_output[2N-1:0] = G[2N-1:0]
ELSE IF G[2N+Ng-1] = '1' THEN
   # set to the minimum negative 2N-bit value
   MAC_output[2N-1:0] = {1'b1, 2N-2'b0, 1'b1}
ELSE
   # set to the maximum positive 2N-bit value
   MAC_output[2N-1:0] = {1'b0, 2N-1'b1}
END
```

III. Evaluation

A. Evaluated Architectures

We consider three architectures that share the same structure for the PP unit, the final adder, and the accumulate adder.

- **MAC-2C** represents the two-cycle MAC whose critical path goes through the PP unit and the final adder (Fig. 1).
- **MAC-3C** represents the three-cycle MAC whose critical path lies entirely within the PP unit (Fig. 1).
- **MAC-NEW** represents our proposed two-cycle MAC whose critical path also lies entirely within the PP unit (Fig. 2).

B. Methodology

All PP units are based on the power-efficient Baugh–Wooley algorithm for partial-product generation [12], [13] and the HPM partial-product reduction tree [15]. The accumulate adder is of conditional-sum type [16] and has an extension of eight guard bits $(Ng = 8)$. This allows the MAC unit to support loops of up to 256 iterations without requiring the output to be right-shifted to avoid overflow. A final adder based on [17] supports fast addition of the PP unit outputs in the MAC-2C and MAC-3C.

Our VHDL descriptions for MAC-2C, MAC-3C, and MAC-NEW use registers on primary input and output. Each architecture is implemented for four different operand sizes; 16, 32, 48, and 64 bits. The VHDL blocks are synthesized using Synopsys Design Compiler and a commercial 65-nm standard-V_T library. Delay and power estimates are done for 1.1 V and the worst-case corner. We verify all netlists with logic simulation, and perform place-and-route in Cadence SoC Encounter. We extract the critical delays of individual units of each architecture with Synopsys PrimeTime. We estimate

TABLE I
EVALUATION RESULTS OF THREE ARCHITECTURES FOR 16, 32, 48, AND 64 BITS IN OPERAND SIZE

Operand size	Architecture	Delay (ps)				Clock (ps)	Power (mW)	Energy (pJ)	Area (μm^2)
		PP	Final adder	Final MAC stage*	Critical path delay				
16	MAC-2C	1317	597	1039	1914	2294	6.9	15.8	12937
	MAC-3C	1312	621	1039	1312	1692	10.3	17.4	13711
	MAC-NEW	1312	*N/A*	1247	1312	1692	8.2	13.8	12014
32	MAC-2C	1671	701	1110	2372	2794	17.1	47.7	42216
	MAC-3C	1664	711	1110	1664	2045	21.2	43.3	43545
	MAC-NEW	1664	*N/A*	1318	1664	2045	19.8	40.5	39357
48	MAC-2C	1976	884	1089	2860	3196	30.4	97.3	84894
	MAC-3C	1931	925	1089	1931	2310	38.3	88.5	86898
	MAC-NEW	1931	*N/A*	1297	1931	2310	32.3	74.5	82696
64	MAC-2C	2090	824	1242	2914	3279	54.6	179.0	149121
	MAC-3C	2082	848	1242	2082	2463	62.1	152.9	152179
	MAC-NEW	2082	*N/A*	1450	2082	2463	41.6	102.5	142942

(*) Here the delay of the final MAC stage is the cumulative delay of the extended accumulate adder and the saturation unit for MAC-2C and MAC-3C. For MAC-NEW, the delay of the carry-save adder is also included.

TABLE II
ENERGY PER OPERATION AT IDENTICAL CLOCK RATE AND TIMING CONSTRAINT

Operand size	Architecture	Power dissipation (mW)					Clock (ps)	Power (mW)	Energy (pJ)
		PP	Final adder	Carry-save adder	Acc. adder	FFs*			
16	MAC-2C	1.64	1.64	*N/A*	1.33	1.90	2294	6.52	15.0
	MAC-NEW	0.66	*N/A*	0.26	1.64	2.12		4.68	10.7
32	MAC-2C	6.60	6.32	*N/A*	2.23	3.04	2794	18.20	50.9
	MAC-NEW	3.47	*N/A*	0.39	3.07	3.69		10.62	29.7
48	MAC-2C	13.56	9.34	*N/A*	2.88	3.97	3196	29.75	95.1
	MAC-NEW	5.54	*N/A*	0.36	3.51	4.89		14.30	45.7
64	MAC-2C	27.58	17.23	*N/A*	4.32	5.77	3279	54.90	180.0
	MAC-NEW	11.76	*N/A*	0.47	4.58	6.83		23.64	77.5

(*) FFs (flip-flops) represents input, internal and output registers, as well as the clock tree.

power dissipation using a value change dump (VCD) analysis with 20 000 random test vectors and RC data extracted from SoC Encounter.

Performance and power evaluations are functions of the synthesis methodology. The "bottom-up" methodology used here exposes the architecture's impact on performance and avoids tool-driven optimization as much as possible: Each of the blocks (PP unit, final adder, and accumulate adder) is: *i)* synthesized with fairly strict timing constraint; *ii)* placed-and-routed under the synthesis timing constraint; and *iii)* used as a building block for all three architectures. Scripts control all steps of the design flow. We remove no artifacts of the heuristics used in the EDA tools, e.g., the small inconsistencies in delay trends.

C. Results and Discussion

Table I presents the results of our evaluation. Clearly, the critical path goes through the PP unit for all three implementations.[2] Since MAC-NEW uses pipeline registers after the PP unit, it obviously operates at the same speed as MAC-3C, while it is 31% faster than MAC-2C, on average. The delay difference between the first and second stages of MAC-NEW increases for larger operand sizes, so even more guard bits can be accommodated to support many multiply-accumulate iterations without performance degradation.

We use energy per operation to simultaneously capture power and performance for the considered architectures. Averaged across the four operand sizes, MAC-NEW dissipates 32% and 23% less energy than MAC-2C and MAC-3C, respectively. Furthermore, in terms of area, the proposed MAC architecture results in 4% and 7% smaller footprints, on average, than MAC-2C and MAC-3C, respectively.

[2]The critical path delay entry represents only the longest combinational logic delay, neglecting setup, hold, and propagation delays of surrounding registers.

D. Translating Timing Slack to Power Savings

The new MAC architecture intrinsically has a potential for higher speed than the basic two-cycle architecture. Thus, we investigate whether the available timing slack can be utilized to reduce power and energy. In the following, we use the existing timing slack for gate downsizing only, as this is a practical way to save power. Supply voltage reduction is clearly an alternative, but this requires the precise generation of a different voltage and incurs an overhead.

We now compare MAC-2C and MAC-NEW under equal timing constraints. This, for example, means that stage 1 of MAC-NEW is implemented with a timing constraint that corresponds to the critical delay of stage 1 in MAC-2C. Since the final adder is eliminated in MAC-NEW, its PP unit can fulfill the timing constraint using low-speed gates. On the other hand, since the MAC-NEW's accumulate adder is preceded by the new carry-save adder, its gates need to be upsized slightly to make the stage as fast as stage 2 of MAC-2C.

The delay profile of the PP unit is such that the bits in the center of the output word have the longest delays. We downsize the gates by iteratively examining output bits from the outside

toward the centre in order to balance the output delay for the PP unit while still meeting the timing constraint.

Table II shows the power dissipation of individual units for the two architectures[3] when they are operated at MAC-2C's maximal clock rate. In MAC-NEW, the removal of the final adder reduces power. However, since the PP unit only produces a partial product, there are $2N$ extra registers between stages one and two in Fig. 2, which increases the clock power compared to MAC-2C. More importantly, replacing the final adder with a carry-save adder leads to a 30% reduction of total power, on average. Furthermore, the removal of the final adder relaxes the timing constraint on the PP unit to the extent that its power drops 57% across the four operand sizes. This is significant, as the PP unit represents a big fraction of total power.

In summary, by downsizing gates in the PP unit, MAC-NEW dissipates an average of 52% less energy than MAC-2C for the same operating frequency and supply voltage. Downsizing thus offers a 29% additional reduction in energy per cycle over the MAC-NEW implementation in Section III-C that uses the same gate sizes as MAC-2C. Again, the numbers given here depend on the enforced timing constraint for the individual stages. For a more relaxed constraint, the utilization of slack in the faster architecture would be less effective.

IV. Extension to a Versatile MAC Unit

Adapting circuits to operate on the actual data precision of an application can save energy, as demonstrated in micro-processors [18] and dedicated circuits [19]. Many embedded applications are based on a 16-bit dynamic range, while embedded processors generally have a 32-bit datapath. Thus, potentially a 32-bit datapath could accommodate the execution of two simultaneous 16-bit operations. When the dynamic range of the data varies significantly across applications, run-time adaptation of the computational precision of a single circuit would be useful, rather than using several circuits that each has its own fixed operand size. Our previous work [13] shows that adding this kind of run-time adaptation to the multiplier of a general-purpose processor reduces execution time of an FFT application by 15%.

We refer to a MAC unit that can optionally switch between N-bit operations and $2 \times N/2$-bit operations as a double-throughput MAC unit (DTMAC). A 32-bit instance of such a MAC unit could be implemented by tying together two separate, 16-bit MAC units [20]. To support 32-bit operations the two 16-bit multipliers must be combined into one 32-bit multiplier, which requires complex routing and is difficult to implement efficiently. FPGA technology offers reconfigurability that can support double-throughput multiply-accumulate operations [21], but FPGAs are still inefficient in terms of speed and power compared to the ASIC solutions we consider here.

A. An Efficient Double-Throughput MAC Unit

A critical feature of any double-throughput MAC unit is that it should support several operating modes, without incurring any significant overheads on timing and power for the default $1 \times N$-bit mode. Thanks to the architecture introduced in Section II,

[3]This investigation uses a newer version of Encounter and thus the MAC-2C results differ slightly from those in Table I.

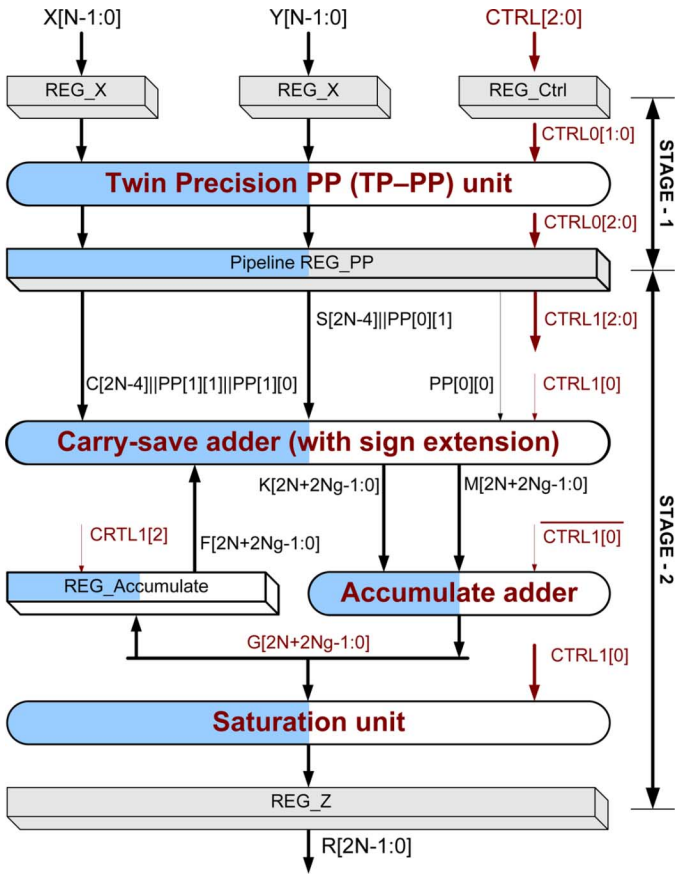

Fig. 5. The general architecture of the proposed DTMAC unit. $N = 32$ and $Ng = 8$ denotes the input operand width and the number of guard bits, respectively.

and the fact that product sign extension, accumulation, and carry propagation take place in the second stage of a two-cycle MAC unit, we can create an efficient DTMAC unit [22], see Fig. 5.

While other schemes, such as Kuang and Wang's scheme [23], may be used, our two's complement DTMAC unit employs the Twin-Precision (TP) technique [24]. A twin-precision partial-product reduction tree generates the TP-PP unit's outputs, as shown in Fig. 6(a), which in conventional schemes are fed to a final adder in order to obtain the final product. Instead, here we insert the proposed carry-save adder that sums the TP-PP unit outputs and the accumulate adder output according to Fig. 6(b). The output of the carry-save adder is fed to an accumulate adder that performs the carry propagation to produce the final result, as shown in Fig. 6(c).

As for conventional MACs, the TP-PP unit dominates the critical path delay. The DTMAC unit actually has the same critical delay as that of a basic three-cycle 32-bit MAC architecture, in which a pipeline register is inserted between the PP unit and the final adder to shorten the critical path of the multiplication. The result is that, despite the operating-mode flexibility, the DTMAC unit has small area requirements, low power dissipation, and short critical path delay.

B. Components of the DTMAC Unit

1) TP-PP Unit: To support double-throughput operations, the partial-product generation and reduction are based on the twin-precision (TP) technique [24]. Here, the partial products

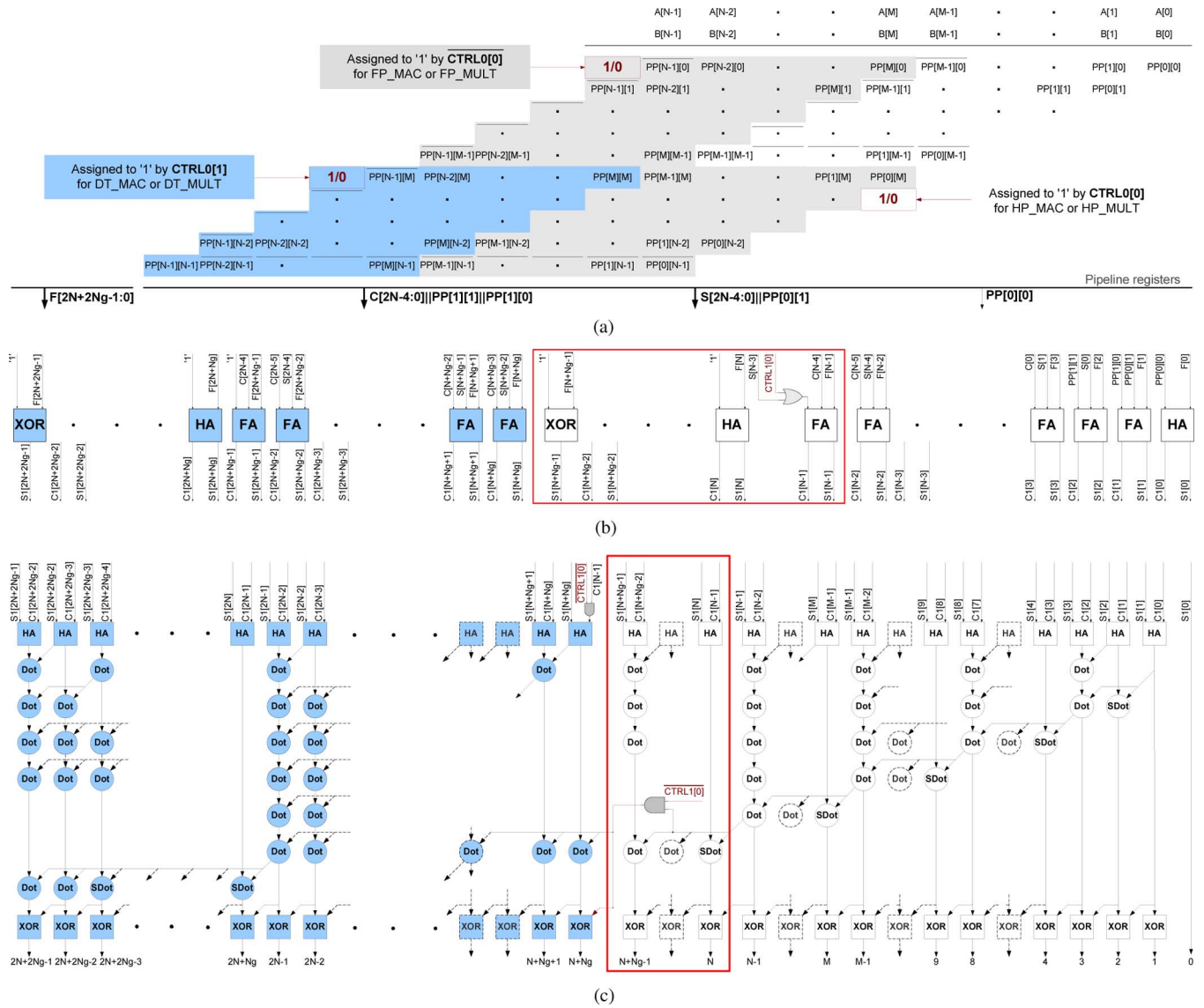

Fig. 6. Structure of the DTMAC unit's components: (a) The TP-PP unit, which is based on the Baugh–Wooley multiplication algorithm. (b) The carry-save adder. (c) The accumulate adder, which is based on the conditional-sum adder architecture. (Dark shaded and white colors denote computation in most significant and least significant $N/2 = M$ circuit sections, respectively.)

that are not needed during narrow-width operations are forced to zero [gray areas in Fig. 6(a)], while some lower-significance partial products are negated[4] to provide the correct function for the M-bit multiplication in the lower-significance section. Depending on the operating mode, "1" bits can be set in position $N+M$, N and M, as shown in Fig. 6(a).

From now on, we assume that $M = N/2$ and we call the lower-significance section the "low half."

2) Carry-Save Adder: The carry-save adder for the DTMAC unit mainly has the same function as the carry-save adder described in Section II. The difference here is that guard bits and sign extension for the $N/2$-bit operation in the low half must be accommodated [see framed gates in Fig. 6(b)]. This is achieved by inserting a row of $Ng+1$ bits of "1" that is summed together with the accumulated value and the most significant bit of the result from the TP-PP unit for the $N/2$-bit operation in the low half [$C[N-4]$ in bit position $N-1$ in Fig. 6(b)]. During $N/2$-bit operations in the low half, $S[N-3]$ will always be zero, due to the TP technique in which partial products are forced to zero. Since $S[N-3]$ will not carry any useful information during $N/2$-bit operations in the low half, this signal can be used to add the required "1" at bit position $N-1$. This is easily done by feeding $S[N-3]$ and a control signal through an extra OR gate, whose output may optionally be forced to "1," as shown in Fig. 6(b).

3) Accumulate Adder: The accumulate adder of the DTMAC unit is based on the conditional-sum adder structure [16], enabling efficient separation into high and low halves, each with Ng guard bits to avoid overflow. Fig. 6(c) frames the added guard bits for the $N/2$ operation in the low half have been framed. To control the operating mode, an AND gate is inserted; one control bit (CTRL1[0]) sets the AND's input at position $N+Ng$ either to zero or to the carry signal of the $N-1$-bit data part of the low half of the accumulate adder. For full precision operations, this effectively bypasses the Ng guard bits used for

[4]The negation is a function of operating mode.

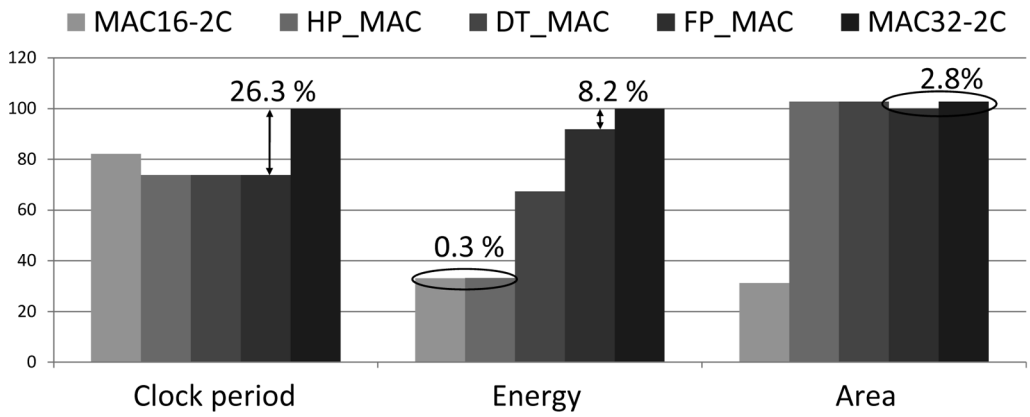

Fig. 7. Normalized values of clock period, energy per cycle and area for MAC16-2C, MAC32-2C, and DTMAC.

$N/2$-bit operations in the low half. Similarly, the accumulator output bits that correspond to unused guard bits ($F[N+Ng-1:N]$) are discarded during N-bit operation.

4) Saturation Circuit: The saturation unit for the DTMAC not only needs to consider full precision (N) operations but also the $N/2$ operations in the high and low halves.

- In full-precision mode, $2N + Ng$ bits ($[2N + 2Ng - 1 : N+Ng] : [N-1 : 0]$ in the output of the accumulate adder in Fig. 6(c)) are processed according to the algorithmic description in Section II.
- In half-precision mode, $N + Ng$ bits ($[N + Ng - 1 : 0]$) are processed according to the algorithmic description in Section II.
- In double-throughput mode, not only $N + Ng$ bits of the low half are processed, but also $N + Ng$ bits of the high half ($[2N+2Ng-1 : N+Ng]$) are processed in the same manner.

C. Operating Modes

The DTMAC unit supports six operating modes—three for multiply-accumulate operations and three for multiply operations—as determined by the value of the three-bit control signal (CTRL):

- **000**: Full-Precision 32-bit multiply-accumulate (FP_MAC).
- **001**: Half-Precision 1×16-bit multiply-accumulate (HP_MAC).
- **011**: Double-Throughput 2×16-bit multiply-accumulate (DT_MAC).
- **100**: Full-Precision 32-bit multiplication (FP_MULT).
- **101**: Half-Precision 1×16-bit multiplication (HP_MULT).
- **111**: Double-Throughput 2×16-bit multiplication (DT_MULT).

In Figs. 5 and 6, CRTL, CTRL0, and CTRL1 denote the three-bit control signal, its one-cycle delayed version, and its two-cycle delayed version, respectively. Moreover, in CTRL[2:0], CTRL[2] is the leftmost of the three bits and is used to force the output of the accumulate register to zero during multiply operations.

TABLE III
EVALUATION OF 32-BIT DTMAC UNIT

Architecture	Clock (ps)	Power (mW)	Energy (pJ)	Area (μm^2)
MAC16-2C	2294	6.90	15.83	13167
MAC32-2C	2794	17.10	47.78	42216
HP_MAC	2060	7.71	15.88	43397
DT_MAC	2060	15.62	32.18	43397
FP_MAC	2060	21.30	43.88	43397

D. Evaluation Methodology

We evaluate our design with a VHDL model of a 32-bit DTMAC unit. The DTMAC implementation is fully verified in logic simulation. We use MAC16-2C and MAC32-2C for comparison and the DTMAC implementation is synthesized using the same tool (Synopsys Design Compiler) and 65-nm cell library as in Section III. The implementation is placed-and-routed using SoC Encounter, and PrimeTime is used to find the critical path delay. Power dissipation is estimated through a VCD analysis on RC-extracted data from SoC Encounter using the same test vectors as for MAC32-2C.

E. Evaluation Results

Table III and Fig. 7 present the results of the evaluation. Thanks to the short critical path delay of the proposed MAC architecture, the 32-bit DTMAC unit can be used at 10% and 26% higher clock rates than conventional two-cycle 16-bit and 32-bit MAC units, respectively.

In terms of energy per cycle, when the DTMAC unit operates in 1×16-bit MAC mode it dissipates a negligible 0.3% more than the basic, fixed-function, 16-bit MAC unit. The DTMAC unit has a 2.8% larger footprint than MAC32-2C due to extra circuitry to support the multiple operation modes. These comparisons reveal that the implementation of operating-mode flexibility in the DTMAC unit comes at a limited overhead.

The important point is that we can save energy by adjusting the operating mode to the precision of the data:

- When the DTMAC unit operates in the default 32-bit MAC mode (FP_MAC), its energy dissipation is 8% lower than MAC32-2C when performing 32-bit computations.
- When the DTMAC unit operates in 1×16-bit MAC mode (HP_MAC), the 32-bit DTMAC unit performs 16-bit

multiply-accumulate operations 67% more energy efficiently than MAC32-2C performs computations on 16-bit operands. This reduction largely stems from avoiding unnecessary switching caused by the 16-bit sign extension of two's complement 32-bit data that carry only 16 bits of information.

- When the DTMAC unit operates in the 2×16-bit MAC mode (DT_MAC), its energy dissipation per 16-bit multiply-accumulate operation is similar to that of MAC16-2C. However, the DTMAC unit uses only half the cycles of MAC16-2C to compute all operations, so the surrounding datapath circuits are engaged for a significantly shorter time. This leads to significant energy savings for a system in which the DTMAC unit is integrated. In the next section, we give a brief account of an evaluation of this design scenario.

F. Double-Throughput Processor Datapath

In the context of a processor, the execution time reduction offered by the double-throughput modes may help save substantial amounts of energy. In order to evaluate the extent to wich the DTMAC unit can improve the execution of a C application, we integrated one such 32-bit unit into a 32-bit embedded FlexCore processor [22]. The FlexCore processor has a flexible datapath interconnect [25] that allows for accelerator extensions in a fairly straightforward manner, and it has a flexible instruction set that supports run-time reconfiguration.

We use two benchmarks from the EEMBC Telecom suite that make use of the multiply-accumulate operation in quite different ways: The auto-correlation benchmark (Autcor) contains many long sequences of 16-bit operations; the fast Fourier transform benchmark (FFT) has many short sequences of the same. A FlexCore datapath with a 32-bit DTMAC unit executes Autcor $4.37\times$ faster than a conventional five-stage datapath that lacks the DTMAC unit, but in exchange has a 32-bit integer multiplier. The accompanying reduction in energy dissipation was $4.00\times$, since the DTMAC unit incurs a small power dissipation overhead. The sequences of multiply-accumulate operations are quite short for the FFT benchmark, so the computational efficiency of the dedicated MAC accelerator drops. Still, the datapath equipped with the DTMAC unit executes $1.82\times$ faster than the reference, leading to a $1.64\times$ reduction in energy.

V. Conclusion

We describe a new high-speed, energy-efficient two's complement, two-cycle multiply-accumulate (MAC) architecture. Replacing the final adder of the multiplier by a carry-save adder with a new sign extension technique makes our two-cycle MAC architecture faster and more area- and energy-efficient than a basic two-cycle MAC architecture. Our evaluation for a commercial 65-nm 1.1-V cell library shows that the new architecture computes 31% faster and reduces energy per operation by 32%, on average. The timing slack difference allows us to downsize gates so that our new MAC architecture dissipates half the energy of the reference architecture.

We use the new architecture to develop a versatile MAC unit that supports several different operating modes: three for multiply-accumulate operations and three for multiply operations. We show that a 32-bit DTMAC unit can perform 16-bit multiply-accumulate operations at one third of the energy of a fixed-function, 32-bit architecture with the same cycle count. In double-throughput mode, executing two concurrent 16-bit multiply-accumulate operations delivers high energy efficiency. Deploying our design in a processor datapath can yield significant speed and energy impact for applications that compute many 16-bit multiply-accumulate operations.

References

[1] O. L. MacSorley, "High-speed arithmetic in binary computers," *Proc. Inst. Radio Eng. (IRE)*, vol. 49, pp. 67–91, Jan. 1961.

[2] W.-C. Yeh and C.-W. Jen, "High-speed booth encoded parallel multiplier design," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 692–701, Jul. 2000.

[3] M. R. Santoro and M. A. Horowitz, "SPIM: A pipeline 64×64 bit iterative multiplier," *IEEE J. Solid-State Circuits*, vol. 2, no. 1, pp. 487–493, Apr. 1989.

[4] V. G. Oklobdzija, D. Villeger, and S. S. Liu, "A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach," *IEEE Trans. Comput.*, vol. 45, no. 3, pp. 294–306, Mar. 1996.

[5] S. K. Mathew, M. A. Anders, B. Bloechel, T. Nguyen, R. K. Krishnamurthy, and S. Borkar, "A 4-GHz 300-mW 64-bit integer execution ALU with dual supply voltages in 90-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 40, no. 1, pp. 44–51, Jan. 2005.

[6] J. Liu, S. Zhou, H. Zhu, and C.-K. Cheng, "An algorithmic approach for generic parallel adders," in *Proc. IEEE Int. Conf. Comput. Aided Des. (ICCAD)*, Dec. 2003, pp. 734–740.

[7] P. F. Stelling and V. G. Oklobdzija, "Implementing multiply-accumulate operation in multiplication time," in *Proc. Int. Symp. Comput. Arithmetic (ARITH)*, July 1997, pp. 99–106.

[8] J. Großschädl and G.-A. Kamendje, "A single-cycle $(32 \times 32 + 32 + 64)$-bit multiply/accumulate unit for digital signal processing and public-key cryptography," in *Proc. IEEE Int. Conf. Electron., Circuits, Syst. (ICECS)*, Dec. 2008, pp. 739–742.

[9] A. Abdelgawad and M. Bayoumi, "High speed and area-efficient multiply accumulate (MAC) unit for digital signal processing applications," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2007, pp. 3199–3202.

[10] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. San Mateo, CA: Morgan Kaufmann, 2003.

[11] T. T. Hoang, M. Själander, and P. Larsson-Edefors, "High-speed, energy-efficient 2-cycle multiply-accumulate architecture," in *Proc. IEEE Int. SOC Conf. (SOC)*, Sep. 2009, pp. 119–122.

[12] C. R. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Trans. Comput.*, vol. C-22, pp. 1045–1047, Dec. 1973.

[13] M. Själander and P. Larsson-Edefors, "Multiplication acceleration through twin precision," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, pp. 1233–1246, Sep. 2009.

[14] M. Hatamian and G. L. Cash, "A 70-MHz 8-bit $\times$ 8-bit parallel pipelined multiplier in 2.5-μm CMOS," *IEEE J. Solid-State Circuits*, vol. JSSC-21, no. 4, pp. 505–513, 1986.

[15] H. Eriksson, P. Larsson-Edefors, M. Sheeran, M. Själander, D. Johansson, and M. Schölin, "Multiplier reduction tree with logarithmic logic depth and regular connectivity," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2006, pp. 4–8.

[16] J. Sklansky, "Conditional-sum addition logic," *IRE Trans. Electronic Comput.*, vol. EC-9, pp. 226–231, 1960.

[17] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. Comput.*, vol. C-22, no. 8, pp. 786–193, Aug. 1973.

[18] D. Brooks and M. Martonosi, "Dynamically exploiting narrow width operands to improve processor power and performance," in *Proc. Int. Symp. High-Perform. Comput. Archit.*, 1999, pp. 13–22.

[19] S. Yoshizawa and Y. Miyanaga, "Use of a variable wordlength technique in an OFDM receiver to reduce energy dissipation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 9, pp. 2848–2859, Oct. 2008.

[20] R. K. Kolagotla, J. Fridman, B. C. Aldrich, M. M. Hoffman, W. C. Anderson, M. S. Allen, D. B. Witt, R. R. Dunton, and L. A. Booth, "High performance dual-MAC DSP architecture," *IEEE Signal Process. Mag.*, vol. 19, no. 4, pp. 42–53, Jul. 2002.

[21] S. Hong and S.-S. Chin, "Reconfigurable embedded MAC core design for low-power coarse-grain FPGA," *Electron. Lett.*, vol. 39, no. 7, pp. 606–608, Apr. 2003.

[22] T. T. Hoang, M. Själander, and P. Larsson-Edefors, "Double throughput multiply-accumulate unit for FlexCore processor enhancements," presented at the IEEE Int. Symp. Parallel Distrib. Process. (IPDPS), Reconfigurable Archit. Workshop (RAW), Rome, Italy, May 2009.

[23] S.-R. Kuang and J.-P. Wang, "Design of power-efficient configurable booth multiplier," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 3, pp. 568–580, Mar. 2010.

[24] M. Själander, H. Eriksson, and P. Larsson-Edefors, "An efficient twin-precision multiplier," in *Proc. IEEE Int. Conf. Comput. Des. (ICCD)*, Oct. 2004, pp. 30–33.

[25] M. Thuresson, M. Själander, M. Björk, L. Svensson, P. Larsson-Edefors, and P. Stenstrom, "FlexCore: Utilizing exposed datapath control for efficient computing," *Springer J. Signal Process. Syst.*, vol. 57, no. 1, pp. 5–19, Oct. 2009.

Tung Thanh Hoang received the B.S. degree in electronic engineering from Hanoi University of Science and Technology, Hanoi, Vietnam, in 2003 and the M.Sc. degree in electrical engineering from Korea University, Seoul, in 2007. He is currently working toward the Ph.D. degree at the Department of Computer Science and Engineering, Chalmers University of Technology, Sweden.

His research interests are in the areas of high-performance, low-power digital circuits, and its application in embedded systems.

Magnus Själander received the M.Sc. degree in computer science and engineering from Luleå University of Technology, Sweden, in 2003 and the Ph.D. degree in computer engineering from Chalmers University of Technology, Göteborg, Sweden, in 2008.

He currently holds a postdoctoral position at Chalmers University of Technology. His research interests range from high-speed and low-power digital circuits to complete systems and the interaction between hardware and software.

Per Larsson-Edefors received the M.Sc. degree in electrical engineering and engineering physics and the Ph.D. degree in electronic devices from Linköping University, Sweden, in 1991 and 1995, respectively.

He was a Visiting Scientist at National Microelectronics Research Center, Ireland, 1996–1997, and a Visiting Professor at Intel Corporation's Circuit Research Lab in 2000. He currently holds the Chair of Computer Engineering at Chalmers University of Technology, Gothenburg, Sweden. He has published more than 80 papers in international conferences and journals, and served on many technical program committees, such as ESSCIRC and Symposium on VLSI Circuits. His research interests range from low-power high-performance digital circuits to design methodologies for energy-efficient embedded processors.