Multiplication Acceleration Through Twin Precision

Magnus Själander and Per Larsson-Edefors, *Senior Member, IEEE*

***Abstract*—We present the twin-precision technique for integer multipliers. The twin-precision technique can reduce the power dissipation by adapting a multiplier to the bitwidth of the operands being computed. The technique also enables an increased computational throughput, by allowing several narrow-width operations to be computed in parallel. We describe how to apply the twin-precision technique also to signed multiplier schemes, such as Baugh–Wooley and modified-Booth multipliers. It is shown that the twin-precision delay penalty is small (5%–10%) and that a significant reduction in power dissipation (40%–70%) can be achieved, when operating on narrow-width operands. In an application case study, we show that by extending the multiplier of a general-purpose processor with the twin-precision scheme, the execution time of a Fast Fourier Transform is reduced with 15% at a 14% reduction in datapath energy dissipation. All our evaluations are based on layout-extracted data from multipliers implemented in 130-nm and 65-nm commercial process technologies.**

***Index Terms*—Area efficient, Baugh–Wooley multiplier, high speed, low power, modified-Booth multiplier, SIMD, twin-precision.**

I. Introduction

MULTIPLICATION is a complex arithmetic operation, which is reflected in its relatively high signal propagation delay, high power dissipation, and large area requirement. When choosing a multiplier for a digital system, the bitwidth of the multiplier is required to be at least as wide as the largest operand of the applications that are to be executed on that digital system. The bitwidth of the multiplier is, therefore, often much larger than the data represented inside the operands, which leads to unnecessarily high power dissipation and unnecessary long delay. This resource waste could partially be remedied by having several multipliers, each with a specific bitwidth, and use the particular multiplier with the smallest bitwidth that is large enough to accommodate the current multiplication. Such a scheme would assure that a multiplication would be computed on a multiplier that has been optimized in terms of power and delay for that specific bitwidth. However, using several multipliers with different bitwidths would not be an efficient solution, mainly because of the huge area overhead.[1]

There have been several studies on operand bitwidths of integer applications and it has been shown that for the SPECint95 benchmarks more than 50% of the instructions are instructions where both operands are less than or equal to 16 bits [1] (henceforth called narrow-width operations). This property has been explored to save power, through operand guarding [1]–[3]. In operand guarding the most significant bits of the operands are not switched, thus power is saved in the arithmetic unit when multiple narrow-width operations are computed consecutively. Brooks *et al.* [1] showed that power dissipation can be reduced by gating of the upper part of narrow-width operands. For the SPECint95 and MediaBench benchmarks, the power reduction of an operand-guarded integer unit was 54% and 58%, respectively, which accounts for a total power reduction of 5–6% for an entire datapath.

Narrow-width operands have also been used to increase instruction throughput, by computing several narrow-width operations in parallel on a full-width datapath. Loh [4] showed a 7% speedup for the SPECint2000 benchmarks by using a simple 64-bit ALU, which excluded the multiplier, in parallel with four simple 16-bit ALUs that share a 64-bit routing. Brooks *et al.* [1] did a similar investigation, where they envisioned a 64-bit adder that could be separated into four 16-bit adders by severing the carry chain.

There have been several studies on operand guarding for multipliers. Huang *et al.* [2] introduced two-dimensional operand guarding for array multipliers, resulting in a power dissipation that was only 33% of a conventional array multiplier. Han *et al.* [3] did a similar investigation on a 32-bit Wallace multiplier and were able to reduce the switching activity by 72% with the use of 16-bit operand guarding.

While there have been a lot of work on simple schemes for operand guarding, work that simultaneously considers multiplication throughput is more scarce. Achieving double throughput for a multiplier is not as straightforward as, for example, in an adder, where the carry chain can be cut at the appropriate place to achieve narrow-width additions. It is of course possible to use several multipliers, where at least two have narrow bitwidth, and let them share the same routing, as in the work of Loh, but such a scheme has several drawbacks: i) The total area of the multipliers would increase, since several multiplier units are used. ii) The use of several multipliers increases the fanout of the signals that drive the inputs of the multipliers. Higher fanout means longer delays and/or higher power dissipation. iii) There would be a need for multiplexers that connect the active multiplier(s) to the result route. These multiplexers would be in the critical path, increasing total delay as well as power dissipation. Work has been done to use 4:2-reduction stages to combine small tree multipliers into larger multipliers [5], [6]. This can be done in several stages, creating a larger multiplier out of smaller for each extra 4:2 reduction stage. The desired bitwidth of the multiplication is then obtained by using multiplexers, which has a negative effect on the delay.

We present the twin-precision technique [7] that offers the same power reduction as operand guarding *and* the possi-

Manuscript received November 22, 2007; revised April 03, 2008. First published May 05, 2009; current version published August 19, 2009. This work was supported by the Swedish Foundation for Strategic Research (SSF) through the FlexSoC project.

The authors are with the Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden

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.2008.2002107

[1]There is also a potential static power overhead of inactive multipliers, but this could be alleviated by power gating.

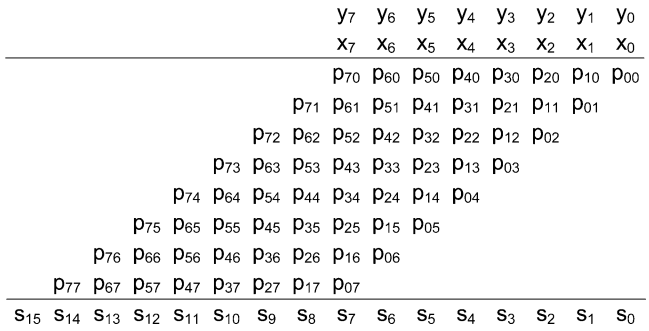

Fig. 1. Illustration of an unsigned 8-bit multiplication.

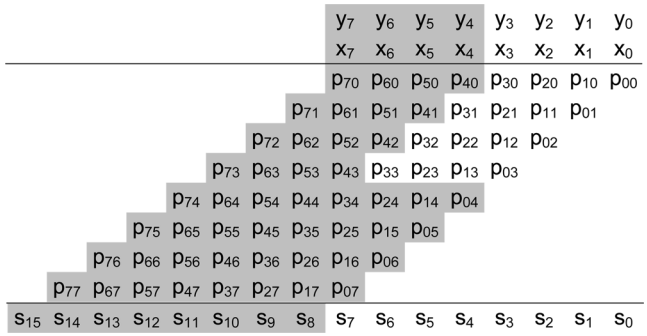

Fig. 2. Illustration of an unsigned 8-bit multiplication, where the precision of the operands is smaller than the precision of the multiplication. Unused bits of operands and product, as well as unused partial products, are shown in gray.

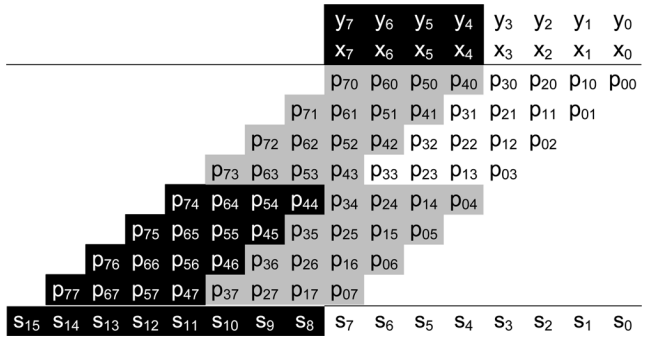

Fig. 3. Illustration of an unsigned 8-bit multiplication, where a 4-bit multiplication, shown in white, is computed in parallel with a second 4-bit multiplication, shown in black.

bility of performing double-throughput multiplications. The twin-precision technique is an efficient way of achieving double throughput in a multiplier with low area overhead and with only a small delay penalty. We show how to apply the twin-precision technique on signed multipliers based on the regular High Performance Multiplier (HPM) [2] reduction tree. The two algorithms for signed multiplications that have been used are Baugh–Wooley and the popular modified-Booth algorithm.

To exemplify the practical value of the twin-precision technique, we have added a light-weight SIMD extension that utilizes a twin-precision multiplier to a simple five-stage processor (similar to a MIPS R2000 [8]). In comparison to the baseline processor, the modified processor has negligible delay and power overhead, but can still achieve significant application speedups and energy savings by utilizing its intrinsic SIMD capabilities.

This paper is organized as follows. First, we introduce the twin-precision technique in Section II. This is followed by a description of how the twin-precision technique can be applied to a Baugh–Wooley, Section III, and a modified-Booth multiplier, Section IV. The evaluation setup and netlist generation is reviewed in Section V, whereas Section VI and Section VII present the evaluation results for a 130-nm and a 65-nm process, respectively. Section VIII presents the proportion of narrow-width (16-bit) multiplications in some typical embedded applications, while Section IX presents a case study. Finally, the paper is concluded in Section X.

II. Twin-Precision Fundamentals

Initially we present the twin-precision technique using an illustration of unsigned binary multiplication. In an unsigned binary multiplication each bit of one of the operands, called the multiplier, is multiplied with the second operand, called multiplicand

$$p_{ij} = y_i x_j. \tag{1}$$

That way one row of partial products is generated. Each row of partial products is shifted according to the position of the bit of the multiplier, forming what is commonly called the partial-product array. Finally, partial products that are in the same column are summed together, forming the final result. An illustration of an 8-bit multiplication is shown in Fig. 1. Let us look at what happens when the precision of the operands is smaller than the multiplier we intend to use. In this case, the most significant bits of the operands will only contain zeros, thus large parts of the partial-product array will consist of zeros. Further, the summation of the most significant part of the partial-product array and the most significant bits of the final result will only consist of zeros. An illustration of an 8-bit multiplication, where the precision of the operands is four bits, is shown in Fig. 2.

Fig. 2 shows that large parts of the partial-product array only consist of zeros and are, thus, not contributing any useful information to the final result. What if these partial products could be utilized for a second, concurrent multiplication?

Since partial products of the same column are summed together, it would not be wise to use any of the partial products that are in the same column as the multiplication that is already computed. Looking closer at the 4-bit multiplication marked in white in Fig. 2, one can also observe that the column at position S_7 should not be used either. This is because that column might have a carry from the active part of the partial-product array that will constitute the final S_7. Altogether this makes only the partial products in the most significant part of the partial-product array available for a second multiplication.

In order to be able to use the partial products in the most significant part, there has to be a way of setting their values. For this we can use the most significant bits of the operands, since these are not carrying any useful information. If we are only looking at the upper half of the operands, the partial products generated from these bits are the ones shown in black in Fig. 3. By setting the other partial products to zero, it is possible to perform two multiplications within the same partial-product array, without changing the way the summation of the partial-product array is

done. How the partial products, shown in gray, can be set to zero will be presented in the implementation section later on.

Assume, for now, that there is a way of setting unwanted partial products to zero: Now it suddenly becomes possible to partition the multiplier into two smaller multipliers that can compute multiplications in parallel. In the above illustrations the two smaller multiplications have been chosen such that they are of equal size. This is not necessary for the technique to work. Any size of the two smaller multiplications can be chosen, as long as the precision of the two smaller multiplications together are equal or smaller than the full precision (N) of the multiplication

$$N \geq N_{\mathrm{LSP}} + N_{\mathrm{MSP}}. \tag{2}$$

To be able to distinguish between the two smaller multiplications, they are referred to as the multiplication in the Least Significant Part (LSP) of the partial-product array with size N_{LSP}, shown in white, and the multiplication in the Most Significant Part (MSP) with size N_{MSP}, shown in black. It is functionally possible to partition the multiplier into even more multiplications. For example, it would be possible to partition a 64-bit multiplier into four 16-bit multiplications. Given a number K of low-precision multiplications, their total size needs to be smaller or equal to the full-precision multiplication

$$N \geq \sum_{i=1}^{K} N_i. \tag{3}$$

In the remainder of this paper, the precision of the two smaller multiplications will be equal and half the precision $(N/2)$ of the full precision (N) of the multiplier.

A. A First Implementation

The basic operation of generating a partial product is that of a 1-bit multiplication using a two-input AND gate, where one of the input signals is one bit of the multiplier and the second input signal is one bit of the multiplicand. The summation of the partial products can be done in many different ways, but for this investigation we are only interested in parallel multipliers that are based on 3:2 full adders.[2] For this first implementation an array of adders will be used because of its close resemblance to the previously used illustration of a multiplication; see Fig. 4.

In the previous section we assumed that there is a way of setting unwanted partial products to zero. This is easily accomplished by changing the two-input AND gate to a three-input AND gate, where the extra input can be used for a control signal. Of course, only the AND gates of the partial products that have to be set to zero need to be changed to a three-input version. During normal operation when a full-precision multiplication is executed the control signal is set to high, thus all partial products are generated as normal and the array of adders will sum them together and create the final result. When the control signal is set to low the unwanted partial products will become zero. Since the summation of the partial products is not overlapping, there is no need to modify the array of adders. The array of adders will produce the result of the two multiplications in the upper and lower part of the final output. The block diagram of an 8-bit twin-precision array multiplier capable of computing two 4-bit multiplications is shown in Fig. 5. The two multiplications have been colored in white and black to visualize what part of the adder array is used for what multiplication.

[2]Higher-radix compression *is* compatible with our strategy.

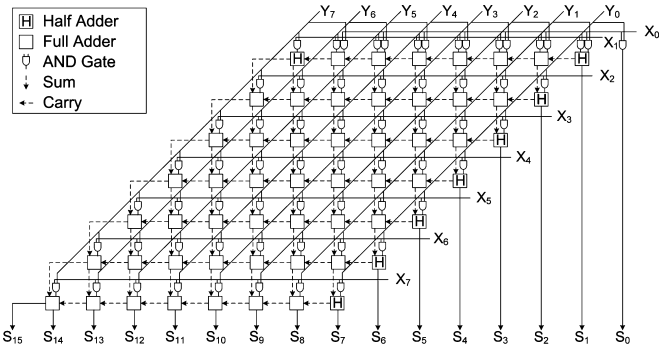

Fig. 4. Block diagram of an unsigned 8-bit array multiplier.

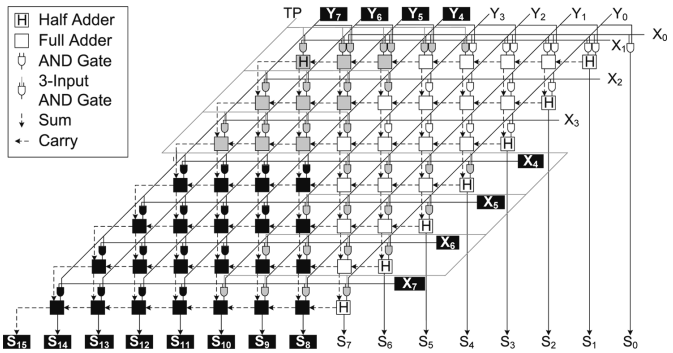

Fig. 5. Block diagram of an unsigned 8-bit twin-precision array multiplier. The TP signal is used for controlling if one full-precision multiplication should be computed $(\mathrm{TP} = \mathrm{high})$ or two 4-bit multiplications should be computed in parallel $(\mathrm{TP} = \mathrm{low})$.

More flexibility might be desired, like the possibility to compute a single narrow-width multiplication or two parallel narrow-width multiplications, within the same multiplier. This can be done by changing the two-input AND gates for the partial-product generation of the narrow-width multiplications as well. In the array multiplier in Fig. 5, the AND gates for the 4-bit MSP multiplication, shown in black, can be changed to three-input AND gates to which a second control signal can be added. Assuming the multiplier is divided into two equal parts, this modification makes it possible to either compute an N-bit, a single $N/2$-bit or two concurrent $N/2$-bit multiplications.

B. An HPM Implementation

The array multiplier in the previous section was only used to show the principle of the twin-precision technique. For high-speed and/or low-power implementations, a reduction tree with logarithmic logic depth, such as TDM [9], Dadda [10], Wallace [11] or HPM [12] is preferred for summation of the partial products. Such a log-depth reduction tree has the benefit of shorter logic depth. Further, a log-depth tree suffers from fewer glitches making it less power dissipating. A twin-precision implementation based on the regular HPM reduction tree is shown in Fig. 6.

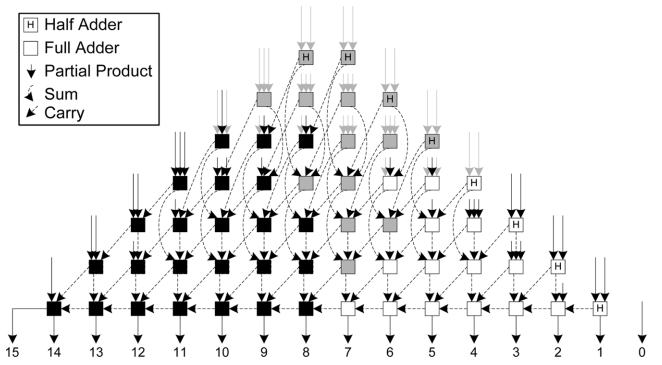

Fig. 6. Block diagram of an unsigned 8-bit twin-precision multiplier that is based on the regular HPM reduction tree. A 4-bit multiplication, shown in white, can be computed in parallel with a second 4-bit multiplication, shown in black. For simplicity of the figure the AND gates for partial-product generation is not shown and a ripple carry is used as final adder.

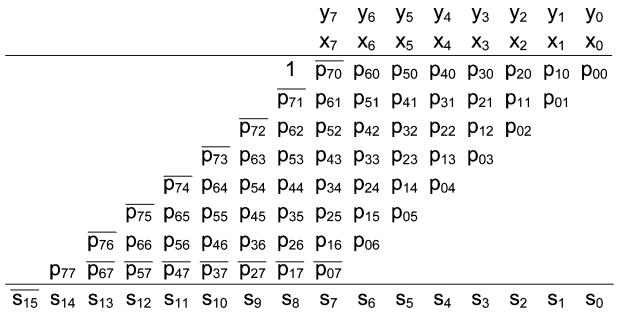

Fig. 7. Illustration of a signed 8-bit multiplication, using the Baugh–Wooley algorithm.

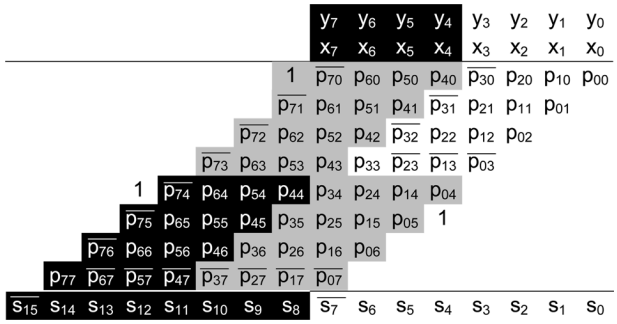

Fig. 8. Illustration of a signed 8-bit multiplication, using the Baugh–Wooley algorithm, where one 4-bit multiplication, shown in white, is computed in parallel with a second 4-bit multiplication, shown in black.

C. The Final Adder

The ripple-carry adder has been used as a final adder to simplify the figures and the description of the twin-precision technique. However, the ripple-carry adder can be exchanged for any other more suitable adder, without modifying the adder itself. The reason for this is that when operating on narrow-width operations, there will be no carry passing between the two multiplications and hence, no modification of the adder is needed.

III. A Baugh–Wooley Implementation

In the previous section, the concept of twin-precision was introduced by way of an unsigned multiplication example. However, for many applications signed multiplications are needed and consequently an unsigned multiplier is of limited use. In this section a twin-precision multiplier based on the Baugh–Wooley (BW) algorithm will be presented.

A. Algorithms for Baugh–Wooley

The BW algorithm [13] is a relative straightforward way of performing signed multiplications. Fig. 7 illustrates the algorithm for an 8-bit case, where the partial-product array has been reorganized according to the scheme of Hatamian [14]. The creation of the reorganized partial-product array comprises three steps: i) the most significant partial product of the first $N-1$ rows and the last row of partial products except the most significant have to be negated, ii) a constant one is added to the Nth column, iii) the most significant bit (MSB) of the final result is negated.

B. Twin-Precision Using the Baugh–Wooley Algorithm

It is not as easy to deploy the twin-precision technique onto a BW multiplication as it is for the unsigned multiplication, where only parts of the partial products need to be set to zero. To be able to compute two signed $N/2$ multiplications, it is necessary to make a more sophisticated modification of the partial-product array. Fig. 8 illustrates an 8-bit BW multiplication, in which two 4-bit multiplications have been depicted in white and black.

When comparing the illustration of Fig. 7 with that of Fig. 8 one can see that the only modification needed to compute the 4-bit multiplication in the MSP of the array is an extra sign bit '1' in column S_{12}. For the 4-bit multiplication in the LSP of the array, there is a need for some more modifications. In the active partial-product array of the 4-bit LSP multiplication (shown in white), the most significant partial product of all rows, except the last, needs to be negated. For the last row it is the opposite, here all partial products, except the most significant, are negated. Also for this multiplication a sign bit '1' is needed, but this time in column S_4. Finally the MSB of the results needs to be negated to get the correct result of the two 4-bit multiplications.

To allow for the full-precision multiplication of size N to coexist with two multiplications of size $N/2$ in the same multiplier, it is necessary to modify the partial-product generation and the reduction tree. For the $N/2$-bit multiplication in the MSP of the array all that is needed is to add a control signal that can be set to high, when the $N/2$-bit multiplication is to be computed and to low, when the full precision N multiplication is to be computed. To compute the $N/2$-bit multiplication in the LSP of the array, certain partial products need to be negated. This can easily be accomplished by changing the two-input AND gate that generates the partial product to a two-input NAND gate followed by an XOR gate. The second input of the XOR gate can then be used to invert the output of the NAND gate. When computing the $N/2$-bit LSP multiplication, the control input to the XOR gate is set to low making it work as a buffer. When computing a full-precision N multiplication the same signal is set to high making the XOR work as an inverter. Finally the MSB of the result needs to be negated and this can again be achieved by using an XOR gate together with an inverted version of the

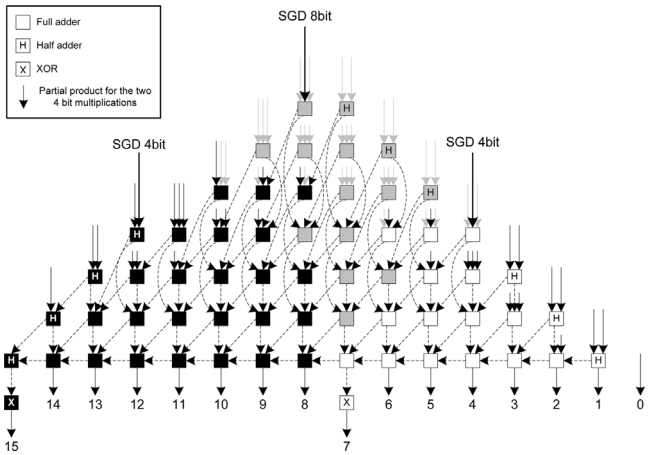

Fig. 9. Block diagram of a signed 8-bit multiplication, using the Baugh–Wooley algorithm, where one 4-bit multiplication, shown in white, is computed in parallel with a second 4-bit multiplication, shown in black.

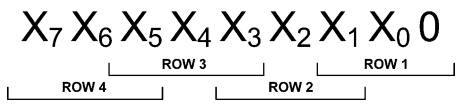

Fig. 10. 8-bit modified-Booth encoding.

								y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
								x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
						$\overline{p_{80}}$	p_{70}	p_{60}	p_{50}	p_{40}	p_{30}	p_{20}	p_{10}	p_{LSB0}	
					$\overline{p_{81}}$	p_{71}	p_{61}	p_{51}	p_{41}	p_{31}	p_{21}	p_{11}	p_{LSB1}		
				$\overline{p_{82}}$	p_{72}	p_{62}	p_{52}	p_{42}	p_{32}	p_{22}	p_{12}	p_{LSB2}			
		$\overline{p_{83}}$	p_{73}	p_{63}	p_{53}	p_{43}	p_{33}	p_{23}	p_{13}	p_{LSB3}					
1	0	1	0	1	0	1	1	a_3		a_2		a_1		a_0	
s_{15}	s_{14}	s_{13}	s_{12}	s_{11}	s_{10}	s_9	s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0

Fig. 11. Illustration of a signed 8-bit multiplication using the modified-Booth algorithm.

control signal for the XOR gates used in the partial-product generation. Setting unwanted partial products to zero can be done by three-input AND gates as for the unsigned case.

Fig. 9 shows an implementation of a twin-precision 8-bit BW multiplier. The modifications of the reduction tree compared to the unsigned 8-bit multiplier in Fig. 6 consist of three things; i) the half adders in column 4 and 8 have been changed to full adders in order to fit the extra sign bits that are needed, ii) for the sign bit of the 4-bit MSP multiplication there is no half adder that can be changed in column 12, so here an extra half adder has been added, which makes it necessary to also add half adders for the following columns of higher precision, and iii) finally XOR gates have been added at the output of column 7 and 15 so that they can be inverted.

The simplicity of the twin-precision BW implementation makes it easy to also compute unsigned multiplications. All that is needed is to set the control signals accordingly, such that none of the partial products are negated, the XOR gates are set to not negate the final result and all the sign bits are set to zero.

IV. A Modified-Booth Implementation

Modified Booth (MB) is a popular and common algorithm for implementation of signed multipliers. MB is a more complicated algorithm than Baugh–Wooley (BW), but it has the advantage of only producing half the number of partial products. In this section a twin-precision multiplier based on the MB algorithm will be presented.

A. Algorithms for Modified Booth

The original Booth algorithm [15] is a way of coding the partial products generated during a $s = x \times y$ multiplication. This is done by considering two bits at a time of the multiplier, x, and coding them into $\{-2, -1, 0, 1, 2\}$. The encoded number is then multiplied with the multiplicand, y, into a row of recoded partial products. The number of recoded partial products is fewer than for a scheme with unrecoded partial products, which during implementation may translate into higher performance.

The drawback of the original Booth algorithm is that the number of generated partial products depends on the x-multiplier, which makes the Booth algorithm unsuitable for implementation in hardware. The modified-Booth (MB) algorithm [16] by MacSorley remedies this by looking at three bits at a time of the multiplier. Then we are guaranteed that only half the number of partial products will be generated, compared to a conventional partial product generation using two-input AND gates. With a fixed number of partial products the MB algorithm is suitable for hardware implementation. Fig. 10 shows which parts of the multiplier that are encoded and used to recode the multiplicand into a row of partial products.

An MB multiplier works internally with two's complement representation of the partial products, in order to be able to multiply the encoded $\{-2,-1\}$ with the multiplicand. To avoid having to sign extend the rows of recoded partial products, the sign-extension prevention scheme presented by Fadavi–Ardekani [17] has been used in our twin-precision implementation. In two's complement representation, a change of sign includes the insertion of a '1' at the Least Significant Bit (LSB) position. To avoid getting an irregular partial-product array we draw on the idea of Yeh *et al.* [18], called modified partial-product array. The idea is to pre-compute the impact on the two least significant positions of a row of recoded partial products by the insertion of a '1' during sign change. The pre-computation calculates the addition of the LSB with the potential '1', from which the sum is used as the new LSB for the row of recoded partial products. A potential carry from the pre-computation is inserted at the second least significant position. The pre-computation of the new LSB can be done according to (4). The pre-computation of a potential carry is as given by (5). Here (5) is different from that presented by Yeh *et al.* [18].

$$p_{\mathrm{LSBi}} = y_0(x_{2i-1} \oplus x_{2i}) \tag{4}$$

$$a_i = x_{2i+1}(\overline{x_{2i-1} + x_{2i}} + \overline{y_{\mathrm{LSB}} + x_{2i}} + \overline{y_{\mathrm{LSB}} + x_{2i-1}}) \tag{5}$$

Fig. 11 shows an illustration of an 8-bit MB multiplication with the sign-extension prevention and modified partial-product array scheme.

B. Twin-Precision Using the Modified-Booth Algorithm

Implementing the twin-precision scheme on the MB algorithm is not as straightforward as for the BW case (Section III).

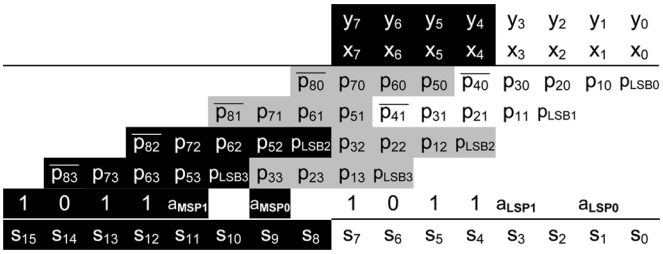

Fig. 12. Illustration of a signed 8-bit multiplication, using the modified-Booth algorithm, where one 4-bit multiplication, shown in white, is computed in parallel with a second 4-bit multiplication, shown in black.

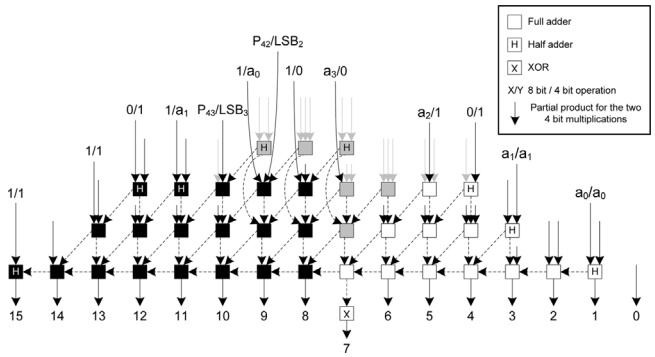

Fig. 13. Block diagram of a signed 8-bit multiplication using the modified-Booth algorithm, where a 4-bit multiplication, shown in white, is computed in parallel with a second 4-bit multiplication, shown in black.

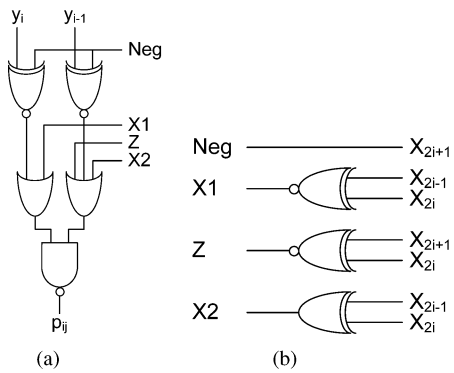

Fig. 14. Encode and decode circuit for modified Booth.

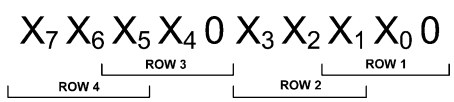

Fig. 15. Encoding scheme for two 4-bit multiplications.

It is not possible to take the partial products from the full-precision MB multiplication and use only the partial products that are of interest for the narrow-width MB multiplications. The reason for this is that all partial products are not computed the same way and there exist several special cases that need to be handled.

By comparing the two multiplications in Figs. 11 and 12 (for which we used $N = 8$), we can see what needs to be handled in order to switch between the different modes of operation:

- The partial products that are denoted p_{40} and p_{41} during normal 8-bit multiplication (Fig. 11) need to define partial products that are used to prevent sign extension in the narrow-width 4-bit multiplication in the LSP (Fig. 12).
- The partial-products that are denoted p_{42} and p_{43} during normal 8-bit multiplication need to define p_{LSB2} and p_{LSB3} for the narrow-width 4-bit multiplication in the MSP.
- The a_{MSP0} and a_{MSP1} that are needed for the multiplication in the MSP have to be added.
- The pattern of 1's and 0's for the normal 8-bit multiplication cannot be used in narrow-width mode. For the two 4-bit multiplications, we need two shorter patterns of 1's and 0's.

The implementation of the MB twin-precision multiplication does not call for any significant changes to the reduction tree of a conventional MB multiplier. When comparing the multiplications in Figs. 11 and 12, we can see that the position of the signals in the lowest row is the only difference that has an impact on the reduction tree. This means that there is a need for an extra input in two of the columns ($N/2$ and $3N/2$) compared to the conventional MB multiplier; this requires two extra half adders in the reduction tree.

The biggest difference between a conventional MB multiplier and a twin-precision MB multiplier is the generation of inputs to the reduction tree. To switch between modes of operation, logic is added to the recoder to allow for generation of the partial products needed, for sign-extension prevention as well as $p_{\mathrm{LSB}i}$, which are needed for $N/2$-bit multiplications in the LSP and the MSP, respectively. There is also a need for multiplexers that, depending on the mode of operation, select the appropriate signal as input to the reduction tree. Further, partial products that are not being used during the computation of $N/2$-bit multiplications have to be set to zero in order to not corrupt the computation. An example of an 8-bit MB twin-precision multiplier is shown in Fig. 13.

The recoding logic can be implemented in many different ways. For this implementation we have chosen the recoding scheme presented by Yeh *et al.* [18]. The circuits for encoding and decoding are shown in Fig. 14.

For the partial products that need to be set to zero, the output of the decode circuit can be connected to the input of a two-input AND gate. The second input of the AND gate can then be used as a control signal, as in the case of the unsigned and BW implementations. This is a straightforward method, and it is possible to construct more efficient solutions for setting the partial product to zero, if the option of custom designed cells is available. The latter is not necessary for correct function, but it could increase the speed of the decode circuit.

For correct operation the input to the encoder for the first row in the $N/2$-bit MSP multiplication has to be set to zero, instead of using $x_{N/2-1}$ as its input. An example of the encoding scheme for two 4-bit multiplications is shown in Fig. 15.

In order to separate the two different $N/2$-bit multiplications, such that the multiplication in the LSP does not interfere with the multiplication in the MSP, we need to consider another issue. By looking at the pattern of 1's and 0's that is used for sign-extension prevention, we see that the most significant '1' is only used to invert the final s_{2N-1} bit. However, the carry that potentially could be generated by this extra '1' does not affect for the final result. If the most significant '1' for the multiplication in

the LSP would be inserted into the reduction tree it would mean that a carry could be generated. This potential carry would propagate into the multiplication in the MSP and corrupt the result. To avoid inserting the most significant '1', an XOR gate is added after the final adder allowing the MSB of the $N/2$-bit LSP multiplication to be negated, which is the sole purpose of the most significant '1'.

V. Netlist Generation and Evaluation Setup

To evaluate the efficiency of the twin-precision technique a multiplier generator was written [19]. The generator is capable of generating VHDL descriptions of conventional Baugh–Wooley (BW) and modified-Booth (MB) multipliers as well as twin-precision versions of both of these, according to the schemes presented in Sections III and IV.[3] The VHDL generator was verified to generate correct multipliers of sizes up to 16 bits by simulating all possible input patterns and verifying the result using Cadence NC-VHDL [21]. For multipliers larger than 16 bits, the functionality was verified by feeding the multipliers with one million random input vectors and verifying the result.

The VHDL descriptions were synthesized using Synopsys Design Compiler [22] together with a commercially available 1.2-V 130-nm process. The synthesized netlist was taken through place-and-route using Cadence Encounter [23]. To create a common interface to the multipliers and ease the usage of the EDA tools that do not interact efficiently with purely combinatorial circuits, registers were inserted at the primary multiplier inputs and outputs. Delay estimations are based on RC extracted data from the placed-and-routed netlists, while power estimates are obtained after applying value change dump (VCD) statistics of simulations of 10 000 random input vectors for each of the possible modes of operation. Delay, power, and area figures include the input and output registers on the multipliers and are given for the worst-case corner at 1.08 V.

The timing constraints for place-and-route were set systematically through extensive use of scripting; the goal was to push the limits of the timing for each generated VHDL description.[4] This strategy indeed achieves a fast implementation, however, the power dissipation becomes high, due to excessive buffering and resizing of gates. Therefore, the timing was relaxed with 100, 300, and 600 ps, relative to the fastest timing obtained for each implementation. Power estimates were subsequently obtained for each timing constraint.

A. Synthesis and Layout of Baugh–Wooley Netlists

Our timing analysis showed that the simplicity of the BW implementation comes with an added benefit in that it does not create high fanout signals. The signals with highest fanout in the BW case are the input signals, which are connected to the input of N two-input AND gates for a multiplier of size N. This creates a reasonable fanout of the input signals for multipliers up to at least 64 bits, without degrading the performance significantly. The mapping of the BW multiplier code to gate-level netlist during synthesis was only constrained in the way that half and full adders of the reduction tree were mapped to their respective minimum-size cells and the map effort in Design Compiler was set to high.

[3]A Kogge–Stone [20] adder was chosen as final adder for all types of multipliers.

[4]For the twin-precision implementations, the control signals for switching mode between N, $1 \times N/2$, and $2 \times N/2$ were not included when estimating timing. However, these control signals do not have any significant effect on the timing in a practical case, as will be shown in Section IX.

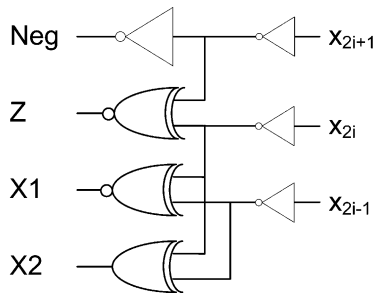

Fig. 16. Buffered encode circuit for fanout reduction.

B. Synthesis and Layout of Modified-Booth Netlists

The first attempt at creating layouts for the VHDL generated MB multipliers exposed a big problem with high fanout signals. This can easily be realized by investigating the encode and decode circuits from Fig. 14. As can be seen, the x_{2i+1} input signal goes straight through the encoder as the NEG signal and drives two XNOR gates for each partial product of a single row. This means that the fanout of half of the primary x-inputs is at least $2N$ XNOR gates for a multiplier of size N. Further, the encoder outputs X1, Z, and X2 drive N decoders creating a need for large XOR and XNOR gates in the encode stage, which increases the fanout of the x-inputs even more.

To deal with the fanout problem, the fanout of X1, Z and X2 was reduced by instantiating multiple encoders for each row of partial products. To limit the fanout of the x_{2i+1} signal, an inverter buffer was inserted to drive the NEG signal. Finally, minimum-size inverters were added to x_{2i-1}, x_{2i}, and x_{2i+1}, inputs of the encoder, thus, the fanout of primary x-inputs is reduced. The new encode circuit is shown in Fig. 16.

The mapping of the MB multiplier code to a gate-level netlist was constrained in such way that Design Compiler could not remove the minimum-size inverters of the new encoder, Fig. 16. In all other respects the synthesized netlist of the MB multiplier was constrained in the same way as the BW netlist. In other words, half and full adders of the reduction tree were mapped to their respective cells of minimum size from the cell library and the map effort was set to high.

VI. Results and Discussion

One of the goals of the twin-precision technique is to keep the performance degradation of the multiplier's full-precision operation at a minimum. To compare twin-precision implementations against conventional Baugh–Wooley (BW) and modified-Booth multipliers (MB), each multiplier was taken through the EDA flow outlined previously.

Fig. 17 shows the delay and power dissipation for conventional and twin-precision BW and MB multipliers of size 16, 32, and 48 bits. The figure shows how the power changes as the timing requirements are relaxed with 100, 300, and 600 ps.

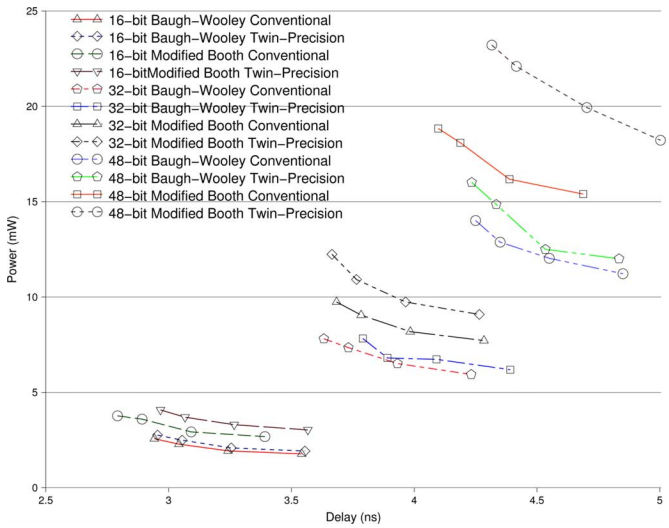

Fig. 17. The power dissipation for different timing constraints applied to conventional and twin-precision Baugh–Wooley and modified-Booth multipliers of size 16, 32, and 48 bits.

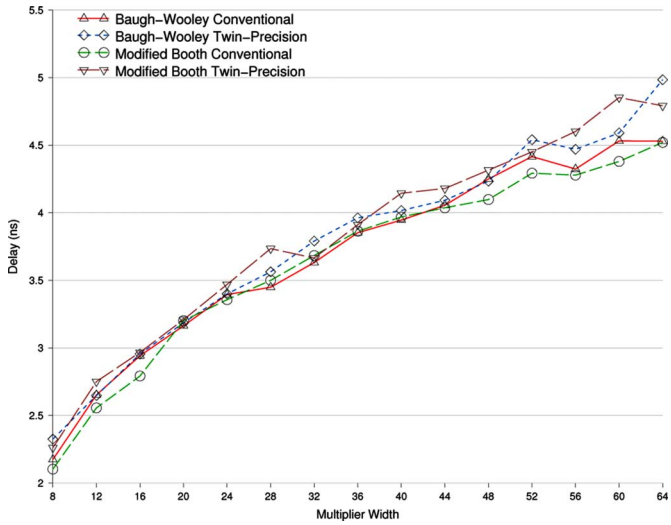

Fig. 18. The delay for conventional and twin-precision Baugh–Wooley and modified-Booth multipliers of sizes from 8 to 64 bits.

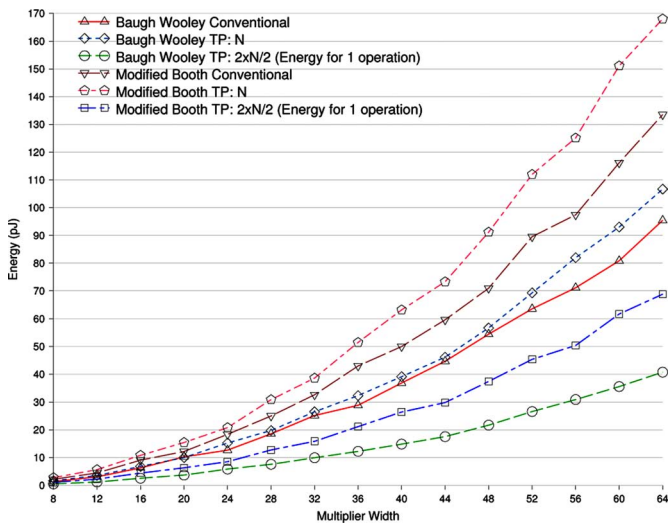

Fig. 19. The energy-per-operation for conventional and twin-precision Baugh–Wooley and modified-Booth multipliers of sizes from 8 to 64 bits.

A clear trend is that a BW implementation is more power efficient than a MB implementation. However, a MB implementation can, in some cases, exhibit higher maximum speed.

The result of the comparison of the twin-precision implementations with their conventional counterparts is that a twin-precision implementation of Baugh–Wooley performs equal in terms of delay for the 16- and 48-bit case and is only 160 ps slower for the 32-bit case. When we consider power, the twin-precision implementation dissipates 8%, 5%, and 6% more power than a conventional 16-, 32-, and 48-bit BW implementation.

From our results it is clear that the complexity of the MB recoding circuit makes it difficult to efficiently update an MB implementation into a twin-precision multiplier. The MB twin-precision implementation has the poorest performance, both in terms of delay and power, of the four compared implementation choices.

A. Delay

It is clear that the delay is not greatly degraded by the introduction of the twin-precision technique. Fig. 18 shows the minimum delay for conventional and twin-precision BW and MB multipliers. As can be seen the difference in timing is not large, ranging from 0 to 150 ps in difference when comparing the two BW multipliers.[5] The figure shows that a twin-precision BW implementation is the better choice compared to a twin-precision MB implementation.

B. Energy per Operation

One of the reasons to choose a twin-precision implementation instead of a conventional multiplier implementation is that energy can be saved[6] by reducing the precision of the multiplier, when operating on narrow-width operands. To compute the energy-per-operation, we extracted delay and power values (Fig. 17) for various sizes of each multiplier design. We then computed the energy, as $\text{energy} = \text{delay} \cdot \text{power}$, for each delay and power pair and from these we chose the smallest energy for each multiplier design and size. The result from this investigation is shown in Fig. 19.

The twin-precision multiplier on average has 10% higher energy-per-operation, when operating on full-precision N-bit data, than a conventional BW implementation. However, for half of the implementations the energy is on average only 5% higher. The main benefit, in terms of energy, of the twin-precision technique is when operating the multiplier in narrow-width mode. Fig. 19 also shows the energy-per-operation for a single multiplication of size $N/2$-bit when computing two $N/2$ multiplications concurrently. When working with narrow-width data the twin-precision multiplier has on average 59% lower energy dissipation than the conventional BW multiplier.

From Fig. 19 we can establish that a MB implementation is not energy efficient, whether it is a conventional or twin-precision implementation.

[5]Not counting the 64-bit implementation.

[6]The other reason is to improve computational throughput.

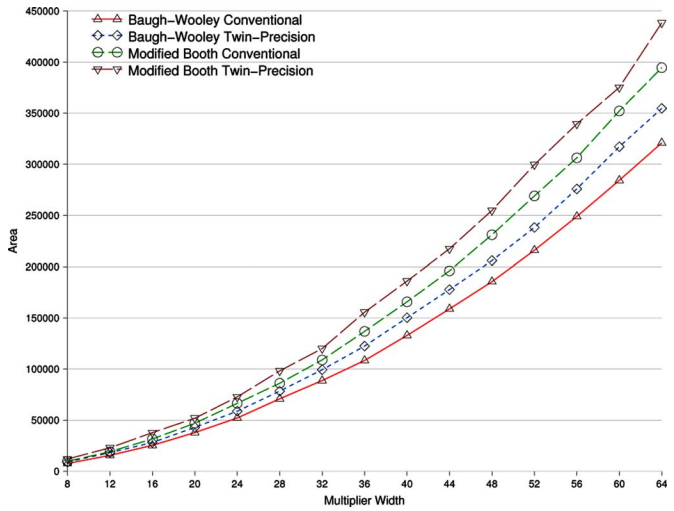

Fig. 20. The area for conventional and twin-precision Baugh–Wooley and modified-Booth multipliers of sizes from 8 to 64 bits.

C. Area

Fig. 20 shows that a twin-precision multiplier requires slightly more area than its conventional counterpart. However, the twin-precision implementation based on the BW algorithm is smaller than the commonly used conventional MB implementation.

D. Power Reduction by the Use of Power Gating

Even though the reduction in power by the use of the twin-precision technique is substantial, the power overhead of an $N/2$-bit multiplication compared to a conventional multiplier of size $N/2$ is high. For the 32-bit twin-precision BW implementation operating on a single 16-bit data, the overhead is 55% in our 130-nm technology evaluation. This overhead is reduced to 15% per operation if the twin-precision multiplier is operating on two 16-bit data concurrently. The reduction in power overhead is due to less logic of the multiplier being idle and leaking, and that the static power dissipation of the idle logic is amortized over two 16-bit operations.

To reduce the power dissipation overhead of the narrow-width mode it is possible to apply power gating, instead of only setting the partial products to zero by the use of three-input AND gates. Power gating involves adding power switches to the logic cells that should be gated off when idle, effectively isolating the cell from the power supply when the cell is idle. The output of the power-gated cells will, thus, be undefined and to avoid these signals to interfere with the computation in the active part of the multiplier, they need to be forced to zero. This can be done by adding a transistor, on the output of the cells, that is connected to ground. Thus, when the power supply is gated off, the transistor is activated making the output go low. This calls for a custom layout though, since such cells, where the output is forced to zero when power gated, can be hard to find in standard-cell libraries.

To be able to apply power gating to different areas of the partial-product generation, the reduction tree, and the final adder, a suitable power grid is needed. Fig. 21 shows an example of a power grid for an 8-bit twin-precision BW multiplier.

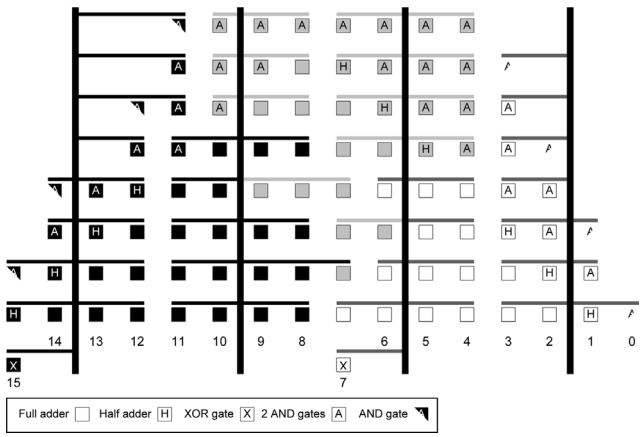

Fig. 21. Power grid for an 8-bit twin-precision Baugh–Wooley implementation.

Previous work of ours [24] showed that by using precision-dependent power gating based on the SCCMOS technique, the power overhead for a 16-bit BW multiplier can be reduced by 53% at a delay penalty of less than 3%. The results from that work cannot be directly applied to these investigations, because both process technology and implementation abstraction level are different.[7] However, that work indicates the potential in using precision-dependent power-gating for reducing the power dissipation even further for twin-precision multipliers.

VII. Implementation in a 65-nm Process

We also used a commercial 65-nm process (low power cell library with standard V_T) to extend our evaluation. We implemented four different 32-bit multipliers; a conventional Baugh–Wooley (BW), a conventional modified-Booth (MB), a twin-precision BW and a twin-precision MB multiplier.

For this 65-nm EDA flow, Cadence First Encounter [25] was used for synthesis, placement, and routing. For the 65-nm process, we did not do the same fanout investigations and buffer insertions for the MB multipliers, as described for the 130-nm process. All VHDL descriptions were taken through the EDA flow using the same script and setup. The synthesis was restricted to use only full and half adder cells. However, the technology mapping algorithm is able to improve the designated gate structure, and thus the adder functionality in some cases is implemented as a set of elementary standard cells. Timing and power estimates are obtained for the worst case 125 °C corner at 1.1 V. The power dissipation was estimated using value change dump (VCD) data from simulations with 10 000 random input vectors; just as in the case of the 130-nm evaluation. The timing we present includes switching of control signals for changing operation mode of the twin-precision multipliers.

The results from the placed-and-routed 65-nm 32-bit conventional and twin-precision BW and MB multipliers are shown in Table I. The table shows the minimum delay and area for the four different multiplier implementations, as well as the power dissipation for the conventional multipliers and for the three different operation modes of the twin-precision multipliers.

[7]The process technologies are both 130 nm, however the vendors are different.

TABLE I
DELAY, POWER, AND AREA ESTIMATES FOR 32-BIT CONVENTIONAL AND TWIN-PRECISION BAUGH–WOOLEY AND MODIFIED-BOOTH MULTIPLIERS IN A 65-NM PROCESS

	Delay (ns)	Power (mW)	Area ($k\mu m^2$)
Baugh Conv.	2.59 (100%)	23.4 (100%)	48 (100%)
Baugh 1x32-bit	2.80 (108%)	24.2 (103%)	53 (111%)
Baugh 1x16-bit	-	8.8 (38%)	-
Baugh 2x16-bit	-	14.5 (62%)	-
Booth Conv.	2.50 (97%)	37.5 (160%)	52 (108%)
Booth 1x32-bit	2.68 (103%)	38.0 (162%)	53 (111%)
Booth 1x16-bit	-	20.2 (86%)	-
Booth 2x16-bit	-	27.0 (115%)	-

TABLE II
DELAY, POWER, AND AREA ESTIMATES FOR 32-BIT CONVENTIONAL AND TWIN-PRECISION BAUGH–WOOLEY AND MODIFIED-BOOTH MULTIPLIERS IN A 130-NM PROCESS

	Delay (ns)	Power (mW)	Area ($k\mu m^2$)
Baugh Conv.	3.63 (100%)	7.8 (100%)	89 (100%)
Baugh 1x32-bit	3.99 (110%)	7.3 (94%)	99 (111%)
Baugh 1x16-bit	-	4.0 (51%)	-
Baugh 2x16-bit	-	5.5 (71%)	-
Booth Conv.	3.68 (101%)	9.7 (124%)	109 (122%)
Booth 1x32-bit	3.75 (103%)	11.1 (142%)	120 (135%)
Booth 1x16-bit	-	7.2 (92%)	-
Booth 2x16-bit	-	9.3 (119%)	-

The twin-precision implementation is about 200 ps slower than its conventional counterpart. Further, the MB implementation is slightly faster than the corresponding BW implementation. The twin-precision BW implementation occupies 11% more area than a conventional BW implementation, while for the twin-precision MB implementation the increase in area is only 3%, compared to the conventional MB.

More interestingly, there is a large power reduction when a 32-bit twin-precision multiplier operates on 16-bit data. The power dissipation is reduced by more than 60%, when computing one 16-bit multiplication with the twin-precision BW multiplier, compared to a conventional BW multiplier. If an application is adopted to compute two 16-bit multiplications in parallel, the reduction in power dissipation per operation is improved further and would be close to 70%.

We can observe similar improvements for the twin-precision MB multiplier, but the generally high power dissipation makes MB a poor choice for low-power implementations.

For reference, Table II shows the same data for the 130-nm process technology as Table I shows for the 65-nm process technology. We notice that the timing and area for the 32-bit conventional and twin-precision BW implementation show similar results for the two different technologies. Considering power we can see that the overhead for the twin-precision multiplier, when operating on 32-bit data, is higher for the 65-nm technology. However, the reduction in power when operating on 16-bit data is significantly larger for the 65-nm process than for the older 130-nm process.

When comparing the MB implementations for the two different process technologies, the 65-nm process offers large improvements for the timing and area values. The BW implementations do not gain as much from scaling. However, the power dissipation of the MB implementations drastically increases relative a BW implementation when moving to the newer process.

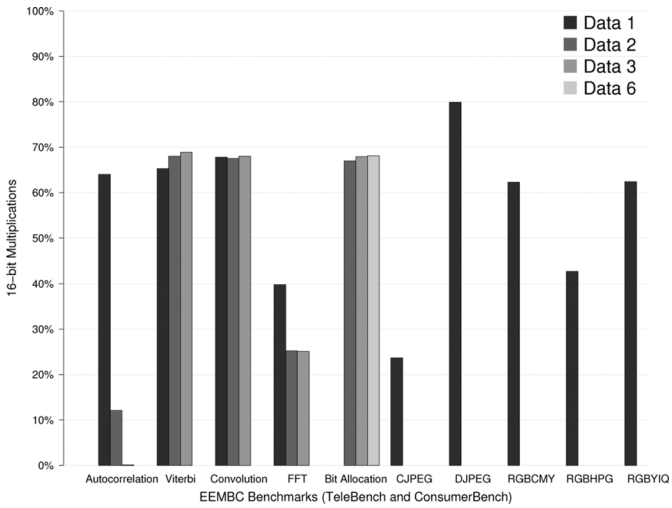

Fig. 22. Fraction of narrow-width (16-bit) multiplications in the EEMBC TeleBench and ConsumerBench suites.

VIII. WORKLOAD CHARACTERIZATION

The SimpleScalar [26] (Version 3.0d) instruction-set simulator has been used to simulate ten different benchmarks from the EEMBC [27] TeleBench and ConsumerBench benchmark suites. This was done in order to get realistic data of the amount of narrow-width (16-bit) multiplications in typical embedded workloads. The sim-fast version of the SimpleScalar simulator was modified such that the result for each executed multiplication is monitored: If a result is small enough to be represented using only 16 bits, a counter variable is incremented. Fig. 22 shows the result from the simulations, with the number of narrow-width multiplications shown as the percentage of the total number of executed multiplications. The figure clearly shows that many of the multiplications being performed could be represented using only 16 bits. On average, for all the benchmarks and their different data sets, the fraction of narrow-width operations is 52% of all executed multiplications.

The way the multiplications of the benchmarks were monitored assures that the result from the multiplier can indeed be fed, without any truncation, directly to another unit operating on narrow-width operands, since the result is only 16-bit wide. However, this gives a pessimistic value of the number of narrow-width multiplications that really exists in the benchmarks, since only the input operands need to be explicitly expressed using 16 bits. Take for example the fast Fourier transform (FFT) benchmark, which works with 16-bit data: In the FFT benchmark simulation 25–40% of the multiplications were found to be of narrow-width type, depending on the input data. However, the FFT benchmark works only with 16-bit data, which has the consequence that the result of the multiplication is right-shifted 15 bits. This inherent truncation makes it possible to compute all multiplications in narrow-width mode, as will be shown in Section IX.

From the data given in Tables I and II, it is possible to calculate the energy dissipation when performing multiplications

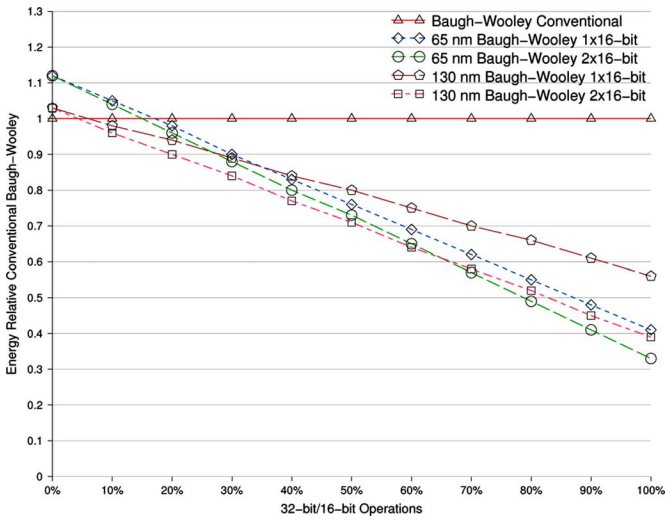

Fig. 23. Energy dissipation when varying the relation between 32-bit and 16-bit operands for twin-precision Baugh–Wooley multipliers in the 65-nm and 130-nm processes, relative their respective conventional multiplier.

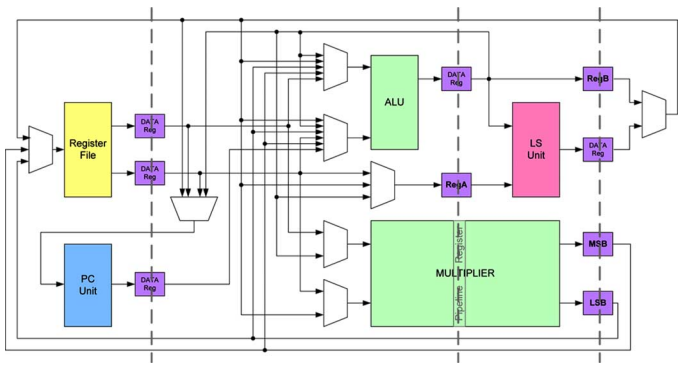

Fig. 24. Schematic view of the implemented processor datapath.

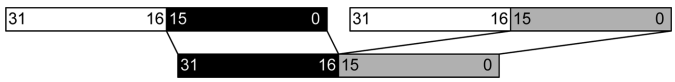

Fig. 25. Two 32-bit data items are converted to one packed 16-bit item.

with a certain fraction of 16-bit operands. Fig. 23 shows the energy dissipation for twin-precision Baugh–Wooley (BW) multipliers, as the fraction of signed 16-bit operands is varied. The energy is shown relative that of a conventional (signed) BW multiplier that dissipates roughly the same energy regardless of what input operand bitwidth is used.

In the 65-nm process, for a twin-precision BW multiplier to have equal to or less energy dissipation than a conventional BW multiplier, at least 15%–18% of the operations have to be executed in 16-bit mode. For the 130-nm process, the same figure is only 5%–7%. However, as the fraction of 16-bit operations increases, further to the right in the figure, the benefit of utilizing a twin-precision multiplier increases more rapidly for the 65-nm process.

The energy dissipation when performing the multiplications for the ten chosen benchmarks can on average be reduced by 25%–28% for the 65-nm implementation and 21%–30% for the 130-nm implementation, if the 1×16-bit and 2×16-bit modes would be utilized by the benchmarks. The energy reduction from using the twin-precision technique is, thus, similar across the entire set of workloads. Specifically an energy reduction of about 25% can be expected for the multiplier unit.

IX. SIMD Multiplier Extension, A Case Study

Single Instruction Multiple Data (SIMD) is a way of exploiting data parallelism in an application and is commonly used in high-end and embedded processors. SIMD instructions are commonly supported by a specialized datapath unit or coprocessor [28]–[30] that requires additional hardware units or as an intrinsic part of the standard datapath [31]–[34] that commonly lack support for SIMD integer multiplications.

The twin-precision technique makes it possible to support SIMD multiplications without the need to add another costly multiplier unit. A designer can apply the twin-precision technique, at a very small cost, on the conventional multiplier of a processor. This allows for SIMD multiplier functionality in processors, where it is normally not considered.

To evaluate this possibility, we have chosen to modify a processor that is inspired by the MIPS R2000 [8]. The Arithmetic Logic Unit (ALU) and the multiplier are modified such that the processor not only can perform 32-bit operations, but such that it also has support for performing operations on packed data. Here, a packed data item represents the concatenation of two 16-bit data items; we henceforth refer to this as *a packed 16-bit data*. A schematic of the implemented datapath is shown in Fig. 24.

A. Instruction Set Extension

New instructions need to be augmented to the basic instruction set, in order to allow the processor to support 16-bit SIMD operations. But to reduce the impact on the processor cycle time, we add as few instructions as possible. Logical operations are performed bit-wise, thus, their functionality is the same regardless if the data is 32-bit wide or if it is packed 16-bit data. Therefore, there is no need to modify or add any instructions for logical operations. To support arithmetic operations on packed 16-bit data, three new instructions are added: Addition (PADD), subtraction (PSUB), and multiplication (PMULT). In order to limit the instruction growth's impact on performance, specialized instructions for shift, compare and branch are not added.

To ease conversion from 32-bit to packed 16-bit data, an instruction (PACK) is added: As illustrated in Fig. 25, PACK takes two 32-bit data items, and generates a packed 16-bit data item out of each respective lower 16 bits.

The conversion of a packed 16-bit data item into two 32-bit data items is partly supported by existing instructions: i) The 16 higher bits of the packed 16-bit data item are extracted through a shift of 16 positions to the right. This can either be done with the conventional Shift Right Logic (SRL) or Shift Right Arithmetic (SRA) instruction to get an unsigned or signed 32-bit data item. ii) For the 16 lower bits, the upper half of the packed 16-bit data is truncated (for unsigned representation) or it needs to represent a sign extension (for signed representation). A truncation can be done by a normal AND instruction, and a bit mask with '0' for the 16 higher bits and '1' for the 16 lower bits. However, the sign extension of the lower 16 bits can not be performed with

TABLE III
ADDED INSTRUCTIONS TO THE PROCESSOR INSTRUCTION SET

Instruction	Functional Unit
PADD	ALU
PSUB	ALU
PACK	ALU
SIGN16	ALU
PMULT	Multiplier

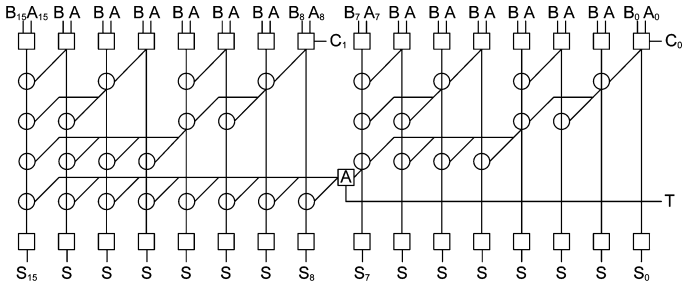

Fig. 26. A 16-bit Sklansky adder can be modified to optionally support also two 8-bit additions.

a single instruction from the conventional instruction set of the processor. An instruction (SIGN16) that sign extends the lower 16 bits is therefore included.

Table III lists the five instructions that we add to the basic instruction set and the functional units that subsequently need to be modified, to provide the hardware support.

B. Arithmetic Logic Unit

The adder in the Arithmetic Logic Unit (ALU) needs to be explicitly designed to support the PADD and PSUB instructions. Our implementation is based on a Sklansky prefix adder [35], an adder type which is known to represent a good trade-off between performance and area. More importantly, for this work, it has a SIMD-friendly structure that inherently offers the twin-precision capability for addition: By only cutting the carry chain between the lower and the higher 16 bits of the adder, a 32-bit Sklansky adder can also support the computation of two 16-bit additions in parallel. The ALU part of our light-weight SIMD extension relies on the insertion of a single AND-gate (A) in the carry-propagation path and the connection of one of its inputs to a control signal (T), see Fig. 26. If control signal T is high, the adder will operate as a conventional 32-bit adder. On the other hand, if T is low, the carry propagation path is broken, and the adder will compute two 16-bit additions instead.

To support the PSUB instruction it is necessary to insert a carry at position 16 of the adder: For this purpose we use, at position 16, a circuit that generates the propagate and generate signals from bits of the A and B operands and an input carry signal (C).

The ALU is also extended with the PACK and SIGN16 instructions. These are low-complexity instructions, which only cause input signals to be rerouted.

C. Evaluation and Results

The processor using the light-weight SIMD extension was evaluated using the compiler, simulation and implementation framework of the FlexSoC project [36], [37]. The experimental framework consists of a compiler that reads MIPS assembly, and generates data traces used both for cycle-accurate architectural simulations and for logic simulation of our VHDL implementations.

TABLE IV
DELAY, POWER, AND AREA ESTIMATES

	Delay (ns)	Power (mW)	Area (kμm^2)
Conv.	2.29	6.41	267
SIMD	2.35	6.50	283

The experimental framework was updated so that the compiler supports the new PMULT, PADD, PSUB, PACK and SIGN16 instructions. The VHDL implementation of the processor datapath was updated with the modified ALU and a twin-precision Baugh–Wooley multiplier, as described in the previous sections. In this implementation, in order to reduce the critical path of the processor, the multiplier is pipelined into two stages. Both the reference and the twin-precision enabled processor used multipliers that had one level of registers situated between reduction tree and final adder.[8] The VHDL implementation does not include the logic for the control circuitry, but timing and power estimates do include buffers for the datapath's control signals. The addition of only five instructions is expected to have a negligible impact on processor performance, in terms of control circuitry.

To evaluate the added SIMD capability, we modified the Fast Fourier Transform (FFT) application from the EEMBC [27] TeleBench suite. The application implements a decimation-in-time 256-point 16-bit FFT. The application source code was compiled to assembly using GCC mips-cross-compiler with EEMBC's default optimization flags set. The computational kernel was identified and MULT, ADD, and SUB instructions were manually changed to PMULT, PADD, and PSUB instructions when applicable. The PACK instruction was used for packing the two 32-bit results from a PMULT instruction into one packed 16-bit data item. The SIGN16 instruction was never used by the FFT application, but it is still included in the implementation as a support for other applications.

VHDL implementations of the processor with and without the light-weight SIMD extension were taken through a synthesis [22] and place-and-route [23] flow for the 130-nm process used in previous sections. Delay and power estimations are based on RC extracted data from the placed-and-routed netlists. For power estimation, value change dump (VCD) data from logic simulations of the FFT applications was used to capture accurate switching activities. The presented power estimates are defined at the maximum clock frequency for each of the implementations.

Table IV shows the results for the two processor implementations. The delay and power values are almost identical; a 60-ps and a 0.09-mW increase can be observed for the processor implementation with SIMD support. The processor with SIMD extension has a limited area penalty of 6%.

The number of executed cycles for the FFT application is reduced by 18%, when comparing the result from the original FFT application executed on the conventional processor with that of the execution of the SIMD-enabled FFT application executed on

[8]Pipelining a twin-precision multiplier is as straightforward as for conventional multipliers.

TABLE V
CYCLE COUNT, EXECUTION TIME, AND ENERGY DISSIPATION

	Cycle Count	Time (ms)	Energy (μJ)
Conv.	57368 (100%)	131 (100%)	842 (100%)
SIMD	47292 (82%)	111 (85%)	722 (86%)

TABLE VI
MEMORY READ AND WRITE ACCESS

	Read	Write
Conv.	9013 (100%)	5568 (100%)
SIMD	5170 (57%)	3371 (61%)

the processor with SIMD extension; see Table V. This translates into a 15% speedup of total execution time and a 14% energy reduction for the SIMD-enabled FFT application.

The main memory was not modeled here and, thus, power dissipation for memory accesses is not included. However, since retrieving and storing the data of the FFT computation is done on packed 16-bit data, where two 16-bit data items are read in parallel, the SIMD implementation has 39–43% less read and write accesses to main memory; see Table VI. This is close to the optimal reduction of memory accesses that can be achieved by utilizing SIMD execution. If all performed accesses are for retrieving and storing data of the FFT computation, the number of memory access could ideally be reduced by 50%, since two data items are read in parallel. The reason that the reduction of memory accesses deviates from the ideal is due to storing and retrieving variables, such as address pointers, on the stack located in main memory.

In embedded multimedia systems, access to main memory and traffic on shared buses are two of the main concerns during system design. With a non-ideal memory subsystem, the access to memory could take longer time than the single clock cycle we have assumed. This could have a larger negative impact on the execution time for the conventional implementation. The fewer number of accesses should also reduce the energy dissipation for the SIMD implementation, even though more accesses are 32-bit wide instead of 16-bit as for the conventional implementation.

X. CONCLUSION

The presented twin-precision technique allows for flexible architectural solutions, where the variation in operand bitwidth that is common in most applications can be harnessed to decrease power dissipation and to increase throughput of multiplications.

It turns out that the Baugh–Wooley algorithm implemented on a HPM reduction tree is particularly suitable for a twin-precision implementation. Due to the simplicity of the implementation, only minor modifications are needed to comply with the twin-precision technique. This makes for an efficient twin-precision implementation, capable of both signed and unsigned multiplications.

Currently a lot of research is done on reconfigurable architectures, where the architecture can be adapted to the applications that are being executed. Some of these proposed architectures can adapt their arithmetic logic units to operate on different bitwidths, depending on the application. One such reconfigurable architecture is that of the FlexSoC project [38]. In these types of architectures it is necessary to have a multiplier that can efficiently operate over a wide range of bitwidths. The twin-precision technique, which offers flexibility at a low implementation overhead, makes it possible to efficiently deploy these flexible architectures.

ACKNOWLEDGMENT

The authors wish to thank their coworkers and students who have contributed to this project: Dr. H. Eriksson, Dr. M. Draždžiulis, Lic Eng. M. Islam, M. Brinck, K. Eklund, C. Berglund, L. Johansson, M. Samuelsson, and J. Moe.

REFERENCES

[1] D. Brooks and M. Martonosi, "Dynamically exploiting narrow width operands to improve processor power and performance," in *Proc. 5th Int. Symp. High Perform. Comput. Arch.*, Jan. 1999, pp. 13–22.

[2] Z. Huang and M. D. Ercegovac, "Two-dimensional signal gating for low-power array multiplier design," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2002, pp. 489–492.

[3] K. Han, B. L. Evans, and E. E. Swartzlander, "Data wordlength reduction for low-power signal processing software," in *Proc. IEEE Workshop Signal Process. Syst.*, 2004, pp. 343–348.

[4] G. H. Loh, "Exploiting data-width locality to increase superscalar execution bandwidth," in *Proc. 35th Int. Symp. Microarchitecture*, 2002, pp. 395–405.

[5] A. Danysh and D. Tan, "Architecture and implementation of a vector/SIMD multiply-accumulate unit," *IEEE Trans. Comput.*, vol. 5, no. 4, pp. 284–293, May 2005.

[6] P. Mokrian, M. Ahmadi, G. Jullien, and W. Miller, "A reconfigurable digital multiplier architecture," in *Proc. IEEE Canadian Conf. Electr. Comput. Eng.*, 2003, pp. 125–128.

[7] M. Själander, H. Eriksson, and P. Larsson-Edefors, "An efficient twin-precision multiplier," in *Proc. 22nd IEEE Int. Conf. Comput. Des.*, Oct. 2004, pp. 30–33.

[8] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design, The Hardware/Software Interface*, 2nd ed. New York: Morgan Kaufman, 1998.

[9] 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.

[10] L. Dadda, "Some schemes for parallel adders," *Alta Frequenza*, vol. 34, no. 5, pp. 349–356, May 1965.

[11] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. 13, pp. 14–17, Feb. 1964.

[12] 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.

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

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

[15] A. D. Booth, "A signed binary multiplication technique," *Quarterly J. Mechan. Appl. Math.*, vol. 4, no. 2, pp. 236–240, 1951.

[16] O. L. MacSorley, "High speed arithmetic in binary computers," *Proc. Inst. Radio Eng.*, vol. 49, no. 1, pp. 67–97, Jan. 1961.

[17] J. Fadavi-Ardekani, "M $\times$ N Booth encoded multiplier generator using optimized Wallace trees," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 1, no. 2, pp. 120–125, 1993.

[18] 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.

[19] M. Själander, HMS Multiplier Generator. Feb. 2008 [Online]. Available: http://www.sjalander.com/research/multiplier

[20] 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. 22, no. 8, pp. 786–793, Aug. 1973.

[21] NC-VHDL Simulator Help Version 5.1. Cadence.
[22] Design Compiler User Guide Version W-2004.12. Synopsys.
[23] Encounter User Guide Version 4.1. Cadence.
[24] M. Själander, M. Draždžiulis, P. Larsson-Edefors, and H. Eriksson, "A low-leakage twin-precision multiplier using reconfigurable power gating," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2005, pp. 1654–1657.
[25] Encounter User Guide Version 6.2. Cadence.
[26] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, Feb. 2002.
[27] Embedded Microprocessor Benchmark Consortium. [Online]. Available: http://www.eembc.org
[28] S. K. Raman, V. Pentkovski, and J. Keshava, "Implementing streaming SIMD extensions on the Pentium III processor," *IEEE Micro*, vol. 20, no. 4, pp. 47–57, Jul./Aug. 2000.
[29] K. Diefendorff, P. K. Dubey, R. Hochsprung, and H. Scale, "AltiVec extension to PowerPC accelerates media processing," *IEEE Micro*, vol. 20, no. 2, pp. 85–95, Mar./Apr. 2000.
[30] M. Tremblay, M. O'Connor, V. Narayanan, and L. He, "VIS speeds new media processing," *IEEE Micro*, vol. 16, no. 4, pp. 10–20, Aug. 1996.
[31] R. B. Lee, "Accelerating multimedia with enhanced microprocessors," *IEEE Micro*, vol. 15, no. 2, pp. 22–32, Apr. 1995.
[32] R. B. Lee, "Subword parallelism with MAX-2," *IEEE Micro*, vol. 16, no. 4, pp. 51–59, Aug. 1996.
[33] J. Goodacre and A. N. Sloss, "Parallelism and the ARM instruction set architecture," *IEEE Comput.*, vol. 38, no. 7, pp. 42–50, Jul. 2005.
[34] *"ARM Architecture Reference Manual,"* 1st ed. ARM Limited, Cambridge, U.K., Jul. 2005.
[35] J. Sklansky, "Conditional-sum addition logic," *IRE Trans. Electron. Comput.*, pp. 226–231, Jun. 1960.
[36] M. Själander, P. Larsson-Edefors, and M. Björk, "A flexible datapath interconnect for embedded applications," in *IEEE Comput. Soc. Annu. Symp. VLSI*, May 2007, pp. 15–20.
[37] M. Thuresson, M. Själander, M. Björk, L. Svensson, P. Larsson-Edefors, and P. Stenström, "FlexCore: Utilizing exposed datapath control for efficient computing," in *IEEE Int. Conf. Embedded Computer Systems: Architectures, Modeling and Simulation*, Jul. 2007, pp. 18–25.
[38] J. Hughes, K. Jeppson, P. Larsson-Edefors, M. Sheeran, P. Stenström, and L. J. Svensson, "FlexSoC: Combining flexibility and efficiency in SoC designs," in *Proc. IEEE NorChip Conf.*, 2003, pp. 52–55.

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 is a Digital Hardware Designer with Aeroflex Gaisler, Göteborg. His research interests include multicore and reconfigurable architectures, arithmetic circuits, and embedded systems.

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 holds the Chair of Computer Engineering with Chalmers University of Technology, Göteborg, Sweden. He was a Visiting Scientist with National Microelectronics Research Center, Ireland, in 1996/1997 and a Visiting Professor with Intel Corporation's Circuit Research Laboratory in 2000. His research interests include low-power high-performance digital circuits, with emphasis on design methodology, low-power embedded architectures, and macromodeling for efficient performance and power estimation.