A Power-Efficient and Versatile Modified-Booth Multiplier

Magnus Själander and Per Larsson-Edefors
VLSI Research Group, Department of Computer Science and Engineering
Chalmers University of Technology, SE-412 96 Göteborg, Sweden

Abstract— **A power-efficient twin-precision modified-Booth multiplier is presented. For full-precision operation (16-bit inputs), the twin-precision modified-Booth multiplier shows an insignificant increase of critical-path delay (0.4% or 32 ps) compared to a conventional multiplier. To cancel the power overhead of extra logic and to achieve overall power reduction, the implemented 16-bit twin-precision multiplier needs to run in dual 8-bit mode for more than 4.4% of its computations.**

I. Introduction

Recent development of embedded systems indicates an increased interest in reconfigurable functional units that dynamically can make the datapath adapt to varying computational needs. A system may need to switch between, for example, one application for speech encoding that requires functional units operating at 8-bit precision and another application that is based on 16-bit functional units to perform audio decoding.

The twin-precision multiplier [1] can switch between *N*-bit and *N*/2-bit precision multiplications without significant performance or area overhead. Previous work [1] introduced a twin-precision technique for radix-2 tree multipliers, but when higher performance is needed, multipliers with higher radix than two may be an option. In this paper we therefore explore the possibility of implementing the twin-precision concept on modified-Booth multipliers.

Section II reviews the twin-precision concept as well as the modified-Booth algorithm. Section III describes the use of the twin-precision concept for modified-Booth multipliers and Section IV details the corresponding implementation. Results for the modified-Booth twin-precision multiplier are given in Section V. Finally, Section VI concludes the paper.

II. Preliminaries

A. Basic Unsigned Twin-Precision Multiplication

The chief idea behind twin-precision multiplication is to have the ability to, apart from the default full-precision multiplication, also allow for either one or two low-precision multiplications. By looking at the pen-and-paper multiplication illustration in Fig. 1, where different regions have been grouped together, it is clear that it is possible to make two *N*/2-bit multiplications within an *N*-bit multiplication. By calculating the partial products shown in the gray region, it is possible to compute one *N*/2-bit multiplication. If the partial products for both the gray and black regions are calculated, two *N*/2-bit multiplications can be performed in parallel. When performing an *N*-bit multiplication, all partial products of course need to be computed. To be able to distinguish between the two different ”smaller” multiplications the gray region is said to be in the Least Significant Part (LSP) and the black region in the Most Significant Part (MSP) of the multiplier. The low-precision multiplications are not constrained to *N*/2-bit precision; this precision was merely used here for illustrating the principle. The restriction on the two ”smaller” multiplications is that their precisions combined are less or equal to the full-precision multiplication (Eq. 1).

$$N_{Full} \geq N_{Small1} + N_{Small2} \tag{1}$$

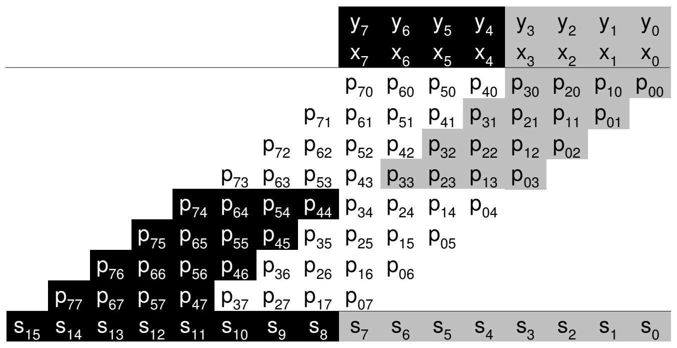
y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0
x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
p_{70} p_{60} p_{50} p_{40} p_{30} p_{20} p_{10} p_{00}
p_{71} p_{61} p_{51} p_{41} p_{31} p_{21} p_{11} p_{01}
p_{72} p_{62} p_{52} p_{42} p_{32} p_{22} p_{12} p_{02}
p_{73} p_{63} p_{53} p_{43} p_{33} p_{23} p_{13} p_{03}
p_{74} p_{64} p_{54} p_{44} p_{34} p_{24} p_{14} p_{04}
p_{75} p_{65} p_{55} p_{45} p_{35} p_{25} p_{15} p_{05}
p_{76} p_{66} p_{56} p_{46} p_{36} p_{26} p_{16} p_{06}
p_{77} p_{67} p_{57} p_{47} p_{37} p_{27} p_{17} p_{07}
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. 1. Partial-product representation of an 8-bit multiplication with two 4-bit multiplications.

$$p_{ij} = y_i x_j \tag{2}$$

The twin-precision concept has been shown [1] to translate into an efficient use of resources, when used for multipliers such as the radix-2 tree multiplier. It has been shown that by optionally setting different regions of partial products to zero, it is possible to compute one *N*, one *N*/2, or two concurrent *N*/2-bit multiplications using an *N*-bit radix-2 tree multiplier. Only with a minor increase of logic, this enables the computation of *N*/2-bit multiplications at reduced power, with shorter delay, and with a possibility for higher throughput, as compared to an *N*-bit multiplier without the twin-precision feature.

B. The Modified-Booth Algorithm

The modified-Booth algorithm has the advantage that it reduces the number of rows of partial products by using clever encoding and decoding of the operands [2]. In this paper we use the modified-Booth encoding scheme that was proposed by Wen-Chang *et al.* [3]. The recoding is done by encoding three bits of one of the operands (Eq. 3-5), which in turn are used to decode the partial products for a single row using the second operand (Eq. 6).

$$X1 = \overline{x_{2j-1} \oplus x_{2j}} \quad (3)$$
$$Z = \overline{x_{2j+1} \oplus x_{2j}} \quad (4)$$
$$X2 = x_{2j-1} \oplus x_{2j} \quad (5)$$

$$p_{ij} = \overline{(\overline{y_i \oplus x_{2j+1}} + X1)(\overline{y_{i-1} \oplus x_{2j+1}} + Z + X2)} \quad (6)$$

Fig. 2 shows which parts of the operand that are encoded and used to decode a specific row of partial products.

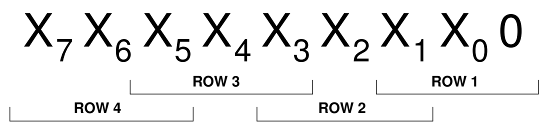

Fig. 2. 8-bit encoding.

To avoid sign extending each row of partial products, the scheme presented by Fadavi-Ardekani [4] has been used. Instead of sign extension, an extra partial product for each row and a special pattern of 1's and 0's have been added. To further simplify implementation, the partial product array is modified according to the scheme used by Wen-Chang *et al.* [3] where *i)* the Least Significant Bit (LSB) (Eq. 7) is treated in a special way, and *ii)* an a_i (Eq. 8) is added for each row of partial products.

$$p_{LSBi} = y_0(x_{2i-1} \oplus x_{2i}) \quad (7)$$
$$a_i = x_{2i+1}(\overline{x_{2i-1} + x_{2i}} + \overline{y_0 + x_{2i}} + \overline{y_0 + x_{2i-1}}) \quad (8)$$

The final result of the modified-Booth algorithm is shown in Fig. 3.

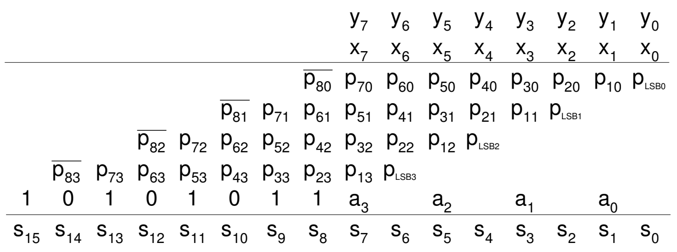
y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0
x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
$\overline{p_{80}}$ p_{70} p_{60} p_{50} p_{40} p_{30} p_{20} p_{10} p_{LSB0}
$\overline{p_{81}}$ p_{71} p_{61} p_{51} p_{41} p_{31} p_{21} p_{11} p_{LSB1}
$\overline{p_{82}}$ p_{72} p_{62} p_{52} p_{42} p_{32} p_{22} p_{12} p_{LSB2}
$\overline{p_{83}}$ 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. 3. 8-bit modified-Booth multiplication.

III. Supporting Twin Precision in Modified Booth

Applying the twin-precision concept on modified Booth is not as straightforward as it was for the unsigned multiplication (Section II-A). It is not possible to take the partial products from the full-precision modified-Booth multiplication and use only the partial product bits that are of interest for the low-precision modified-Booth multiplications. The reason for this is that all partial product bits are not computed the same way, but there exist several special cases that need to be handled.

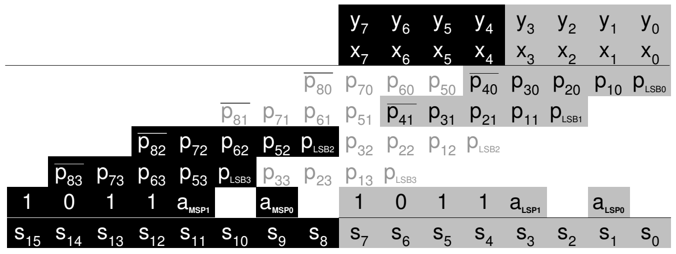
y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0
x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
$\overline{p_{80}}$ p_{70} p_{60} p_{50} $\overline{p_{40}}$ p_{30} p_{20} p_{10} p_{LSB0}
$\overline{p_{81}}$ p_{71} p_{61} p_{51} $\overline{p_{41}}$ p_{31} p_{21} p_{11} p_{LSB1}
$\overline{p_{82}}$ p_{72} p_{62} p_{52} p_{LSB2} p_{32} p_{22} p_{12} p_{LSB2}
$\overline{p_{83}}$ p_{73} p_{63} p_{53} p_{LSB3} p_{33} p_{23} p_{13} p_{LSB3}
1 0 1 1 a_{MSP1} a_{MSP0} 1 0 1 1 a_{LSP1} a_{LSP0}
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. 4. Two 4-bit modified-Booth multiplications.

By comparing the two multiplications in Fig. 3 and Fig. 4 (for which we used $N = 8$), we can see what needs to be handled in order to switch between the different modes of computation:

- The partial-product bits that are denoted p_{40} and p_{41} during normal 8-bit multiplication (Fig. 3) need to define partial products that are used to prevent sign extension in the low-precision 4-bit multiplication in the LSP (Fig. 4).
- The partial-products bits that are denoted p_{42} and p_{43} during normal 8-bit multiplication need to define p_{LSB2} and p_{LSB3} for the low-precision 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 low-precision mode. For the two 4-bit multiplications, we need two shorter patterns of 1's and 0's.

IV. A Modified-Booth Twin-Precision Multiplier

The implementation of the modified-Booth twin-precision multiplication, which was described in the previous section, does not call for any significant changes to the reduction tree of a conventional modified-Booth multiplier. When comparing the multiplications in Fig. 3 and Fig. 4, 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 3*N*/2) compared to the conventional modified-Booth multiplier; this requires two extra half adders in the reduction tree.

The biggest difference between a conventional modified-Booth multiplier and a twin-precision modified-Booth 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_{LSBi}, 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.

Partial products that are not being used during the computation of either one *N*/2-bit or two concurrent *N*/2-bit multiplications have to be set to zero in order to not corrupt the computation. Actually, cancelling partial product bits has the added benefit that it also reduces dynamic power. An example of an 8-bit modified-Booth twin-precision multiplier is shown in Fig. 5.

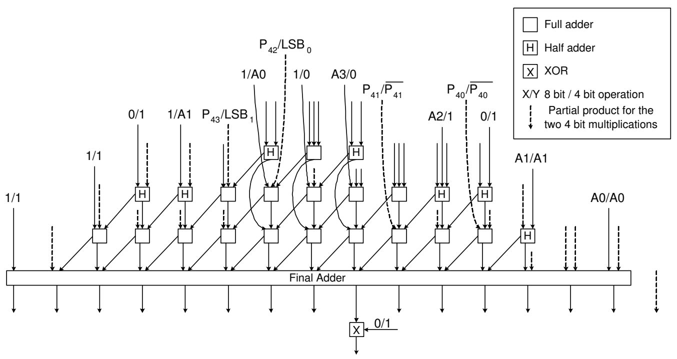
Full adder
H Half adder
X XOR
X/Y 8 bit / 4 bit operation
Partial product for the two 4 bit multiplications
P_{42}/LSB_0
1/A0
1/0
A3/0
$P_{41}/\overline{P_{41}}$
$P_{40}/\overline{P_{40}}$
0/1
1/A1
P_{43}/LSB_1
A2/1
0/1
1/1
A1/A1
1/1
A0/A0
Final Adder
0/1

Fig. 5. 8-bit twin-precision modified-Booth multiplier.

With four extra transistors the decoder can set a partial product to zero (the extra transistors are encircled in Fig. 6); depending on the mode of operation, the *Zero* input controls whether a partial product is set to zero.

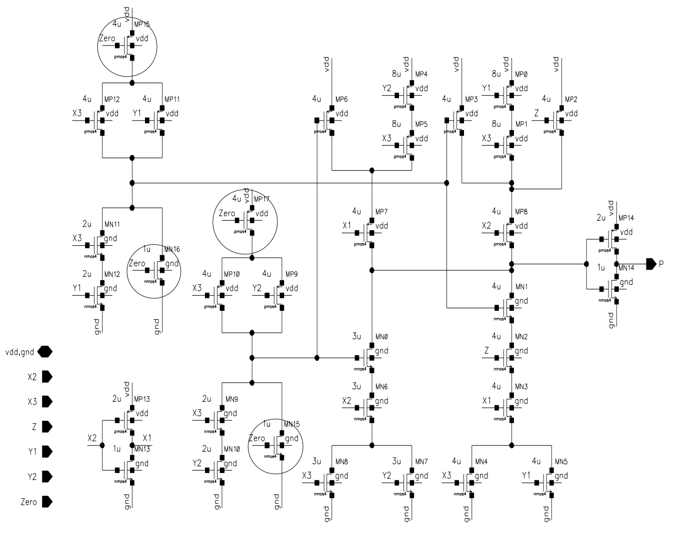

Fig. 6. Decoder circuit that optionally sets partial products to zero.

For correct operation the input to the encoder for the first row in the *N*/2-bit multiplication in the MSP 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 can be seen in Fig. 7, which can be compared to the encoding scheme for the 8-bit multiplication in Fig. 2.

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 some other issues. By looking at the pattern of 1's and 0's that is used for sign-extension prevention, we see that the

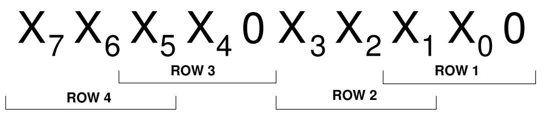

Fig. 7. Encoding scheme for two 4-bit multiplications.

most significant 1 is only used to invert the final s_{2N-1}-bit. However, the carry that this extra 1 potentially could generate is not of interest 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, instead the MSB of the result for the multiplication in the LSP is negated, since this is the single function of the final 1. This is the reason an XOR gate is added after the final adder in order to be able to invert the MSB of *N*/2-bit multiplications in the LSP and still get correct result for *N*-bit multiplications.

V. Simulation and Results

A 16-bit modified-Booth twin-precision multiplier with a Kogge-Stone [5] structure as final adder has been simulated. The multiplier has been partitioned so that it is also capable of performing one 8-bit or two concurrent 8-bit multiplications. As reference a conventional 16-bit modified-Booth multiplier with a Kogge-Stone final adder is used.

Both multipliers have been implemented as circuit netlists in a commercially available 0.35-μm CMOS technology using static logic. The simulations were run at a supply voltage of 3.3 V. Power figures were obtained by Spectre [6] simulations using 64 input vectors running at 100 MHz with an operating temperature of 27°C. Power dissipation from control signal transitions has not been included, since the multiplier is expected to operate in the same mode for longer durations [7]. To obtain the critical-path delay, PathMill [8] has been used (assuming an operating temperature of 90°C).

TABLE I
Total power dissipation [mW]

Mult.	Power [mW, %]	
16-bit	24.8	100%

(a) Conventional

Mult.	Mode	Power [mW, %]	
16-bit	16-bit	25.6	103.2%
16-bit	One 8-bit	10.4	41.9%
16-bit	Two 8-bit	15.2	61.3%

(b) Twin-precision

Table I shows the total power dissipation for the three possible modes of the twin-precision multiplier as well as the power dissipation of the reference multiplier. From the table it is clear that a significant reduction in power dissipation can be achieved when operating the twin-precision multiplier in 8-bit mode. However, there is a small power overhead of 3% for the 16-bit mode.

To achieve a reduction in total power dissipation, the twin-precision multiplier has to operate in single 8-bit mode for more than 10% of the time. If we compute two concurrent 8-bit multiplications only 4.4% (Eq. 9-16) of the multiplications has to be run with lower precision to achieve total power reduction.

$$\begin{aligned}
x &= \mathrm{Number\ of\ 8-bit\ multiplications} && (9)\\
1 &= \mathrm{Total\ number\ of\ multiplications} && (10)\\
P_{conv.} &= 1 * 24.8\,\mathrm{mW} && (11)\\
P_{twin} &= (1-x) * 25.6 + x/2 * 61.3\,\mathrm{mW} && (12)\\
P_{conv.} &= P_{twin} && (13)\\
1 * 24.8 &= (1-x) * 25.6 + x/2 * 61.3 && (14)\\
18x &= 0.8 && (15)\\
x &\approx 4.4\% && (16)
\end{aligned}$$

TABLE II
CRITICAL PATH DELAYS [ns]

Mult.	Delay [ns, %]	
16-bit	9.08	100%

(a) Conventional

Mult.	Mode	Delay [ns, %]	
16-bit	16-bit	9.11	100.4%
16-bit	One 8-bit	8.55	94.2%
16-bit	Two 8-bit	8.70	95.9%

(b) Twin-precision

Table II shows the critical path delays that PathMill found for the different modes of the twin-precision multiplier and the conventional modified-Booth multiplier. The result shows that the twin-precision multiplier is as fast as the conventional multiplier when performing 16-bit multiplications. The delay difference reported by PathMill is an insignificant 32 ps. The delay improvements for the 8-bit multiplications is not as impressive as the reduction in total power dissipation though. This is because the delay through the recoder and the final adder does not change between the 16-bit and 8-bit mode. Further, since the advantage of the modified-Booth algorithm is that it reduces the number of rows in the reduction tree, the delay improvement for the 8-bit case is less than for an array or a radix-2 tree multiplier. As the size of the multiplier increases, the constant delay through the recoder and the final adder is expected to become smaller relative the delay through the reduction tree, so larger delay improvements for *N*/2-bit multiplications can be expected for larger N than 16.

VI. Conclusion

The twin-precision concept has successfully been applied to the modified-Booth multiplier, through the implementation of a 16-bit modified-Booth twin-precision multiplier which is also capable of performing single 8-bit or two concurrent 8-bit multiplications. The low or nearly nonexistent delay overhead for the modified-Booth twin-precision multiplier makes it an interesting design choice, when seeking to reduce the total power dissipation or increase throughput of low-precision operands.

The 16-bit modified-Booth twin-precision multiplier as well as the reference 16-bit modified-Booth multiplier will be manufactured shortly. Measurements will be conducted in the fall of 2005. An attempt will be made to try to convert the circuit netlist to the 0.13-μm process used in [1] in order to compare the power and delay of the radix-2 tree and the modified-Booth multiplier.

VII. Acknowledgment

We would like to acknowledge Camilla Berglund, Lars Johansson, and Mikael Samuelsson for their 16-bit twin-precision modified-Booth multiplier implementation, which is a part of the final year VLSI design project at Chalmers. We would also like to thank the Swedish Foundation for Strategic Research (SSF) for funding, through the FlexSoC project.

References

[1] M. Själander, H. Eriksson, and P. Larsson-Edefors, “An Efficient Twin-Precision Multiplier,” in *Proceedings of the 22nd International Conference on Computer Design*, Oct. 2004.
[2] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits, A design Perspective*. Prentice Hall Electronics and VLSI series, 2003, no. 0-13-597444-5, ch. 11, pp. 587–589.
[3] Y. Wen-Chang and J. Chein-Wei, “High-speed booth encoded parallel multiplier design,” *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 692–701, 2000.
[4] J. Fadavi-Ardekani, “MxN Booth Encoded Multiplier Generator Using Optimized Wallace trees,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, no. 2, pp. 120–125, 1993.
[5] P. M. Kogge and H. S. Stone, “A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations,” *IEEE Transactions on Computers*, vol. 22, no. 8, pp. 786–793, Aug 1973.
[6] *Spectre user guide, Version 5.0.33.092203 – 23 Sep 2003.*
[7] A. Abddollahi, M. Pedram, F. Fallah, and I. Ghosh, “Precomputation-based Guarding for Dynamic and Leakage Power Reduction,” in *Proceedings of the 21st International Conference on Computer Design*, 2003, pp. 90–97.
[8] *PathMill user guide, Version U-2003.03-SP1.*