THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Efficient Reconfigurable Multipliers Based on the Twin-Precision Technique

MAGNUS SJÄLANDER

Division of Computer Engineering
Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2006

Efficient Reconfigurable Multipliers Based on the Twin-Precision Technique
Magnus Själander

Technical report 12L
ISSN 1652-876X
Department of Computer Science and Engineering
VLSI Research Group

Division of Computer Engineering
Chalmers University of Technology
SE-412 96 GÖTEBORG, Sweden
Phone: +46 (0)31-772 10 00

Author e-mail: magnus@sjalander.com

Cover:
Picture by the author that shows the twin-precision technique applied to the reduction tree of the High Performance Multiplier.

Printed by Chalmers Reproservice
Göteborg, Sweden 2006

Efficient Reconfigurable Multipliers Based on the Twin-Precision Technique

Magnus Själander

Division of Computer Engineering, Chalmers University of Technology

ABSTRACT

During the last decade of integrated electronic design ever more functionality has been integrated onto the same chip, paving the way for having a whole system on a single chip. The strive for ever more functionality increases the demands on circuit designers that have to provide the foundation for all this functionality. The desire for increased functionality and an associated capability to adapt to changing requirements, has led to the design of reconfigurable architectures. With an increased interest and use of reconfigurable architectures there is a need for flexible and reconfigurable computational units that can meet the demands of high speed, high throughput, low power, and area efficiency.

Multiplications are complex to implement and they continue to give designers headaches when trying to efficiently implement multipliers in hardware. Multipliers are therefore interesting to study, when investigating how to design flexible and reconfigurable computational units.

In this thesis the results from investigations on flexible multipliers are presented. The new twin-precision technique, which was developed during this work, makes a multiplier able to adapt to different requirements. By adapting to actual multiplication bitwidth using the twin-precision technique, it is possible to save power, increase speed and double computational throughput. The investigations have also led to the conclusion that the long used and popular modified-Booth multiplier is inferior in all aspects to the less complex Baugh-Wooley multiplier. During this work, a VHDL multiplier generator was created and made publicly available.

Keywords: High-Speed, Low-Power, Multipliers, Reconfigurable, Twin-precision, VLSI

Preface

Parts, but far from all, of the contributions presented in this thesis have previously been accepted to conferences or submitted to journals.

- **M. Själander**, H. Eriksson, and P. Larsson-Edefors, "An Efficient Twin-Precision Multiplier," in *IEEE International Conference on Computer Design*, San Jose, United States of America, October 10–13, 2004, pp. 507–510.
- **M. Själander**, M. Draždžiulis, P. Larsson-Edefors, and H. Eriksson, "A Low-Leakage Twin-Precision Multiplier Using Reconfigurable Power Gating," in *IEEE International Symposium on Circuits and Systems*, Kobe, Japan, May 23–26, 2005, pp. 1654–1657.
- **M. Själander** and P. Larsson-Edefors, "A Power-Efficient and Versatile Modified-Booth Multiplier," in *Swedish System-on-Chip Conference*, Tammsvik, Sweden, April 18–19 2005.
- **M. Själander** and P. Larsson-Edefors, "Comprehensive Evaluation of a Modified-Booth Multiplier with Logarithmic Reduction Tree," submitted to *IEEE Transaction on Very Large Scale Integrated Systems*, January 20th 2006.
- 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 *IEEE International Symposium on Circuits and Systems* Island of Kos, Greece, May 21–24, 2006

Acknowledgments

I am grateful to the following people for what they have done for me, for my career, and for this thesis.

- ▷ Professor Per Larsson-Edefors who is not only excellent to discuss technical aspects of my research with, but who cares about me, who helps during the long and late hours before a deadline, who goes for lunch with me and who is a friend to trust.
- ▷ Lic. Eng. Mindaugas Draždžiulis, for teaching me everything I know about leakage reduction and for our fruitful research collaborations that led us to Japan.
- ▷ Martin Thuresson, my personal DJ, that has had to withstand me during our years at Chalmers (or was it the other way around). For many long and interesting discussions and numerous poker nights.
- ▷ Lic. Eng. Daniel Andersson for interesting conversations and for sharing my newly found interest in playing squash.
- ▷ Dr. Henrik Eriksson for helping me start my research on twin-precision multipliers by basing it on the High Performance Multiplier and all the support for understanding the finer details of a multiplier.
- ▷ Dr. Daniel Eckerbert for his wast knowledge in circuit design and for introducing me to the tools that are so critical for our line of research.
- ▷ Dr. Jim Nilsson for our early brainstorm on reconfigurable multipliers.
- ▷ Docent Lars 'J' Svensson for our combined struggles in FlexSoC.

▷ Sten Gunnarsson, for putting me in contact with Professor Per Larsson-Edefors and for the time spent together hanging from a rock-face in the western parts of Sweden.

▷ Conny Olsson, for his consultation on the finer details of the layout of this thesis and for always welcoming me to his home, when I'm visiting my hometown Östersund.

▷ Magnus Almgren, Ana Bosque Arbiol, Charlotta, Bååth, Minh Quang Do, Jochen Hollman, Wolfgang John, Henrik Larsson, Jessica Leong, Lina Ohlsson, Greger Sundqvist, and Marcus Östberg, my good friends.

▷ Pierre Kleberger for interesting discussions on Linux related topics.

▷ Lars Kollberg for always prioritizing problems with the EDA tools.

▷ Camilla Berglund, Lars Johansson, and Mikael Samuelsson for their excellent work on the implementation of a modified-Booth multiplier.

▷ My colleagues at the Division of Computer Engineering that creates a good working environment.

▷ The Swedish Foundation for Strategic Research (SSF) for their funding of my position at Chalmers University of Technology.

▷ My beloved grandmother, Sonja Kallin, for always supporting me and for letting me stay at her place when I was working a year in Östersund, thus, allowing me to start my academic career.

▷ My uncle, Harald Kallin, for being a role model and inspiring me to take my Masters degree in Computer Engineering.

▷ My parents Thomas and Ingeborg Själander and my brother Lars Själander who have supported me through life and made this thesis possible.

I dedicate this thesis to the memories of my dear friends Ulf Allberg and Annika Jämtlid and their daughter Lovisa, who were enjoying the weather in Khao Lak when the tsunami came 29th of December, 2004.

Magnus Själander
Göteborg, February 2006

Contents

List of Figures

Part I

INTRODUCTION

1

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 run on that digital system. The bitwidth of the multiplier is, therefore, often much larger than its operands, which leads to excessive power dissipation and long delay. This could partially be remedied by having several multipliers, each with a specific bitwidth, and using the particular multiplier with the smallest bitwidth that is large enough for 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. Several multipliers with the same bitwidth could also be

used in a Single Instruction Multiple Data (SIMD) fashion, in order to increase throughput, thus reducing total execution time for multiplication-intensive applications. However, using several multipliers with different bitwidths would not be an efficient solution because of area overhead due to having multiple multiplier instances, and power overhead due to static power dissipation of inactive multipliers.

Ever increasing performance requirements make it challenging to implement multipliers that are efficient in terms of throughput, delay, power, and area for a wide range of bitwidths.

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. 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 [2, 3]. Narrow-width operands have also been used to increase instruction throughput, by computing several narrow-width operations in parallel on a full-width datapath [1, 4].

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 media benchmarks, the power reduction of an operand guarded integer unit was 54% and 58%, respectively, which accounts for a total power reduction of around 5-6% for an entire datapath.

Loh [4] and Brooks *et al.* [1] have also investigated the possibility of increasing execution bandwidth by double pumping (doubling the throughput) an integer Arithmetic Logic Unit (ALU) with narrow-width operations. Loh 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.* did a similar investigation, where they envisioned a 64-bit adder that could be separated into four 16-bit adders by breaking the carry chain.

There have been several studies on operand guarding for multipliers. Huang *et al.* [2] introduced a 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 was 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 to increase the throughput of a multiplier is more scarce. Achieving double throughput for a multiplier is not as straightforward as for 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 this 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 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. This technique requires extra reduction stages for the larger multipliers, which has a negative impact on the delay for these.

We present the twin-precision technique [7] that offers the same power reduction as operand guarding *and* the possibility of double-throughput multiplications. The twin-precision technique is an efficient way of achieving double throughput in a multiplier with low area overhead and with practically no delay penalty. We show how to apply the twin-precision technique on signed multipliers based on the regular High Performance Multiplier (HPM) reduction tree. The two algorithms for signed multiplications that have been used are Baugh-Wooley and the popular modified-Booth algorithm.

1.1 Twin-Precision Fundamentals

For a first analysis of the twin-precision technique, the discussion will be based on an illustration of an 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 (Eq. 1.1). 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. 1.1.

$$p_{ij} = y_i x_j \tag{1.1}$$

								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

Figure 1.1: *Illustration of an unsigned 8-bit multiplication.*

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. 1.2.

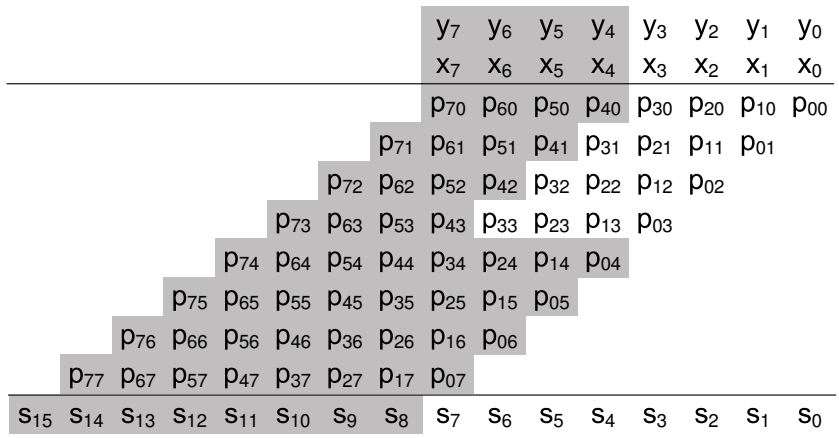


Figure 1.2: *Illustration of an unsigned 8-bit multiplication, where the precision of the operands is smaller than the precision of the multiplication. Unused bits of operands and product, as well as unused partial products, are shown in gray.*

Fig. 1.2 shows that large parts of the partial products are only containing zeros and are, thus, not contributing with any useful information for 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. 1.2, 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 signif-

icant 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. 1.3. By setting the other partial products to zero, it is then 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 investigated in the implementation section later on.

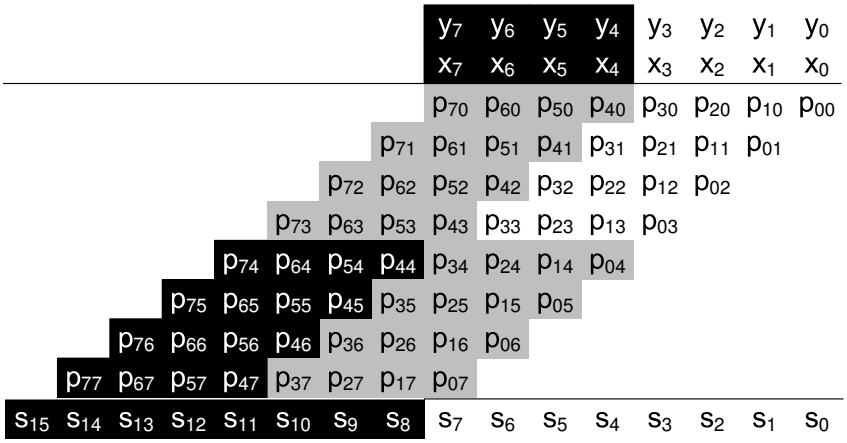


Figure 1.3: *Illustration of an unsigned 8-bit multiplication, where a 4-bit multiplication, shown in white, is computed in parallel with a second 4-bit multiplication, shown in black.*

Assume, for now, that there is a way of setting unwanted partial products to zero, then 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_{FULL}) of the multiplication, Eq. 1.2. 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.

$$N_{FULL} \geq N_{LSP} + N_{MSP} \tag{1.2}$$

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 need to be smaller or equal to the full precision multiplication.

$$N_{FULL} \geq \sum_{i=1}^{K} N_i \tag{1.3}$$

For the rest of this investigation, the precision of the two smaller multiplications will be equal and half the precision ($N/2$) of the full precision (N) of the multiplier.

1.1.1 A First Implementation

The basic operation of generating a partial product is that of a 1-bit multiplication using a 2-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[1]. For this first implementation an array of adders will be used because of its close resemblance to the previously used illustration of a multiplication.

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 2-input AND gate to a 3-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 has to be set to zero need to be changed to a 3-input version. During normal operation when a full-precision multiplication is executed the control signal is set to high,

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

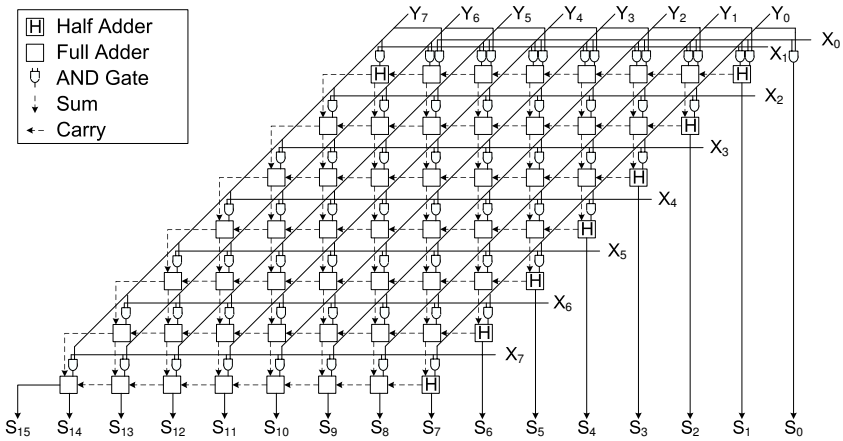


Figure 1.4: *Block diagram of an unsigned 8-bit array multiplier.*

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 are 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. 1.5. The two multiplications have been colored in white and black to visualize what part of the adder array is used for what multiplication.

More flexibility might be wanted, like the possibility to compute a single low-precision multiplication or two parallel low-precision multiplications, within the same multiplier. This can be done by changing the 2-input AND gates for the partial product generation of the low-precision multiplication as well. In the array multiplier in Fig. 1.5, the AND gates for the 4-bit MSP multiplication, shown in black, can be changed to 3-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.

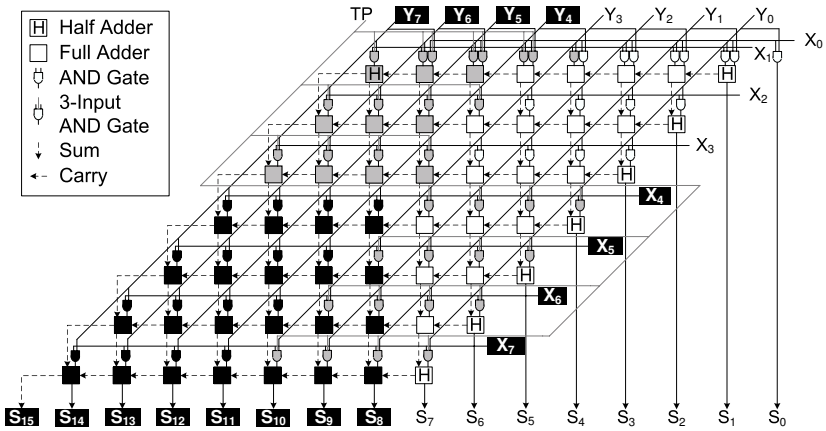


Figure 1.5: *Block diagram of an unsigned 8-bit twin-precision array multiplier. The TP signal is used for controlling if one full-precision multiplication should be computed or two 4-bit multiplications should be computed in parallel.*

1.1.2 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 logarithmic reduction tree such as the TDM [8], Dadda [9], Wallace [10] or HPM [11] is preferred for summation of the partial products. A logarithmic reduction tree has the benefit that the logic depth is shorter. Further, a logarithmic tree has fewer glitches making it less power dissipating. A twin-precision implementation based on the regular HPM reduction tree is shown in Fig. 1.6.

1.2 A Baugh-Wooley Implementation

In the previous section, the concept of twin precision was introduced by looking at an unsigned multiplication. 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.

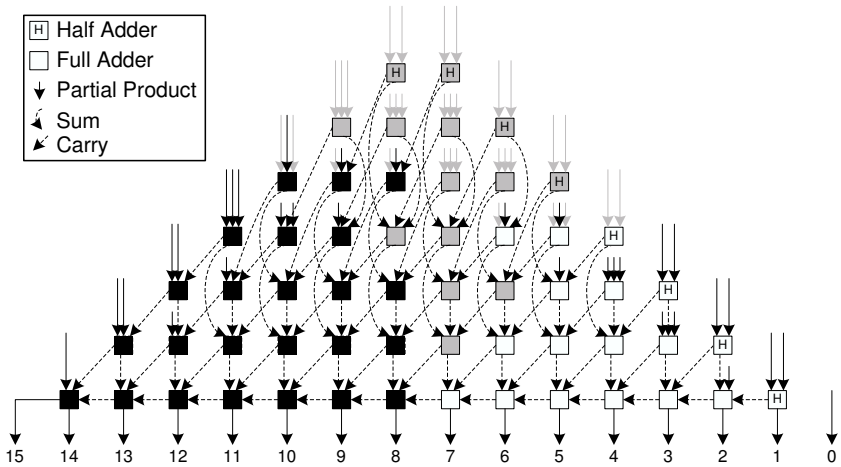


Figure 1.6: *Block diagram of an unsigned 8-bit twin-precision multiplier that is based on the regular HPM reduction tree. A 4-bit multiplication, shown in white, can be computed in parallel with a second 4-bit multiplication, shown in black. For simplicity of the figure the* AND/NAND *gates for partial product generation is not shown and a ripple carry is used as final adder.*

1.2.1 Algorithms for Baugh-Wooley

The BW algorithm [12] is a relative straightforward way of doing signed multiplications. Fig. 5.4 illustrates the algorithm for an 8-bit case, where the partial-product array has been reorganized according the the scheme of Hatamian [13]. 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 has 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.

							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
						1	$\overline{p_{70}}$	p_{60}	p_{50}	p_{40}	p_{30}	p_{20}	p_{10}	p_{00}
						$\overline{p_{71}}$	p_{61}	p_{51}	p_{41}	p_{31}	p_{21}	p_{11}	p_{01}	
					$\overline{p_{72}}$	p_{62}	p_{52}	p_{42}	p_{32}	p_{22}	p_{12}	p_{02}		
				$\overline{p_{73}}$	p_{63}	p_{53}	p_{43}	p_{33}	p_{23}	p_{13}	p_{03}			
			$\overline{p_{74}}$	p_{64}	p_{54}	p_{44}	p_{34}	p_{24}	p_{14}	p_{04}				
		$\overline{p_{75}}$	p_{65}	p_{55}	p_{45}	p_{35}	p_{25}	p_{15}	p_{05}					
	$\overline{p_{76}}$	p_{66}	p_{56}	p_{46}	p_{36}	p_{26}	p_{16}	p_{06}						
p_{77}	$\overline{p_{67}}$	$\overline{p_{57}}$	$\overline{p_{47}}$	$\overline{p_{37}}$	$\overline{p_{27}}$	$\overline{p_{17}}$	$\overline{p_{07}}$							
$\overline{s_{15}}$	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

Figure 1.7: *Illustration of a signed 8-bit multiplication, using the Baugh-Wooley algorithm.*

1.2.2 Twin-Precision Using the Baugh-Wooley Algorithm

To combine twin-precision with BW is not as simple as for the unsigned multiplication, where only parts of the partial products needed 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. 1.8 shows an illustration of an 8-bit BW multiplication, where two 4-bit multiplications have been depicted in white and black.

When comparing the illustration of Fig. 1.7 with that of Fig. 1.8 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. Looking at the active partial-product array of the 4-bit LSP multiplication (shown in white), we see that 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 result needs to be negated to get the correct result of the two 4-bit multiplications.

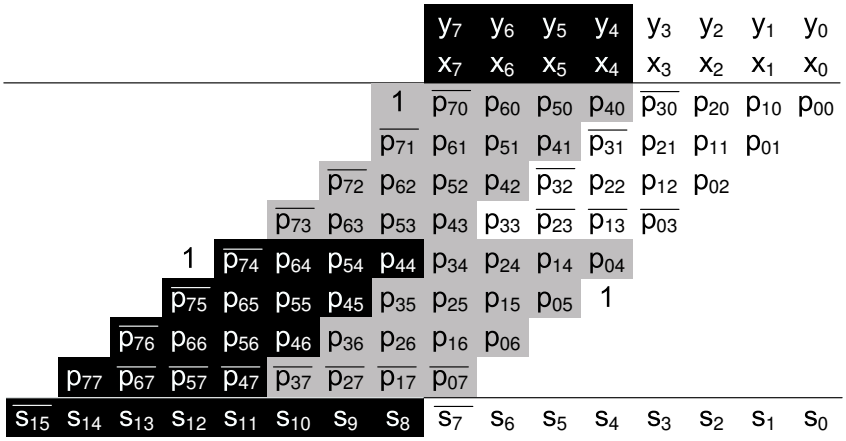


Figure 1.8: *Illustration of a signed 8-bit multiplication, using the Baugh-Wooley algorithm, where a 4-bit multiplication, shown in white, is computed in parallel with a second 4-bit multiplication, shown in black.*

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 2-input AND gate that generates the partial product to a 2-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 control signal for the XOR gates used in the partial-product generation.

Setting unwanted partial products to zero can be done by 3-input AND gates as for the unsigned case.

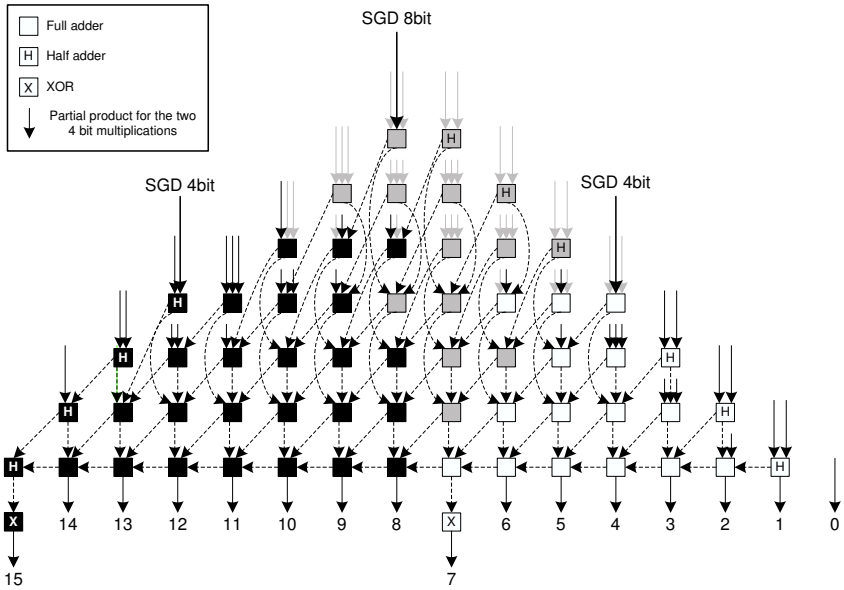


Figure 1.9: *Block diagram of a signed 8-bit multiplication, using the Baugh-Wooley algorithm, where a 4-bit multiplication, shown in white, is computed in parallel with a second 4-bit multiplication, shown in black.*

Fig. 1.9 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. 1.6 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 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.

1.3 A Modified-Booth Implementation

Modified Booth (MB) is a popular algorithm and commonly used for implementation of signed multipliers. MB is a more complicated algorithm for signed multiplication 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.

1.3.1 Algorithms for Modified Booth

The original-Booth algorithm [14] 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 x operand and coding them into $\{-2, -1, 0, 1, 2\}$. The encoded number is then multiplied with the second operand, y, into a row of recoded partial products. The number of recoded partial products is fewer than for a scheme with unrecoded partial products and this can be translated into higher performance.

The drawback of the original-Booth algorithm is that the number of generated partial products depends on the x operand, which makes the Booth algorithm unsuitable for implementation in hardware. The MB algorithm [15] by MacSorley remedies this by looking at three bits at a time of operand x. Then we are guaranteed that only half the number of partial products will be generated, compared to a conventional partial product generation using 2-input AND gates. With a fixed number of partial products the MB algorithm is suitable for hardware implementation. Fig. 1.10 shows which parts of the x operand that are encoded and used to recode the y operand into a row of partial products.

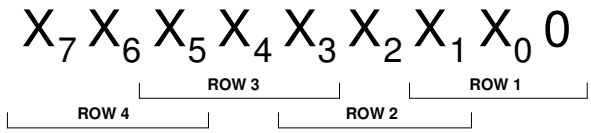


Figure 1.10: *8-bit modified-Booth encoding.*

A 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 y operand. To avoid having to sign extend the rows of of recoded partial products, the sign-extension prevention scheme presented by Fadavi-Ardekani [16] has been used. 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.* [17], 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. An 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 Eq. 1.4. The pre-computation of a potential carry is as given by Eq. 1.5. Here Eq. 5.2 is different from that used by Yeh *et al.* [17].

$$p_{LSBi} = y_0(x_{2i-1} \oplus x_{2i}) \tag{1.4}$$

$$a_i = x_{2i+1}(\overline{x_{2i-1} + x_{2i}} + \overline{y_{LSB} + x_{2i}} + \overline{y_{LSB} + x_{2i-1}}) \tag{1.5}$$

An illustration of an 8-bit MB multiplication with the sign-extension prevention and modified partial-product array scheme can be seen in Fig. 1.11.

1.3.2 Twin-Precision Using the modified-Booth Algorithm

Implementing twin-precision together with the MB algorithm is not as straightforward as for the BW implementation (Section 1.2). It is not possible to take

								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

Figure 1.11: *Illustration of a signed 8-bit multiplication using the modified-Booth algorithm.*

the partial products from the full-precision MB multiplication and use only the partial products that are of interest for the low-precision 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.

								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_{42}	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

Figure 1.12: *Illustration of a signed 8-bit multiplication using the modified-Booth algorithm.*

By comparing the two multiplications in Fig. 1.11 and Fig. 1.12 (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. 1.11) need to define partial products that are used to prevent sign extension in the low-precision 4-bit multiplication in the LSP (Fig. 1.12).

- 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 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.

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 Fig. 1.11 and Fig. 1.12, 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_{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. 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. 1.13.

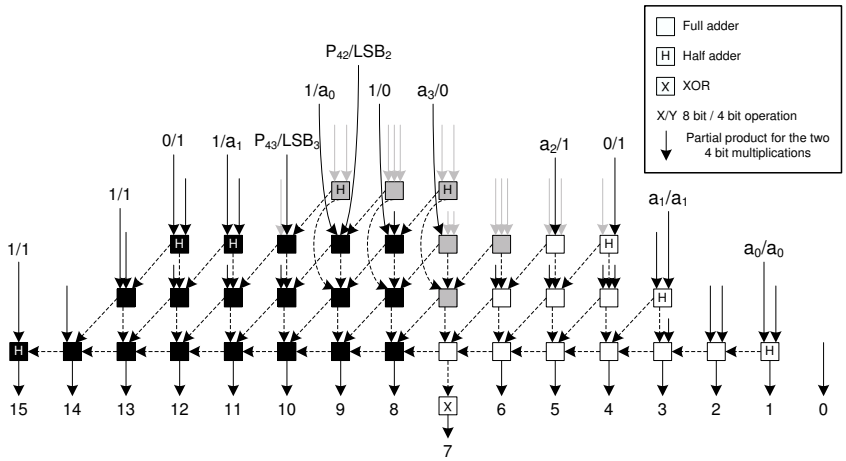
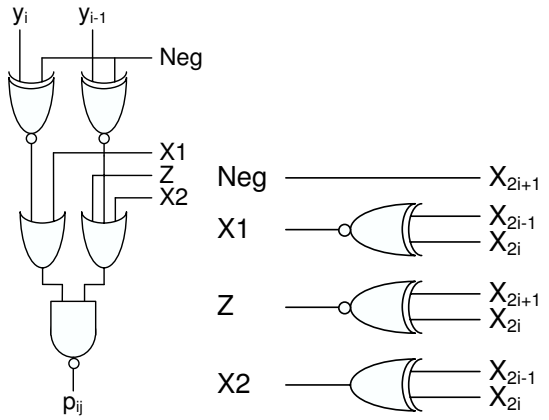


Figure 1.13: *Block diagram of a signed 8-bit multiplication using the modified-Booth algorithm, where a 4-bit multiplication, shown in white, is computed in parallel with a second 4-bit multiplication, shown in black.*

The logic encode and decode can be implemented in many different ways. For this implementation we have chosen the encoding scheme presented by Yeh *et al.* [17], since they claim that their recoding scheme is faster than competing schemes. The circuits for encoding and decoding is shown in Fig. 1.14.

For the partial products that need to be set to zero, an extra 2-input AND gate has been added at the output of the decode stage. The second input of the AND gate can then be used as a control signal, as in the case of for the unsigned and BW implementations. This is a straightforward method, and it is possible to construct even more efficient solutions for setting the partial product to zero. A decode circuit based on a custom layout, capable of setting the output to zero, can be found in PAPER III [18]. The outputs of the encoders have also been set to zero by using AND gates. This is not necessary for correct operation, but it reduces the power dissipation when computing $N/2$-bit multiplications due to reduced switching.



(a) Decode (b) Encode

Figure 1.14: *Encode and decode circuit for modified Booth.*

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 can be seen in Fig. 1.15.

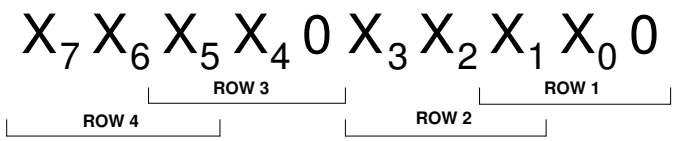


Figure 1.15: *Encoding scheme for two 4-bit multiplications.*

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 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', 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'.

1.4 Simulation Setups

To evaluate the efficiency of the twin-precision technique a multiplier generator has been 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 the previous sections (1.2 and 1.3). A Kogge-Stone [20] adder was chosen as final adder for all types of multipliers. The VHDL generator has been 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 a finite random input pattern and verifying the result.

The VHDL descriptions were synthesized using Design Compiler [22] by Synopsys together with a commercially available 0.13-μm technology. For delay and power estimations, Synopsys PrimeTime [23] and PrimePower [24] were used. The delay has been estimated by assuming a 10 fF load on the primary output signals and a medium-size buffer from the cell library as driver for primary input signals. The chosen buffer cell is specified to be capable of driving approximately 150 minimum-size XOR gates. This is of course an overkill for the smaller multipliers of this evaluation but the same driver has been used for consistency of our analysis. For average power analysis, a clock period of 5 ns (200 MHz) and PrimePower's default values for switching activity. Further, it was assumed that the twin-precision multipliers will operate in the same mode (Full, 1x$N/2$, or 2x$N/2$ precision) for a long period of time. The power

for the control signals has not been included, since these low-activity signals would only have a minor contribution to the total power. All simulations were conducted with a supply voltage V_{dd} of 1.3 V and default temperatures for the respective tools.

The multipliers have not been taken through place and route, but as will be shown later in this section the area overhead of the twin-precision multipliers is not significant. Thus the total wire length of the multipliers is expected to be equal and so is the effect of cross talk.

1.4.1 Synthesized Baugh-Wooley Netlist

The synthesis and the timing analyses 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 2-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 losing any significant performance[2]. The mapping of the BW multiplier VHDL code to gate-level netlist during synthesis was only constrained in the way that half and full adders of the reduction tree was mapped to their respective minimum-size cells and the map effort in Design Compiler was set to high.

1.4.2 Synthesized Modified-Booth Netlist

The first attempt of synthesizing the VHDL code for the 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. 1.14. 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

[2]This observation is based on medium-sized drivers for primary input signals.

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 X1, Z and X2 signals fanout have been reduced by instantiating multiple encoders for each row of partial products. To limit the fanout of the x_{2i+1} signal an inverter buffer has been inserted to drive the NEG signal. Finally min-size inverters have been added to x_{i-1}, x_i, and x_{i+1}, inputs of the encoder, thus, the fanout of primary x-inputs is reduced. The new encode circuit can be viewed in Fig. 1.16.

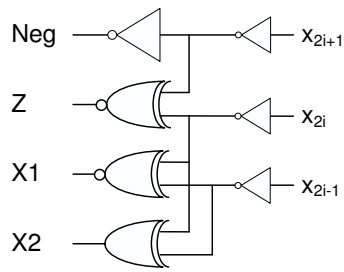


Figure 1.16: *Buffered encode circuit for fanout reduction.*

Fig. 1.17 presents a graph showing the effect on total delay when a maximum of 4, 8, 16, or 32 decoders have been connected to each of the new encoders of Fig. 1.16. The graph clearly shows the benefit in speed by reducing the fanout of the encoders, thus we have chosen to limit the maximum number of decoders connected to each encoder to four. The effect on power and area of the choice of fanout will be discussed later in this section.

The mapping of the MB multiplier VHDL code to gate-level netlist has been constrained in such way that Design Compiler could not remove the minimum-size inverters of the new encoder, Fig. 1.16. In all other respects the synthesized netlist of the MB multiplier has been constrained in the same way as the BW netlist. In other words half and full adders of the reduction tree have been mapped to their respective cells of minimum size from the cell library and the map effort was set to high.

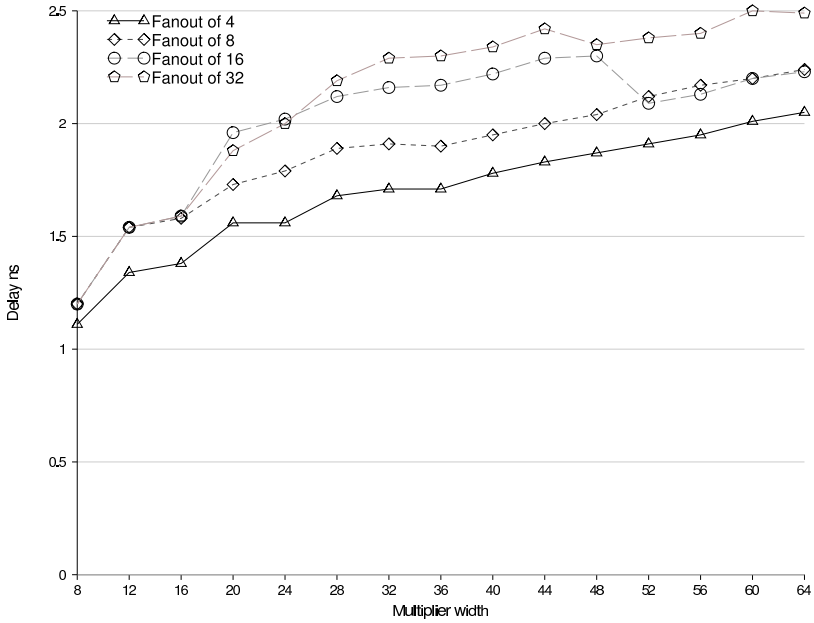


Figure 1.17: *Impact on the total delay for different fanout of the encoder.*

1.5 Results and Discussion

It is possible to operate the twin-precision multiplier in three different modes: *i)* the full-precision mode, denoted N, *ii)* the single $N/2$-bit mode, denoted 1x$N/2$, and *iii)* the double-throughput $N/2$-bit mode, denoted 2x$N/2$.

1.5.1 Power Dissipation

One of the reasons to use the twin-precision technique for executing low-precision multiplications is to reduce power dissipation of the multiplier[3]. Dynamic power is directly dependent on the switching activity (α_i) in each node. Eq. 1.6 gives the switching power given a switching voltage of V_{dd} and a load capacitance in each node of C_{Li} for a circuit operating with a frequency of f_{clk}.

[3]The other reason is the possibility of executing two multiplications concurrently.

$$P_{switching} = \sum \alpha_i C_{Li} V_{dd}^2 f_{clk} \tag{1.6}$$

When operating a twin-precision multiplier in 1x$N/2$ or 2x$N/2$ mode large parts of the partial-product array are forced to zero. This in turn has the effect that the total switching activity is significantly reduced, since large parts of the logic are kept at a constant level. This has a dramatic effect on total power dissipation.

Table 1.1: *Average power dissipation per operation for an N-bit multiplier operating at 200 MHz.*

	Conventional	Twin-precision N-bit		
	N-bit	N	1x$N/2$	2x$N/2$
Baugh-Wooley	100%	103%	35%	32%
Modified Booth	151%	153%	98%	66%

As can be seen in Table 1.1, there is a significant reduction in power when operating an N-bit twin-precision multiplier in 1x$N/2$-bit or 2x$N/2$-bit mode. When operating in 2x$N/2$-bit mode, the total power dissipation is of course larger than for 1x$N/2$-bit mode, but since the two $N/2$-bit multiplications are computed concurrently the power dissipation per multiplication becomes lower. A twin-precision multiplier has a small overhead when operating in full N precision mode compared to a conventional multiplier. Table 1.1 also shows that a twin-precision implementation of modified Booth (MB) is not as efficient as a twin-precision implementation of Baugh-Wooley (BW). MB is clearly not suitable for low-power implementations.

Table 1.2: *Average power dissipation per operation, comparing a conventional $N/2$-bit multiplier with an N-bit twin-precision multiplier when operating at 200 MHz.*

	Conventional	Twin-precision N-bit	
	$N/2$-bit	1x$N/2$	2x$N/2$
Baugh-Wooley	100%	136%	128%
Modified Booth	151%	384%	251%

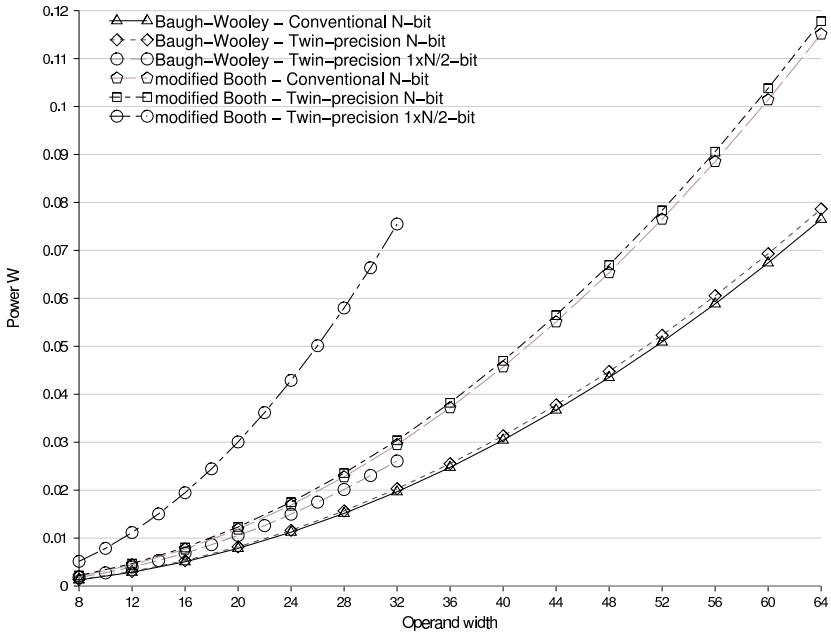


Figure 1.18: *Total power in W for conventional Baugh-Wooley and modified Booth as well as twin-precision versions of both, when operating at 200 MHz. For the twin-precision implementations, the power for a single $N/2$ multiplication is also given.*

Fig. 1.18 shows total power dissipation for both conventional BW and MB multipliers and their respective twin-precision implementations. As can be seen, when operating a twin-precision multiplier in 1x$N/2$ mode (as well as in 2x$N/2$ mode, which is not shown in the graph) the power dissipation is slightly higher than for a conventional multiplier that is of size $N/2$. Table 1.2 shows the average difference in power dissipation.

1.5.2 Delay

The twin-precision technique was conceived with a minimum delay overhead in mind for the N-bit multiplication. This is the main reason as to why the struc-

ture of the reduction tree has been modified as little as possible. This strategy has proven to be very efficient, the delay overhead for both the BW (Fig. 1.19) and the MB (Fig. 1.20) implementation is less than 1%, (on average 0.7% and 0.6%, respectively). This comes at the cost of a higher delay overhead for the $N/2$-bit multiplication compared to a conventional multiplier of size $N/2$, since the reduction tree and the final adder are not primarily optimized for these precisions.

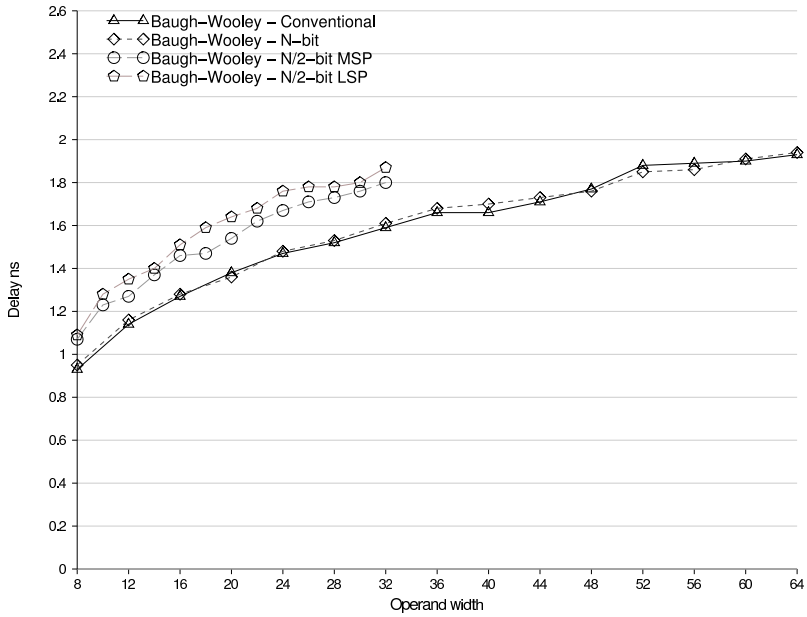


Figure 1.19: *Total delay in ns for a conventional Baugh-Wooley implementation and its twin-precision counterpart.*

As can be seen from Table 1.3 there is a delay overhead for $N/2$-bit multiplication within the twin-precision multiplier when compared to a conventional fixed-size $N/2$-bit multiplier. However, the twin-precision's $N/2$-bit is still 5-10% faster than the full-precision multiplier for the BW implementation and about 6% faster for the MB implementation. The shorter delay of the $N/2$-bit

Table 1.3: *Average delay when comparing a conventional $N/2$-bit multiplier with an N-bit twin-precision multiplier.*

	Conventional	Twin-precision N-bit	
	$N/2$-bit	LSP $N/2$	MSP $N/2$
Baugh-Wooley	100%	118%	115%
Modified Booth	108%	122%	122%

multiplications can be used for either clocking the multiplier faster, thus increasing the throughput or reducing the supply voltage to further reduce power dissipation. As can be seen from Eq. 1.6, power is quadratically dependent on the supply voltage (V_{dd}). The gain in power reduction from reducing the supply voltage in the presence of delay margins has not been investigated.

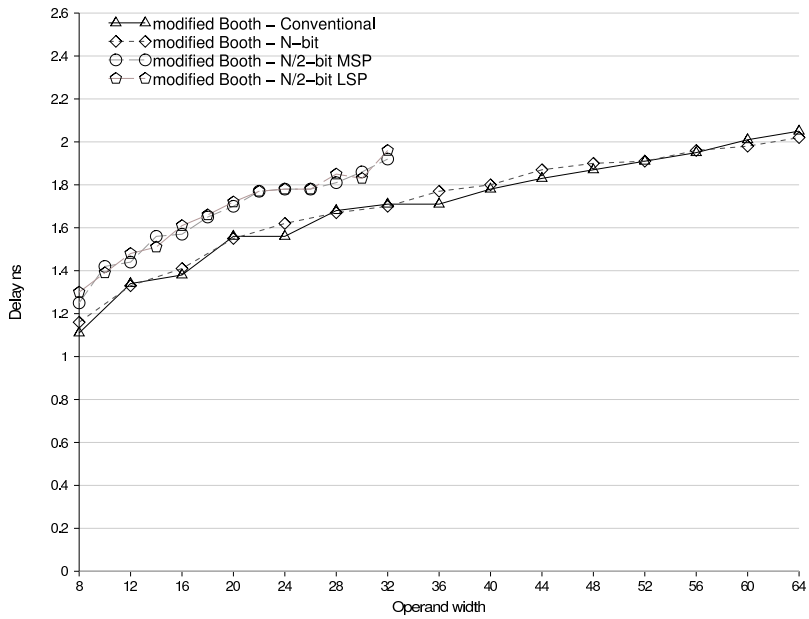


Figure 1.20: *Total delay in ns for a conventional modified-Booth implementation and its twin-precision counterpart.*

1.5.3 Area

We have already considered power and delay for twin-precision implementations of the BW and the MB algorithms. We have learned that there is a large potential to reduce power with an insignificant impact on the delay of the respective multiplier implementation. When it comes to the area, the overhead for a twin-precision implementation is as high as 18% for an 8-bit BW implementation. However, the overhead is rapidly decreasing, becoming close to 6% for larger multipliers. For the MB implementation the overhead goes from a high 13% for an 8-bit multiplier to a low 5.5% for larger multipliers, Fig. 1.21.

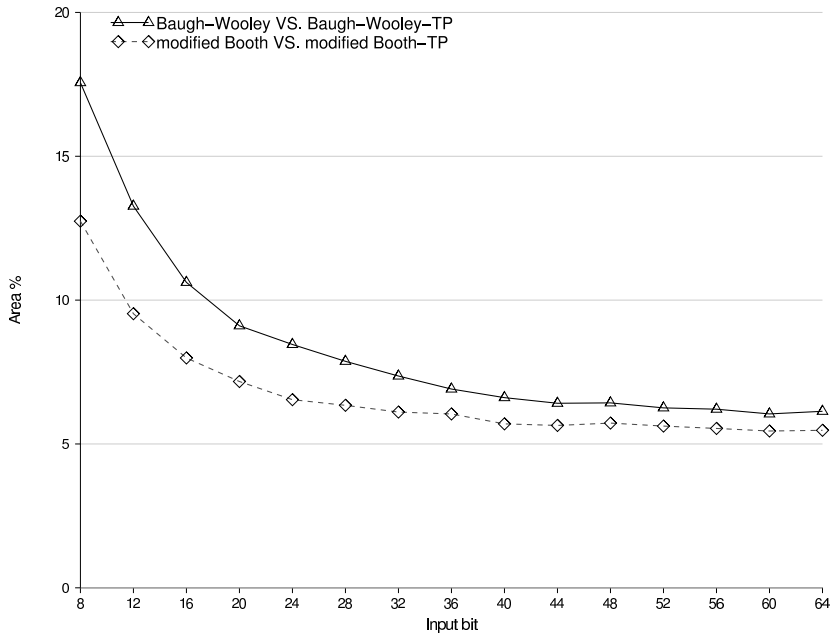


Figure 1.21: *Relative area for conventional Baugh-Wooley and modified-Booth multipliers versus twin-precision versions of both.*

The area overhead for a twin-precision MB implementation is lower than for a twin-precision BW implementation. On the other hand the total area for the

MB implementation is on average 37% larger than for the BW implementation. The area has been estimated as the total size of the logic cells, without area for wiring.

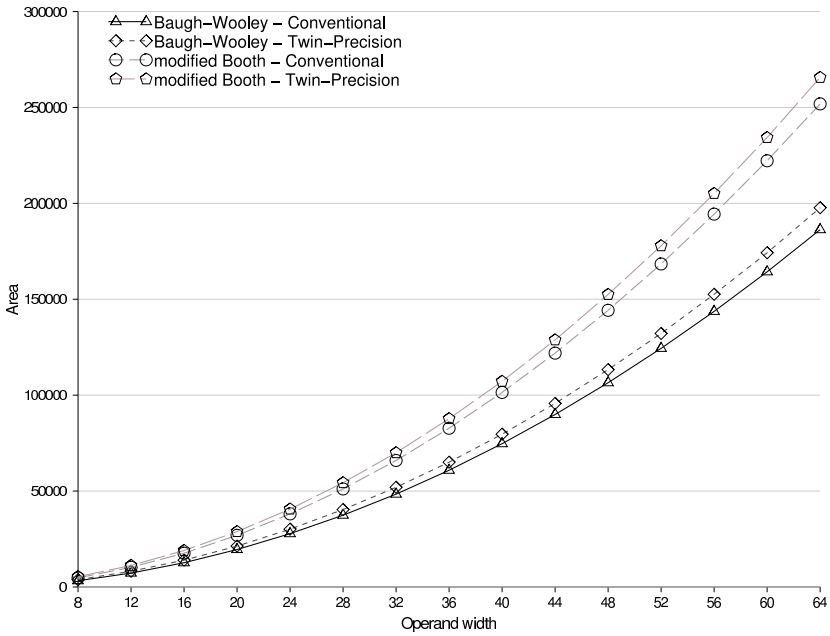


Figure 1.22: *Total area for conventional Baugh-Wooley and modified-Booth multipliers as well as twin-precision versions of both.*

1.5.4 Modified Booth versus Baugh-Wooley

Based on the results presented above, it is clear that a BW multiplier outperforms a state-of-the-art MB multiplier. A detailed comparison was conducted and the findings are presented in PAPER IV [25]. In summary, a conventional (non-TP) MB multiplier is on average 8% slower, 51% more power dissipating, and 37% larger than a conventional (non-TP) BW multiplier.

1.5.5 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 twin-precision BW implementation the overhead is around 30%.

Looking closer at the different contributions to total power, one notices that the contribution due to leakage is quite significant for the $N/2$-bit multiplications, Fig. 1.23. For the 1x$N/2$ mode the static power constitutes more than 15% of the total power dissipation. This is due to the large amount of idle gates. For the 2x$N/2$ mode the leakage is not as large, but is still around 8% of the total power.

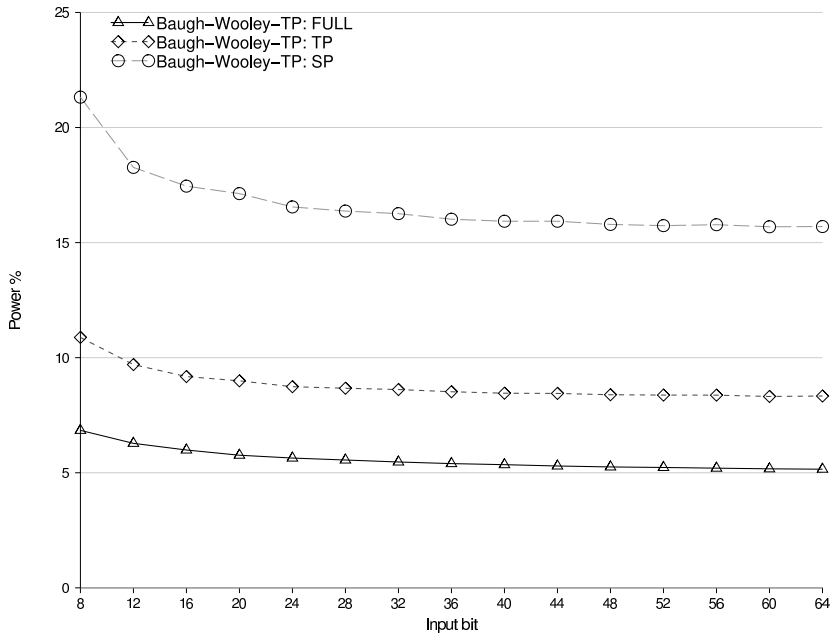


Figure 1.23: *Baugh-Wooley TP static power portion of total power.*

To reduce the fraction of leakage power it is possible to apply power gating, instead of only setting the partial products to zero by the use of 3-input AND gates. Power gating involves adding cut-off transistors to the logic cells that should be gated off when idle. A cut-off transistor is used to isolate 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, is not available in standard-cell libraries.

To be able to apply power gating to different areas of the partial-product generation, reduction tree, and the final adder, a suitable power grid is needed. Fig. 1.24 shows an example of a power grid for an 8-bit twin-precision BW multiplier.

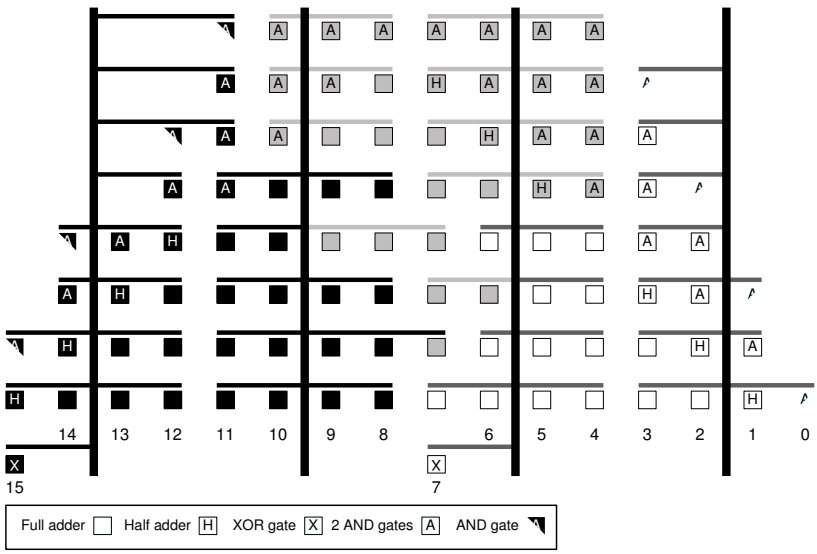


Figure 1.24: *Power grid for an 8-bit twin-precision Baugh-Wooley implementation.*

Previous work [26] (PAPER II) has shown that by using precision-dependent power-gating based on the SCCMOS technique, the power overhead for a 16-bit BW multiplier can be reduced with 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 have changed. However, that work indicates the potential in using precision-dependent power-gating for reducing the power dissipation even further for twin-precision multipliers.

1.5.6 The Impact of the Final Adder on Delay

In this work a Kogge-Stone [20] adder was used as final adder to carry out the last summation with fast carry propagation. The reason this multiplier was chosen is that we were striving for as low delay penalty as possible for the twin-precision implementations, when running at full precision. This penalizes the $N/2$-bit multiplications, since the final adder is optimized for N-bit precision. Comparing the delay profile of the inputs to the final adder for a 32-bit twin-precision BW multiplier, Fig. 1.25 reveals a large dip in the middle of the inputs. This could probably be taken advantage of, to trade a reduced delay penalty of the $N/2$-bit multiplications for a higher delay penalty of the $N/2$-bit multiplications.

1.6 A Case Study

Assume that there is an application that performs 1000 signed multiplications. The application is based on a 32-bit wide multiplication, but through static inspection we find a phase of the application, where the bitwidth of the multiplication is never larger than 16 bits.

Using this scenario we will investigate three different ways to implement the multiplications, when the size/length of the low-precision phase is varied.

- In CASE I a single conventional 32-bit Baugh-Wooley (BW) multiplier is used for all multiplications. This is the conventional way of implementing a multiplier and will be used as a reference for the other cases.

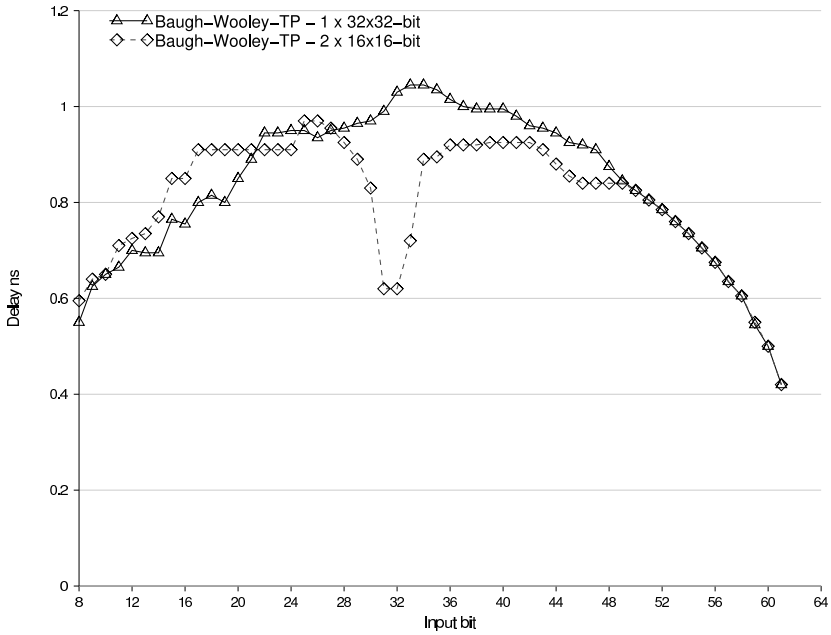


Figure 1.25: *The delay profile for the inputs of the final adder for a twin-precision 32-bit Baugh-Wooley multiplier. The delay for a specific bit is the average of the A and B input.*

- In CASE IIA a single 32-bit and two 16-bit conventional BW multipliers are sharing the same interconnect routing. All three multipliers have been modified, so that when inactive they are operand guarded. This is achieved by changing all 2-input AND gates for the partial-product generation to 3-input AND gates. Multiplexers are used to connect the active multiplier(s) to the result interconnect routing.

- CASE IIB is similar to CASE IIA, but it has a single 16-bit and 32-bit operand-guarded BW multiplier.

- In CASE III a 32-bit twin-precision BW multiplier operates in full-precision (N) mode for normal operation and the double-throughput mode ($2\text{x}N/2$) during the low-precision phase of the application.

CASE IIA and CASE III is assumed to operate in an architecture capable of double-throughput where two 16-bit multiplications can be performed in parallel, as described by Brooks *et al.* [1] and Loh [4].

Using information from Fig. 1.18, Fig. 1.19, Fig. 1.22 and extra simulations for CASE IIA and CASE IIB, the energy, delay, and area have been calculated for the different cases. For all simulations the same input driver have been used for all CASES and only the internal routing for connecting the multipliers together in CASE IIA and IIB have been included.

1.6.1 Total Execution Time

Table 1.4 shows the delay for computing a single 16-bit or 32-bit multiplication for the different cases.

Table 1.4: *Delay in ns for computing one multiplication of 16 or 32 bits*

Bitwidth	CASE I	CASE IIA	CASE IIB	CASE III
16-bit	1.59	0.71	1.42	0.755
32-bit	1.59	1.70	1.70	1.61

As can be seen in Fig. 1.26, only the conventional 32-bit BW multiplier (CASE I) has a constant total execution time, regardless of the number of 16-bit multiplications. For the other cases, the execution time is reduced with the number of 16-bit multiplications. However, because of slightly longer delay for their 32-bit multiplication the other cases perform worse than CASE I when the amount of 16-bit multiplications are low. It is necessary to have at least 12%, 40%, and 2% of 16-bit multiplications for CASE IIA, CASE IIB, and CASE III, respectively, to outperform CASE I. When 16-bit multiplications exceed 66%, CASE IIA becomes the fastest solution.

1.6.2 Total Electric Energy Dissipation

Table 1.5 shows the energy expended for one multiplication of either 16 bits or 32 bits for all different cases. The energy has been calculated by running

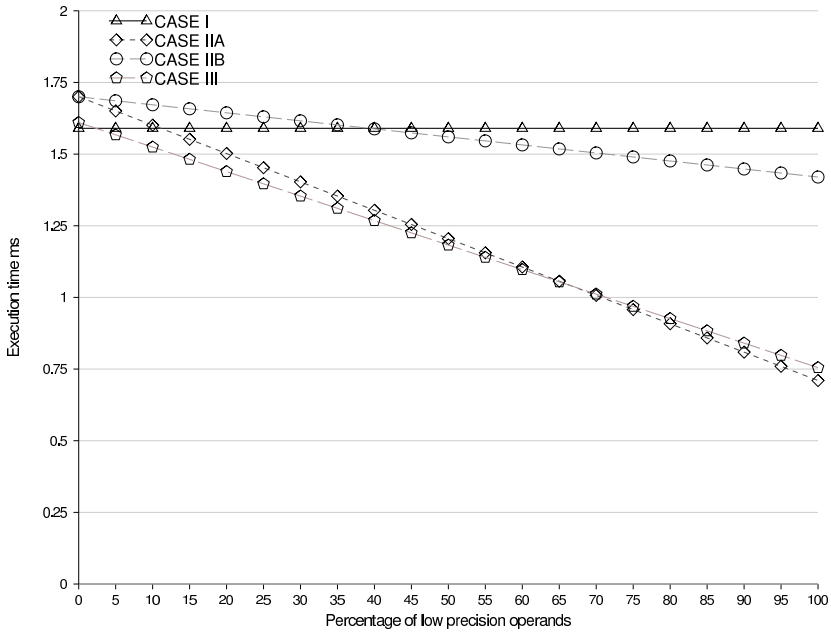


Figure 1.26: *Total execution time for 1000 multiplications where the percentage of low precision multiplications is varied.*

power simulations where each multiplier was operated at maximum speed as given by Table 1.4. The results from the power simulation was then multiplied with respective value from Table 1.4 to get the energy.

Table 1.5: *Energy in fJ for computing one multiplication of 16-bit or 32-bit*

Bitwidth	CASE I	CASE IIA	CASE IIB	CASE III
16-bit	94.445	28.890	30.005	30.570
32-bit	94.445	104.448	102.204	95.682

As seen in Fig. 1.27, the total energy expended for a conventional 32-bit BW multiplier (CASE I) does not change with the amount of 16-bit multiplications. For the other cases there is a dramatic reduction in energy with increased

number of 16-bit multiplications. However, all cases show a higher electric energy dissipation than for (CASE I) when the number of 16-bit multiplications is low. This is due to their overhead for 32-bit multiplications. For CASE III it is necessary to have at least 2% of 16-bit multiplications to achieve a total energy reduction. The same figure for CASE IIA and IIB is 14% and 11%, respectively, of 16-bit multiplications to achieve a total energy reduction. Due to the low energy dissipation for a 16-bit multiplication in CASE IIA, it will eventually have the lowest energy expenditure. This happens when as much as 84% of the multiplications are 16-bit for which CASE IIA uses less energy than CASE III.

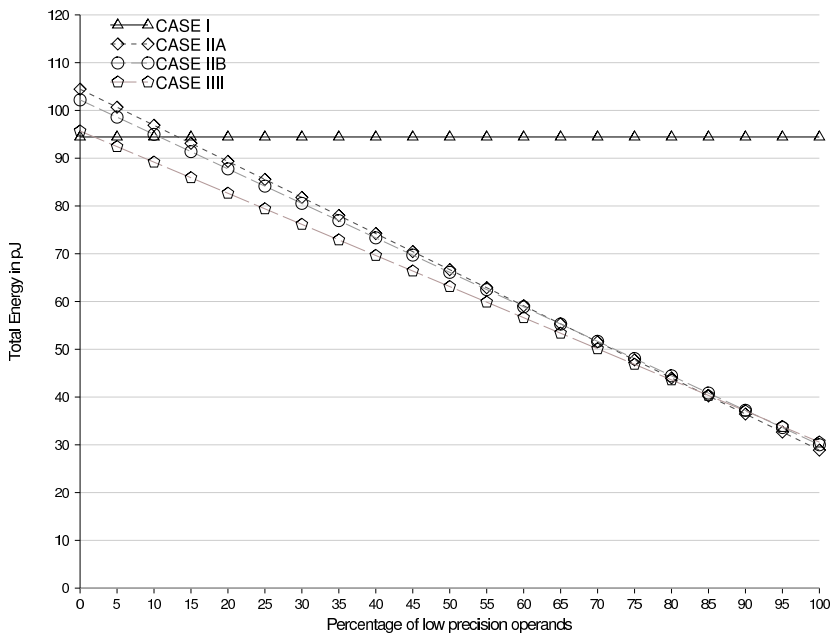


Figure 1.27: *Total energy expended, with the percentage of low-precision multiplications being varied.*

1.6.3 Area

Considering area, CASE III has an overhead of 7.4% compared to the conventional 32-bit BW multiplier (CASE I), while for CASE IIA the area overhead is a staggering 63%. CASE IIB has a more reasonable area overhead of 35% compared to the conventional 32-bit BW multiplier.

Table 1.6: *Total unit area*

CASE I	CASE IIA	CASE IIB	CASE III
48392	79050	65335	51955

1.6.4 Summary

CASE I, where a conventional 32-bit Baugh-Wooley multiplier is used, represents the traditional way of implementing multipliers. CASE IIA, IIB and III all are alternative implementations that have not yet come to be widely used. However, one can conclude that the conventional way of using a single full-precision (32-bit) multiplier, as in CASE I, is not an efficient solution when phases of low-precision (16-bit) multiplications exist. With as few as 2% 16-bit multiplications of the total number of multiplications to be carried out, the twin-precision implementation (CASE III) has both a lower total execution time and uses less energy, with a negligible 7.4% area overhead.

A single 16-bit and a single 32-bit multiplier (CASE IIB) show poor performance in both execution time and expended energy. This is due to the overhead that comes with increased fanout of the primary input signals, for driving the two multipliers, and the necessity of multiplexers that connect the active multiplier to the result routing.

CASE IIA, where two 16-bit and one 32-bit multiplier are used in the same way as in the work of Loh [4], has better performance than when only one 16-bit multiplier is used. This is due to that the same overhead as in CASE IIB is divided between the two concurrent 16-bit multiplications. For a high amount of 16-bit multiplications, this solution actually has the shortest execution time

and uses the least energy. However, the amount of 16-bit multiplications needed for CASE IIA to perform better than CASE III is large (as much as 84%) and the area overhead is a staggering 63% compared to a conventional 32-bit multiplier. Furthermore, if we take technology scaling and the subsequent increase of static power dissipation into account, the large footprint and large amount of idle logic will make CASE IIA[4] scale significantly worse than CASE III, where the amount of idle logic is kept small.

One might think that adding more 16-bit multipliers in parallel should give even better results, much in the same way as CASE IIA shows better results than CASE IIB. This is however not the fact. Adding more than two 16-bit multipliers mean that more routing would be needed, to supply the extra multiplier with its operands and for the result. This extra wiring would increase the area and energy, and it is likely it would have a negative impact also on the delay.

The execution time and total energy dissipation for the different cases do not include the overhead for switching between 32-bit and 16-bit multiplications. Therefore the multipliers in CASE IIA, IIB and III have been considered to operate on 32-bit or 16-bit multiplications for a long period of time. The values stated above are not valid if it is necessary to frequently switch between 32-bit and 16-bit multiplications.

1.7 Conclusions

The twin-precision technique allows for flexible architectural solutions, where the variation in operand bitwidth can be used to decrease power dissipation and to increase throughput of multiplications.

The Baugh-Wooley algorithm is particularly suitable for a twin-precision implementation. Due to the simplicity of this algorithm, 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.

[4]CASE IIB will also suffer from scaling but not as severely as for CASE IIA, because of the smaller footprint and amount of idle logic.

Currently a lot of research is done on reconfigurable architectures, where the architecture can be adapted to the applications to be executed. Some of these proposed architectures can adapt their arithmetic logic units to operate with different bitwidth, depending on the application. One such reconfigurable architecture is that of the FlexSoC project [27]. 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 makes it possible to efficiently deploy these flexible architectures.

This work has further shown that the modified-Booth multiplier is not the obvious choice for a high-speed implementation and it is especially unsuitable for a twin-precision implementation. Contrary to many beliefs, it has been shown that the popular modified-Booth multiplier is inferior to a Baugh-Wooley multiplier in terms of delay, power and area.

Bibliography

[1] David Brooks and Margaret Martonosi, "Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance," in *5th International Symposium on High Performance Computer Architecture*, January 1999, pp. 13–22.

[2] Zhijun Huang and Miloš D. Ercegovac, "Two-Dimensional Signal Gating for Low-Power Array Multiplier Design," in *IEEE International Symposium on Circuits and Systems*, May 2002, pp. 489–492.

[3] Kyungtae Han, Brian L. Evans, and Earl E. Swartzlander Jr., "Data Wordlength Reduction for Low-Power Signal Processing Software," in *IEEE Workshop on Signal Processing Systems*, 2004, pp. 343–348.

[4] Gabrial H. Loh, "Exploiting Data-Width Locality to Increase Superscalar Execution Bandwidth," in *35th International Symposium on Microarchitecture*, 2002.

[5] Alber Danysh and Dimitri Tan, "Architecture and Implementation of a Vector/SIMD Multiply-Accumulate Unit," *IEEE Transactions on Computers*, vol. 5, no. 4, pp. 284–293, May 2005.

[6] Pedram Mokrian, Majid Ahmadi, Graham Jullien, and W.C. Miller, "A Reconfigurable Digital Multiplier Architecture," in *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering*, 2003, pp. 125–128.

[7] Magnus Själander, Henrik Eriksson, and Per Larsson-Edefors, "An Efficient Twin-Precision Multiplier," in *IEEE Proceedings of the 22nd International Conference on Computer Design*, October 2004, pp. 30–33.

[8] Vojin G. Oklobdzija, David Villeger, and Simon S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," *IEEE Transactions on Computers*, vol. 45, no. 3, pp. 294–306, March 1996.

[9] Luigi Dadda, "Some Schemes for Parallel Adders," *Alta Frequenza*, vol. 34, no. 5, pp. 349–356, May 1965.

[10] Christopher S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Transactions on Electronic Computers*, vol. 13, pp. 14–17, February 1964.

[11] Henrik Eriksson, Per Larsson-Edefors, Mary Sheeran, Magnus Själander, Daniel Johansson, and Martin Schölin, "Multiplier Reduction Tree with Logarithmic Logic Depth and Regular Connectivity," in *IEEE International Symposium on Circuits and Systems*, May 2006.

[12] Charles R. Baugh and Bruce A. Wooley, "A Two's Complement Parallel Array Multiplication Algorithm," *IEEE Transactions on Computers*, vol. 22, pp. 1045–1047, December 1973.

[13] Mehid Hatamian, "A 70-MHz 8-bit x 8-bit Parallel Pipelined Multiplier in 2.5-μm CMOS," *IEEE Journal on Solid-State Circuits*, vol. 21, no. 4, pp. 505–513, August 1986.

[14] Andrew D. Booth, "A Signed Binary Multiplication Technique," *Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 1951.

[15] O.L.MacSorley, "High Speed Arithmetic in Binary Computers," in *Proceedings of the IRE*, January 1961, vol. 49, pp. 67–97.

[16] Jalil 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.

[17] Yeh Wen-Chang and Jen Chein-Wei, "High-Speed Booth Encoded Parallel Multiplier Design," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 692–701, 2000.

[18] Magnus Själander and Per Larsson-Edefors, "A Power-Efficient and Versatile Modified-Booth Multiplier," in *Proceedings of Swedish System-on-Chip Conference*. The Fifth Program Conference jointly organized by CCCD, FlexSoC, IRSED, RaMSiS, Socware and Stringent, April 2005.

[19] Magnus Själander, "HMS Multiplier Generator," `http://www.ce.chalmers.se/~hms/multiplier.html`, December 2005.

[20] Peter M. Kogge and Harold 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, August 1973.

[21] *Cadence NC-VHDL Simulator Help Version 5.1*.

[22] *Design Compiler User Guide Version W-2004.12*.

[23] *PrimeTime X-2005.06 Synopsys Online Documentation*.

[24] *PrimePower Manual Version W-2004.12*.

[25] Magnus Själander and Per Larsson-Edefors, "Comprehensive Evaluation of a Modified-Booth Multiplier with Logarithmic Reduction Tree," *IEEE Transaction on Very Large Scale Integrated Systems*, January 2006, Submitted manuscript.

[26] Magnus Själander, Mindaugas Draždžiulis, Per Larsson-Edefors, and Henrik Eriksson, "A Low-Leakage Twin-Precision Multiplier Using Reconfigurable Power Gating," in *International Symposium on Circuits and Systems*, May 2005, pp. 1654–1657.

[27] John Hughes, Kjell Jeppson, Per Larsson-Edefors, Mary Sheeran, Per Stenström, and Lars "J." Svensson, "FlexSoC: Combining Flexibility and Efficiency in SoC Designs," in *Proceedings of the IEEE NorChip Conference*, 2003.

Part II

PAPERS

PAPER I

M. Själander, H. Eriksson, and P. Larsson-Edefors
An Efficient Twin-Precision Multiplier
IEEE International Conference on Computer Design
San Jose, United States of America
October 10–13, 2004, pp. 507–510

2

PAPER I

We present a twin-precision multiplier that in normal operation mode efficiently performs N-b multiplications. For applications where the demand on precision is relaxed, the multiplier can perform $N/2$-b multiplications while expending only a fraction of the energy of a conventional N-b multiplier. For applications with high demands on throughput, the multiplier is capable of performing two independent $N/2$-b multiplications in parallel. A comparison between two signed 16-b multipliers, where both perform single 8-b multiplications, shows that the twin-precision multiplier has 72% lower power dissipation and 15% higher speed than the conventional one, while only requiring 8% more transistors.

2.1 Introduction

Recent development at the micro architecture level shows that there is an increasing interest in datapath components that are capable of performing computations with variable operand size, e.g. adders capable of doing both N and $N/2$-b additions [1]. By using only a part of the datapath component for computation, it has been demonstrated [2] that reductions in the total power dissipation can be effected. Datapath components that can perform both one N, one single $N/2$, or two $N/2$-b operations give the designer the opportunity to design a system which can adapt to changing modes, such as low-power, high-throughput, or high-precision operation. Such a datapath component could be used for dynamic power reduction in the same way as described by Abddollahi *et al.* [2]; by using the same kind of logic for detecting if the effective bit rate is within $N/2$-b precision, it is possible to control at what precision the datapath component should be operating. This versatile type of datapath component is also suitable for systems in which several applications, having quite different requirements on precision and/or throughput, are executed [3]. Furthermore, such a datapath component could prove useful in processors that can support several instruction sets. In a processor that combines x86-32 and x86-64, a flexible datapath could be used for 64-b operations as well as for Single Instruction Multiple Data (SIMD) instructions, where two 32-b operations are performed in parallel.

It has been shown [4] that it is relatively straightforward to partition an array multiplier, so as to obtain a multiplier that can perform multiplications with varying operand size[1]. In comparison to tree multipliers, however, an array multiplier is slow and power hungry which makes it a poor design choice when a fast and efficient multiplier is needed [5]. It was claimed, but not substantiated, that the power-reduction techniques used for array multipliers [2] can be applied also to tree multipliers. It is certainly not straightforward to transfer the proposed technique to tree multipliers. Mokrian *et al.* presented a reconfigurable multiplier, which is constituted by several smaller tree multipliers [6]. How-

[1] This was done by gating parts of the array of carry-save adders and by using multiplexers to read out the data from a low-precision multiplication.

ever, the recursive nature of this multiplier is, due to an addition of reduction stage(s), likely to have a large impact on the delay for the N-b multiplication, compared to the multiplier proposed in this paper.

In the following we explore the possibility of combining N and $N/2$-b multiplications in the same N-b tree multiplier: we call this a twin-precision multiplier. The key challenges in designing a twin-precision multiplier are to limit the impact of flexibility on power dissipation, delay, and area. The proposed twin-precision multiplier efficiently performs either one N-b multiplication, one single $N/2$-b multiplication, or two $N/2$-b multiplications in parallel.

2.2 Design Exploration

Based on a simple representation of an array multiplier, Figure 2.1, it is obvious that if the partial product bits not being used in a low-precision multiplication are set to zero, the array multiplier will produce the correct result without the need of any additional logic. The 2-input AND gates corresponding to the partial product bits that are not being used in the low-precision multiplication can be replaced by 3-input AND gates to force those bits to zero [2].

When doing an $N/2$-b multiplication within an N-b multiplier only one quarter of the logic is being used, as seen in grey in Figure 2.1. This makes it possible to use the multiplier for two parallel and independent $N/2$-b multiplications. We can partition the partial product bits of the N-b multiplier, such that an $N/2$-b multiplication can be performed in the Least Significant Part (LSP) of the multiplier in parallel with another $N/2$-b multiplication in the Most Significant Part (MSP), without using any additional logic in the partial product reduction tree, as seen in grey and black, respectively, in Figure 2.1. To be able to switch between N, $N/2$, or two $N/2$-b multiplications, the 2-input AND gates used to create the partial products need to be replaced with 3-input AND gates and two control signals for selecting the operating mode of the multiplier need to be introduced.

[2] When performing only one $N/2$-b multiplication it is possible to set the most significant bits of the operands to zero instead.

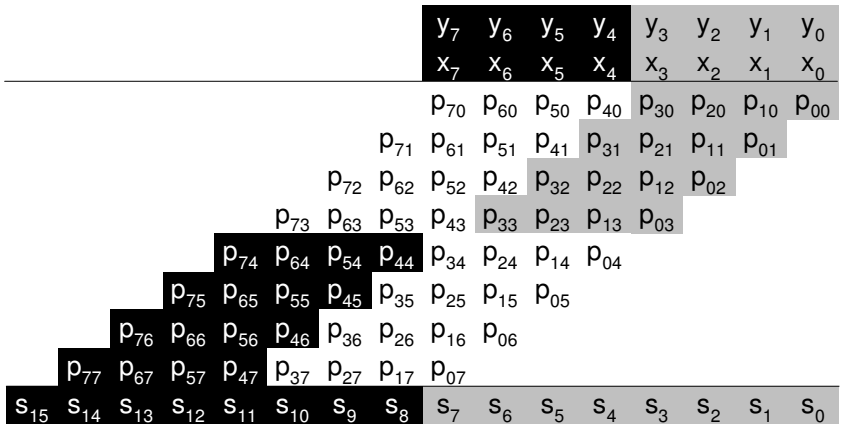


Figure 2.1: *Partial product representation of a 4-b multiplication in an 8-b multiplier.*

2.2.1 Tree Multiplier

Until now we have implicitly used the array multiplier to demonstrate the twin-precision feature. The array multiplier is, however, slow and power dissipating in comparison to a logarithmic tree multiplier. The implementation of the twin-precision feature in an N-b tree multiplier is similar to that of the array multiplier; all that is needed is to set the partial products bits not being used to zero and to partition the partial products bits of the two multiplications into the respective LSP and MSP of the tree. To reduce the critical path for the $N/2$-b multiplications the partial products bits used during the computation are moved as far down the tree as possible, Figure 2.2. In this paper we use a tree multiplier with regular connectivity [7].

To further reduce the critical path of the $N/2$-b multiplications it is possible to move the partial products even further down the tree by adding multiplexers on lower levels[3]. This makes it possible to select either the carry and sum from higher levels, when doing the N-b multiplication, *or* the partial products bits, when doing the $N/2$-b multiplication. This introduces multiplexers in the critical path of the N-b multiplier, which significantly increases the delay of the N-b

[3]Moving further down in the tree implies approaching the final adder.

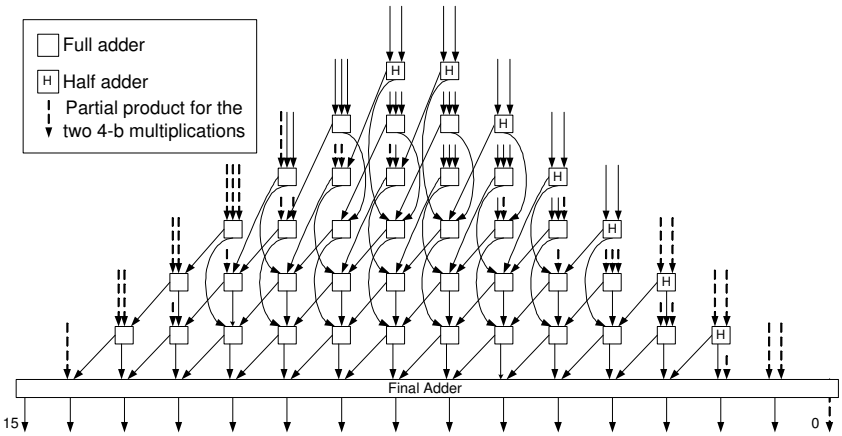


Figure 2.2: *Partitioned tree of an 8-b multiplier.*

multiplication. Thus, this alternative has not been considered here, since our goal is to find a good design tradeoff between the delay of N-b multiplications and $N/2$-b multiplications, respectively.

2.2.2 Signed Multiplication According to Baugh-Wooley

We used the Baugh-Wooley algorithm [8] to investigate the impact of the twin-precision feature on delay and power of a signed tree multiplier [4]. Here, signed multiplication is performed by first inverting all partial product bits that are results of the most significant bit (MSB) of exactly one of the operands, Figure 2.3. Second, for each executed multiplication, a logical one (framed) is added to column N (column 0 is to the far right in Figure 2.3) and, third, the MSB of the product is inverted. This is directly mapped onto the tree multiplier as shown in Figure 2.4.

To be able to generate the inverted partial product bits, we chose to replace the AND gates corresponding to the inverted bits with NAND gates followed by

[4]Modified Booth does not impose any fundamental problems to the twin-precision concept. It has been evaluated, but is not included in this paper because of space constraints.

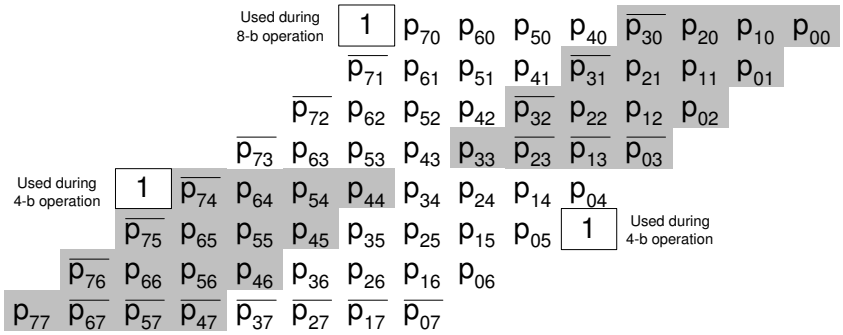


Figure 2.3: *Example showing the inverted partial product bits of two signed 4-b multiplications within a signed 8-b multiplication.*

XOR gates. The option to either invert or not invert the signal from the NAND gates makes it possible to switch between signed and unsigned multiplication

The inversion of the MSB of the product is also done with an XOR gate. The insertion of the logical one to column N of the multiplication is straightforward for the N-b and the $N/2$-b multiplication in the LSP by changing the half adder of that column to a full adder and adding the logical one to the new adder. For the $N/2$-b multiplication in the MSP there is no half adder that can be replaced, but an extra level of half adders has to be added, seen at the far left in Figure 2.4. This added level of half adders does not increase the delay for the N-b multiplication, since none of the half adders are in the critical path.

2.3 Final Adder

The choice of final adder is very important in order to get short delay for both N and $N/2$-b multiplications. The recommendations given by Oklobdzija *et al.* [9] are not directly applicable in our twin-precision multiplier, since the delay profile of the multiplier varies with the multiplication precision. It would be possible to use the adder scheme presented by Oklobdzija *et al.* to reduce the delay for the N-b multiplication, but this could introduce long delays for the

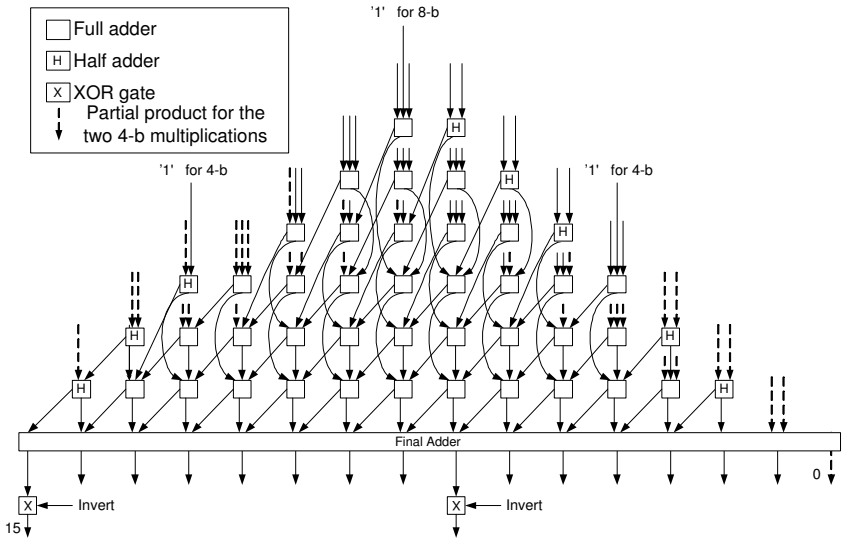


Figure 2.4: *Signed 8-b multiplier capable of doing two signed 4-b multiplications using Baugh-Wooley.*

two independent *N*/2-b multiplications. In order to not increase the delay too much for the *N*/2-b multiplications, it is therefore important to have a final adder that is fast for both *N* and *N*/2-b multiplications. Mathew *et al.* [1] presented a sparse-tree carry-lookahead adder that is capable of doing both fast 64-b and fast 32-b additions. This adder scheme has been adapted to the appropriate word length in order to obtain short delays for both *N* and *N*/2-b multiplications, Figure 2.5.

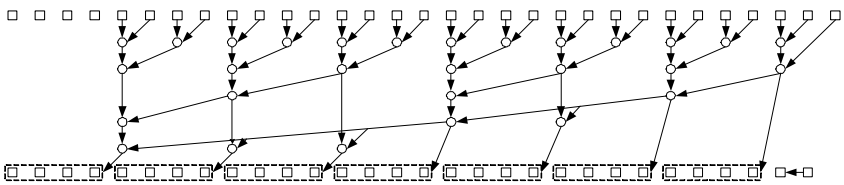


Figure 2.5: *Example of 31-b final adder for a 16-b twin-precision multiplier.*

2.4 Simulation Setup and Results

To evaluate delay and power dissipation, simulations have been performed in a commercially available 0.13-μm technology. The simulated circuit is a 16-b twin-precision tree multiplier, which is capable of performing two 8-b multiplications in parallel, and which uses the fast final adder of Section 3. As reference we use a conventional 8-b and 16-b multiplier, respectively, which both use a Kogge-Stone as final adder. For signed multiplication the Baugh-Wooley algorithm [8] has been implemented for both the conventional and the twin-precision multiplier. All simulations have been done using Spice transistor netlists including estimated wire capacitances. All logic has been implemented as static logic and designed to resemble what could be expected to be found in a standard-cell library. The implemented version of the multiplier cancels the inactive partial products by forcing the AND gates to zero. The impact of using sleep-mode techniques on power and delay has not been investigated[5]. For power simulation 50 random input vectors were applied to HSpice at 500 MHz, a supply voltage of 1.2 V, and an operating temperature of 25 °C. Delay was obtained using PathMill.

Table 2.1: *Reference values normalized to the conventional 8-b multiplier.*

Reference	Delay	Power
16 bit	1.40	4.06
8 bit	1.0	1.0

Table 2.1 lists the values for the conventional 8-b and 16-b multipliers used as comparison references. Table 2.2 lists delay and power for the twin-precision multiplier. All values have been normalized to the 8-b reference multiplier.

With a conventional 16-b multiplier as reference, the delay of the 16-b twin-precision multiplier operating in 16-b mode was 9.0% larger whereas the power dissipation was less than 1.0% larger. When using the 16-b twin-precision multiplier in single 8-b mode, the power dissipation is only 28% of the reference

[5]We expect no fundamental problems in introducing sleep-mode techniques in the twin-precision multiplier.

Table 2.2: *Simulation results for a twin-precision 16-b multiplier, where columns 2 and 3 are normalized to the conventional reference 8-b multiplier. Columns 4 to 7 are comparisons against the conventional reference multipliers given in Table 1.*

Mode	Delay	Power	Compared to 8-b		Compared to 16-b	
			Delay	Power	Delay	Power
16-b	1.52	4.10			9.0%	0.9%
2x8-b	1.29	2.34	29.0%	16.9%	-7.5%	-42.4%
8-b	1.18	1.13	18.2%	13.3%	-15.3%	-72.1%

16-b multiplier. The reason for the power reduction is that in single 8-b mode about two thirds of the multiplier tree is kept at constant zero, eliminating the dynamic power in these parts. The additional decrease in power comes from the reduction of glitches in the multiplier.

With a conventional 8-b multiplier as reference, the delay of the 16-b twin-precision multiplier operating in single 8-b mode was 18.2% larger whereas the power dissipation was 13.3% larger.

When doing two 8-b multiplications in parallel the power dissipation increases by about 4% and the delay is increased with 10% compared to a single 8-b multiplication. The increase in the delay is due to an increased logic depth—the 8-b multiplication in the MSP of the multiplier tree has a longer critical path than the 8-b multiplication in the LSP has, Figure 2.4. Additionally the logical depth of the final adder is one gate deeper for the MSP which contributes to a longer critical path for the 8-b multiplication computed in the MSP of the tree.

The power dissipation for driving the control signals to set the mode of the multiplier was not included in the power simulation. The control signal used to set the partial product bits to zero, when doing two $N/2$-b multiplications, is connected to the input of N AND gates. In order to cancel out the second $N/2$-b multiplication the control signal is connected to the input of $N/2$ AND gates. It has been shown that it is realistic to expect the multiplier to operate in the same mode for longer durations [2]. Since the control signals only toggle when

the mode of the multiplier is changed, the power dissipation for these signals is negligible when the multiplier stays in one mode for longer durations.

2.5 Conclusion

The twin-precision multiplier presented in this paper offers a good tradeoff between precision flexibility, area, delay and power dissipation by using the same multiplier for doing N, $N/2$ or two $N/2$-b multiplications. In comparison to a conventional 16-b multiplier, a 16-b twin-precision multiplier has 8% higher transistor count and 9% longer delay. The relative transistor count overhead decreases for larger multipliers, since the number of AND gates needed to set the partial products to zero does not grow as fast as the number of adders in the tree.

Bibliography

[1] Sanu Mathew, Mark Anders, Brad Bloechel, Tran Nguyen, Ram Krishnamurthy, and Shekhar Borkar, "A 4GHz 300mW 64b Integer Execution ALU with Dual Supply Voltages in 90nm CMOS," in *Proceedings of the International Solid State Circuits Conference*, 2004, pp. 162–163.

[2] Afshin Abddollahi, Massoud Pedram, Farzan Fallah, and Indradeep Ghosh, "Precomputation-Based Guarding for Dynamic and Leakage Power Reduction," in *IEEE Proceedings of the 21st International Conference on Computer Design*, 2003, pp. 90–97.

[3] John Hughes, Kjell Jeppson, Per Larsson-Edefors, Mary Sheeran, Per Stenström, and Lars "J." Svensson, "FlexSoC: Combining Flexibility and Efficiency in SoC Designs," in *Proceedings of the IEEE NorChip Conference*, 2003.

[4] Zhijun Huang and Miloš D. Ercegovac, "Two-Dimensional Signal Gating for Low-Power Array Multiplier Design," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, 2002, pp. I–489–I–492 vol.1.

[5] Thomas K. Callaway and Earl E. Swartzlander, Jr., "Optimizing Multipliers for WSI," in *Proceedings of the Fifth Annual IEEE International Conference on Wafer Scale Integration*, 1993, pp. 85–94.

[6] Pedram Mokrian, Majid Ahmadi, Graham Jullien, and W.C. Miller, "A Reconfigurable Digital Multiplier Architecture," in *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering*, 2003, pp. 125–128.

[7] Henrik Eriksson, *Efficient Implementation and Analysis of CMOS Arithmetic Circuits*, Ph.D. thesis, Chalmers University of Technology, 2003.

[8] Charles R. Baugh and Bruce A. Wooley, "A Two's Complement Parallel Array Multiplication Algorithm," *IEEE Transactions on Computers*, vol. 22, pp. 1045–1047, December 1973.

[9] Vojin G. Oklobdzija, David Villeger, and Simon S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," *IEEE Transactions on Computers*, vol. 45, no. 3, pp. 294–306, March 1996.

PAPER II

M. Själander, M. Draždžiulis, P. Larsson-Edefors, and H. Eriksson
A Low-Leakage Twin-Precision Multiplier Using Reconfigurable Power Gating
IEEE International Symposium on Circuits and Systems
Kobe, Japan, May 23–26, 2005, pp. 1654–1657

3
PAPER II

A twin-precision multiplier that uses reconfigurable power gating is presented. Employing power cut-off techniques in independently controlled power-gating regions, yields significant static leakage reductions when half-precision multiplications are carried out. In comparison to a conventional 8-bit tree multiplier, the power overhead of a 16-bit twin-precision multiplier operating at 8-bit precision has been reduced by 53% when reconfigurable power gating based on the SCCMOS power cut-off technique was applied.

3.1 Introduction

Recent development of embedded systems indicates an increased interest in reconfigurable functional units that dynamically can adapt the datapath to varying

computational needs. A system may need to switch between, for example, one application that needs speech-encoding functional units operating at 8-bit precision and another application that needs 16-bit functional units to perform audio decoding. Since most embedded systems are associated with a strictly limited power budget, both static and dynamic power are of critical importance. In this context, it is hardly acceptable to have idle functional units which dissipate useless static power while the application is running at low precision.

The twin-precision (TP) multiplier [1] can switch between N-bit and $N/2$-bit precision multiplications without significant performance and area overhead. However, in half-precision ($N/2$-bit) mode the TP multiplier is dissipating considerably more power than a conventional fixed-precision $N/2$-bit multiplier, since idle, higher-precision logic gates within the TP multiplier are leaking. In order to efficiently utilize reconfigurable datapath units, leakage reduction techniques need to be incorporated.

In this paper we describe a twin-precision multiplier that uses a power-gating strategy that dynamically adapt to the precision needed and shuts down idle portions of the circuit to save static leakage power.

3.2 Preliminaries

3.2.1 The Twin-Precision Multiplier

The twin-precision multiplier [1] is an N-bit tree multiplier that is capable of performing either *i)* single N-bit, *ii)* single $N/2$-bit or *iii)* two concurrent and independent $N/2$-bit multiplications. When compared to a conventional fixed-precision N-bit tree multiplier, the general gate count overhead of the twin-precision multiplier was very small, and the overhead in delay and active power dissipation for the full-precision mode was shown to be small [1]. The most important features of the twin-precision technique are illustrated in Fig. 3.1.

In order to switch between the three different precision modes it is necessary that the partial products in white and black regions (Fig. 3.1) can be forced to zero independently of each other. The 2-input AND gates, which in con-

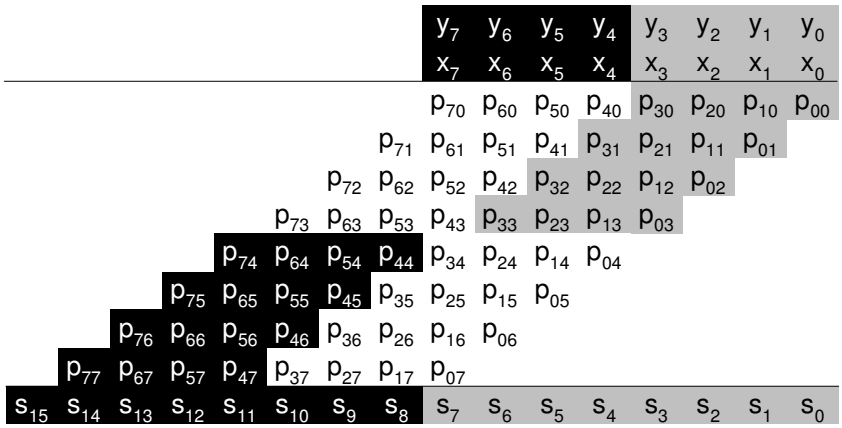


Figure 3.1: *Partial product representation of an 8-bit multiplication. When performing a 4-bit multiplication in a conventional multiplier only one quarter of the logic gates are performing any useful operations (the grey regions). The twin-precision technique rearranges logic to greatly reduce power dissipation and delay for 4-bit multiplications. Also, this technique allows the logic gates, which are not used in the 4-bit multiplication taking place in the grey region, to perform a second, independent 4-bit multiplication (the black regions).*

ventional multipliers generate the partial product bits, are replaced with 3-input AND gates, whose extra input is used to mask the output to zero.

In a twin-precision multiplier it is crucial that the partial products used for $N/2$-bit multiplications are moved as close to the final adder as possible. In half-precision mode, this gives the shortest critical path and the largest number of unused logic gates, which can be translated into dramatic dynamic power reductions. The final adder is also important, since its delay profile will determine the delay for the N-bit and the two $N/2$-bit multiplications. In conventional adder design, the delay is optimized for full-precision operation. This generally makes such adders relatively slow when used for lower precision. A good tradeoff between the delay for N-bit and $N/2$-bit precision is given by the adder presented by Mathew *et al.* [2].

The application of the twin-precision technique to signed N-bit and $N/2$-bit multiplications is quite straightforward. An example of an 8-bit signed multiplier using the Baugh-Wooley algorithm [3], capable of performing two concurrent 4-bit multiplications, is shown in Fig. 3.2. The twin-precision technique can also be applied to modified Booth multipliers, but this requires a slightly extended implementation effort.

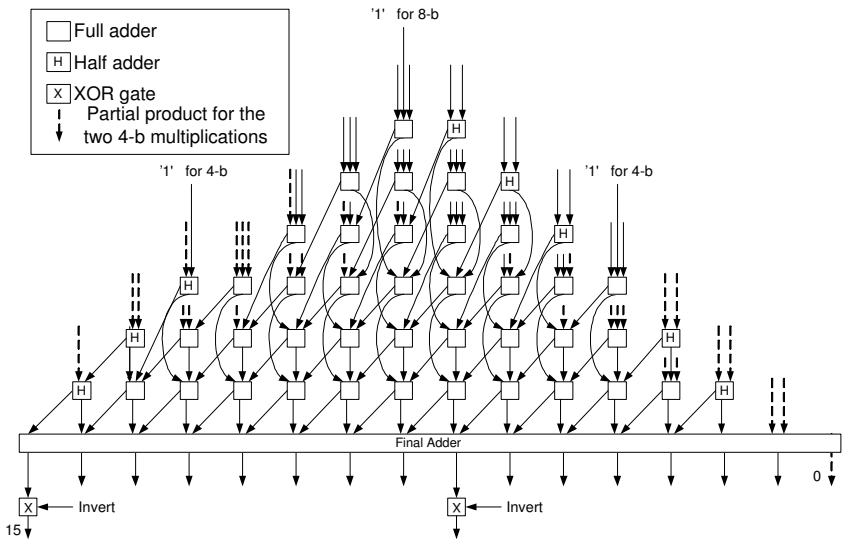


Figure 3.2: *Signed 8-bit twin-precision Baugh-Wooley multiplier.*

3.2.2 Circuits for Leakage Reduction

As soon as unused circuit parts of a twin-precision (TP) multiplier stop switching or when the entire multiplier goes into sleep mode, static leakage currents become visible. Transistor stacking, body biasing, multi-V_t technologies and power cut-off techniques are common methods used to suppress static leakage. We have chosen to study only power cut-off techniques in this paper, because these are compatible with conventional CMOS design and require no special process technologies. Power cut-off techniques are using power switches

(PMOS or NMOS transistors) that are turned *on* when logic circuits are active and *off* when circuits are in sleep mode. Generally, a power cut-off technique is efficient for circuits that stay in sleep mode for relatively long periods of time.

In this paper we have considered a number of power cut-off techniques [4] that can be applied to a TP multiplier: Super Cut-Off CMOS (SCCMOS), Zigzag Super Cut-Off CMOS (ZSCMOS) and Gate leakage Suppression CMOS (GSCMOS). We found that SCCMOS has long sleep-to-active (wake-up) time, while ZSCCMOS has short wake-up time at the expense of a need for sleep-mode state control (input forcing). The GSCMOS technique, on the other hand, is particularly efficient when gate leakage is significant, but it has complex supply rail routing. All considered power cut-off techniques need on-chip voltage generators to control the power switches. Although power cut-off techniques corrupt logic data during sleep, this is not a serious issue for the TP multiplier, since a fresh multiplication is performed every time the precision mode has been changed.

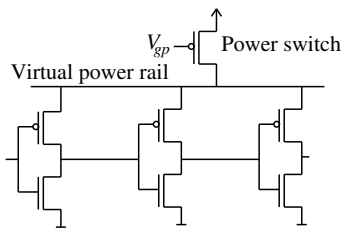


Figure 3.3: *The SCCMOS technique applied to an inverter chain.*

We chose the SCCMOS technique (Fig. 3.3), since this fits our needs and constraints (e.g. low supply routing complexity, a wake-up time within a few clock cycles, and absence of gate leakage) in the 0.13-μm technology used. The power switch used here is a low-V_t PMOS transistor, which in sleep mode is overdriven (i.e. above V_{dd}) with ΔV, which roughly is the threshold voltage difference between a typical high-V_t and a typical low-V_t device. The overdrive voltage completely turns the power switch *off*, thus the voltage on the virtual power rail drops and leakage currents are reduced.

3.3 Power Supply Grid and Tree Organization

To suppress static leakage in idle gates of the twin-precision (TP) multiplier, we deploy the SCCMOS technique so that three separate power-gating regions (Fig. 3.4) are obtained. Independent of other regions, the power switches of one region can be turned *on* or *off*, depending on multiplier precision mode.

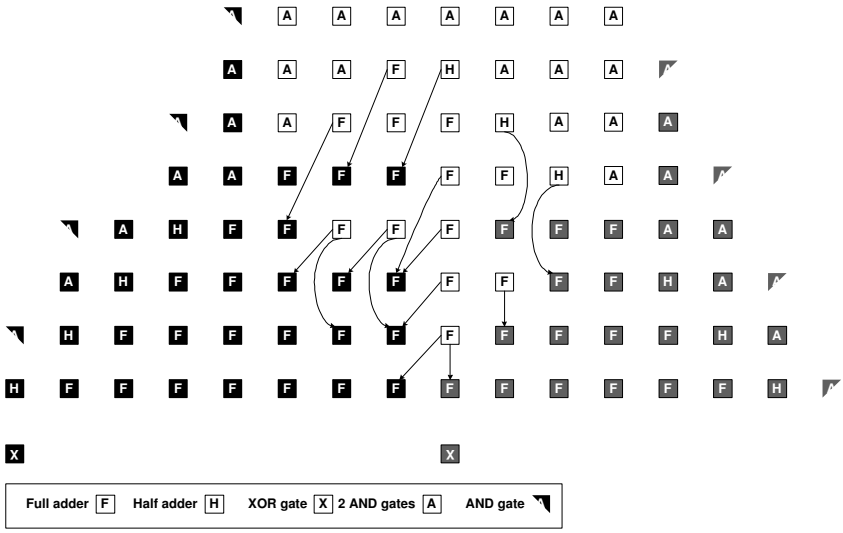


Figure 3.4: *Power-gating regions of an 8-bit TP multiplier. When one* N*/2-bit multiplication is performed, the power switches in black and white regions are turned* off. *When two concurrent* N*/2-bit multiplications are performed, the power switches in the white region are turned* off. *Finally, when an* N*-bit multiplication is performed, the power switches in all regions are turned* on.

When the TP multiplier performs an $N/2$-bit multiplication or two concurrent $N/2$-bit multiplications, outputs of some sleeping logic gates are connected to the inputs of active logic gates (Fig. 3.4). Since those outputs now are floating, the inputs of the active logic gates can not be asserted via the 3-input AND gates that were used in the original TP multiplier. Instead, the inputs of these active logic gates are pulled down by NMOS transistors, which are connected

to the respective outputs of the sleeping gates. When the TP multiplier performs an $N/2$-bit multiplication or two concurrent $N/2$-bit multiplications, the pull-down NMOS transistors are turned *on*, thus forcing the outputs of sleeping logic gates (i.e. the inputs of the active logic gates) to ground. The *off*-state power switches eliminate short-circuit currents in sleeping logic gates.

The virtual power rails that are introduced with the SCCMOS technique need special consideration. Within each power-gating region, the virtual power rails can be partitioned into networks of finer granularity than a single network spanning the entire region. In fact, every logic gate could have its own virtual power rail and power switch, leading to all logic gates having individual virtual supply voltages (which are functions of the gate input pattern) and guaranteeing overall optimal leakage suppression in sleep mode. However, this calls for an extra power-switch control wire that needs to be routed through *every* cell in a layout, in turn causing cell footprints and wire lengths to grow. Longer signal wires lead to larger switched capacitance and, thus, both the dynamic power dissipation and the delay of the active TP multiplier regions will increase. As for the other extreme, i.e. leaving the virtual power rails within each power-gating region non-partitioned, this is generally a recipe for larger than minimal leakage currents. This is because all logic gates of a large power-gating region will have one common voltage level on the virtual power rail. In sleep mode this common level rarely leads to minimal leakage currents in individual logic gates.

For the SCCMOS implementation of this paper (Fig. 3.5) we employ one virtual power rail and one power switch to groups of four full-adder (FA) cells (or for any group of logic cells that have the same total footprint). This partitioning yields a regular power supply grid that is easy to implement in layout. In our implementation, we assume constant cell pitch (cell height) and, as shown in Fig. 3.5, two AND gates are concatenated into one cell to make cell footprints uniform (two AND gates are now as wide as one FA cell).

In active mode, the power switch resistance will cause supply voltages on virtual power rails to drop below external supply voltage levels and, hence, the circuit delay will increase. The worst (largest) supply voltage drop takes

place when the gates on a particular virtual power rail receive the input signals that make the maximum number of PMOS transistors switch simultaneously. We choose to size the power switches so that during switching, for the worst-case conditions described above, the voltage drop on all virtual power rails is equal. Therefore, all virtual power rails have identical width ratios (K), where K is defined for each virtual power supply network as the power switch width divided by the total width of the PMOS transistors that simultaneously switch for the worst-case condition.

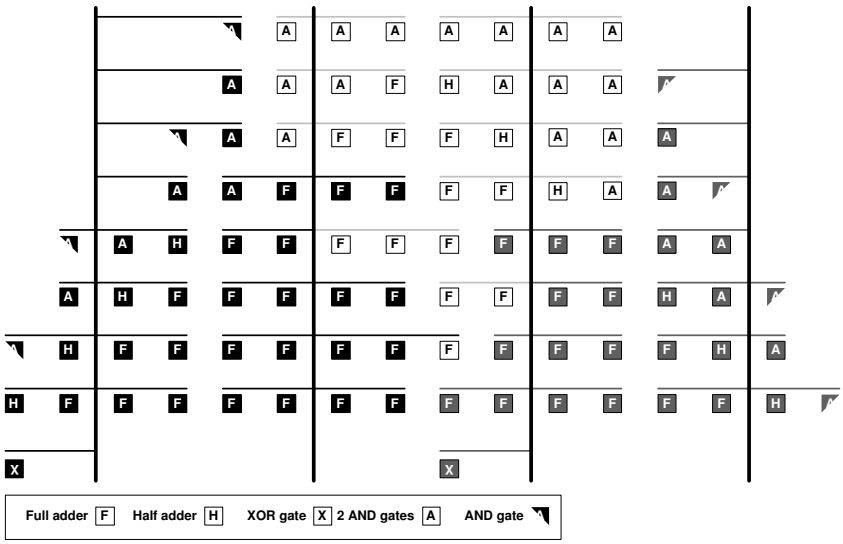


Figure 3.5: *The power supply grid of the 8-bit TP multiplier. Here, virtual power rails are routed horizontally, whereas external power and ground rails are routed vertically.*

Regarding the layout, power switches are inserted where an external power rail crosses a virtual power rail (Fig. 3.5). If an external power rail is routed between two different power-gating regions then *two* power switches (for two different virtual power rails) are inserted at each crossing. Since the N-wells of the PMOS transistors inside the logic gates are connected to the virtual power rails, the external power rails do not have to be routed horizontally. Hence, in the proposed power supply grid and tree organization the area penalty is

limited to the vertically routed external supply rails. Since these rails increase signal wire lengths between cells, we also have a delay and power penalty when circuits are active.

Logic gates in the white region, which are "trapped" between the grey and black regions and which do not have external supply rail routed through them, are connected to the black-region virtual power rails; the "white" FA-cell in the third row from the bottom in Fig. 3.5 is an example of a "trapped" logic gate. This connection simplifies the routing but incurs a small power penalty when two concurrent $N/2$-bit multiplications are performed, since "trapped" gates will be connected to the external power rails via the *on*-state black-region power switches.

Fig. 3.5 has a final adder (in the second row from the bottom) which for the sake of simplicity in this example is a ripple-carry adder. This adder can be replaced by any kind of tree adder, which subsequently can be partitioned in the same way as a multiplier reduction tree.

3.4 Simulation and Results

We will now evaluate power dissipation and delay of a 16-bit twin-precision multiplier *with power-gating regions employing the SCCMOS technique*. As the first reference design we use a conventional twin-precision multiplier, which *does not* use power gating. In the rest of this section, *TP* refers to the conventional twin-precision multiplier in [1], whereas *TP-cutoff* refers to the proposed power-gated twin-precision multiplier. In order to strike a good balance between the delay of 8-bit and 16-bit multiplications, both the TP and TP-cutoff multipliers use the adder presented by Mathew *et al.* [2] as final adder.

As a second set of references we use conventional 8-bit and 16-bit tree multipliers. These references represent fixed-precision multipliers, which have final adders that are optimized for either 8-bit or 16-bit multiplications. Here, we used the Kogge-Stone [5] structures for final adders. Both conventional tree multipliers employ the SCCMOS technique with a power supply network partitioning which is similar to that of the TP-cutoff multiplier.

Simulation data were obtained by running HSpice on a commercially available 0.13-μm technology, at a supply voltage of 1.2 V and an operating temperature of 80°C. Circuit netlists were constructed using static CMOS gates and estimated wire capacitances (pre-layout). Power switches were overdriven with 0.2 V. Power figures were obtained by running simulations for 50 random input vectors applied at a 500-MHz frequency. Power dissipation from control signal transitions and overdrive voltage generators was not included.

We could not use PathMill [6] directly to obtain reliable delay figures for the power-gated multipliers. Instead, delay figures for all multipliers had to be obtained by the following steps: First, using PathMill, we found the critical path for a multiplier that did not use the SCCMOS technique. Then we constructed netlists of *only the critical path* for *i)* a multiplier without the SCCMOS technique and *ii)* a multiplier using the SCCMOS technique. These netlists combine, first, the structure of the critical path originally obtained by PathMill and, second, the logic gates used in the HSpice netlists of the respective complete multipliers. Finally, the critical paths were simulated using HSpice. Inputs to the critical paths were asserted such that signal rippling was guaranteed. To define accurate fan-outs, the critical-path netlists also contained all logic gates and respective wire capacitances that the rippling signals were driving. To realistically model capacitance on the virtual power rails, all logic gates connected to the power switches were also included.

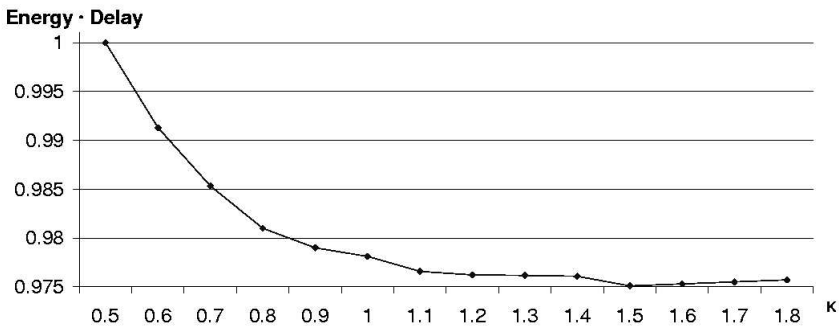


Figure 3.6: *Normalized energy-delay product for different power switch sizes.*

Fig. 3.6 shows the energy-delay product in a 16-bit TP-cutoff multiplier (operating at 8-bit precision) as a function of power-switch size. When K=1.0, for each virtual supply network the power switch width is equal to the total width of the PMOS transistors that simultaneously switch for the worst-case condition. From Fig. 3.6 it can be observed that K=1.1 gives a good compromise between expended energy and delay, so K=1.1 is used henceforth. Also, with K=1.1 the wake-up times for the power-gated multipliers were within one clock cycle.

Table 3.1: *Total power dissipation [*mW*] for 8-bit multiplications.*

Mult.	Mode	Cutoff
8-bit	8-bit	0.9667
16-bit	8-bit	3.473

(a) Conventional

Mult.	Mode	TP	TP-cutoff
16-bit	8-bit	1.216	1.083

(b) Twin-precision

The power dissipation of the 16-bit TP-cutoff multiplier operating at 8-bit precision can be compared to the power of an active conventional 8-bit multiplier, which incorporates power cut-off circuit techniques. In fact, the latter multiplier represents the lower power dissipation bound for any 16-bit multiplier that operates in 8-bit precision mode. As Table 3.1 shows, the 16-bit TP-cutoff multiplier only dissipates 12% more power than a conventional 8-bit multiplier. In comparison to a conventional 16-bit multiplier, which operates on 8-bit precision operands that are sign extended to 16 bits, the TP-cutoff multiplier has 3.2 times lower power dissipation. We finally observe that by using precision-dependent power-gating based on the SCCMOS technique, the power overhead (with reference to an active conventional 8-bit multiplier) of the 16-bit TP multiplier operating at 8-bit precision is reduced by as much as 53%.

As shown in Table 3.2, in 16-bit precision mode the TP multiplier is less than 3% faster than the TP-cutoff multiplier. The conventional 16-bit tree multiplier that uses the SCCMOS technique is 7% faster than the TP-cutoff multiplier operating at 16-bit precision, a ratio that is largely due to the use of different final adders—Kogge-Stone in the former and the adder by Mathew *et al.* in the latter.

Table 3.2: *Delays [*ns*] for 8-bit and 16-bit multiplications.*

Mult.	Mode	Cutoff
8-bit	8-bit	1.190
16-bit	16-bit	1.687

(a) Conventional

Mult.	Mode	TP	TP-cutoff
16-bit	8-bit	1.400	1.402
16-bit	16-bit	1.759	1.805

(b) Twin-precision

The difference in delay between the TP multiplier and the TP-cutoff multiplier operating in 8-bit precision mode is hardly noticeable. However, not surprisingly, the conventional 8-bit tree multiplier using the SCCMOS technique outperforms both twin-precision multipliers with 18%. This is due to *i)* use of a fixed 8-bit final adder and *i)* lower logic depth in the 8-bit reduction tree.

3.5 Conclusions

We have shown that power cut-off techniques can be deployed in different regions of a twin-precision functional unit, so that static leakage reduction can be effected not only when the entire unit is idle, but also when only parts of the unit are active, i.e. when the unit operates in half-precision mode.

Bibliography

[1] Magnus Själander, Henrik Eriksson, and Per Larsson-Edefors, "An Efficient Twin-Precision Multiplier," in *IEEE Proceedings of the 22nd International Conference on Computer Design*, October 2004, pp. 30–33.

[2] Sanu Mathew, Mark Anders, Brad Bloechel, Tran Nguyen, Ram Krishnamurthy, and Shekhar Borkar, "A 4GHz 300mW 64b Integer Execution ALU with Dual Supply Voltages in 90nm CMOS," in *Proceedings of the International Solid State Circuits Conference*, 2004, pp. 162–163.

[3] Charles R. Baugh and Bruce A. Wooley, "A Two's Complement Parallel Array Multiplication Algorithm," *IEEE Transactions on Computers*, vol. 22, pp. 1045–1047, December 1973.

[4] Mindaugas Draždžiulis, Per Larsson-Edefors, Daniel Eckerbert, and Henrik Eriksson, "A Power Cut-Off Technique for Gate Leakage Suppression," in *European Solid-State Circuits Conference*, September 2004, pp. 171–4.

[5] Peter M. Kogge and Harold 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, August 1973.

[6] *PathMill user guide, Version U-2003.03-SP1*.

PAPER III

M. Själander and P. Larsson-Edefors
A Power-Efficient and Versatile Modified-Booth Multiplier
Swedish System-on-Chip Conference
Tammsvik, Sweden April 18–19 2005

4

PAPER III

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.

4.1 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 4.2 reviews the twin-precision concept as well as the modified-Booth algorithm. Section 4.3 describes the use of the twin-precision concept for modified-Booth multipliers and Section 4.4 details the corresponding implementation. Results for the modified-Booth twin-precision multiplier are given in Section 4.5. Finally, Section 4.6 concludes the paper.

4.2 Preliminaries

4.2.1 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. 4.1, 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. 4.1).

$$N_{Full} \geq N_{Small1} + N_{Small2} \tag{4.1}$$

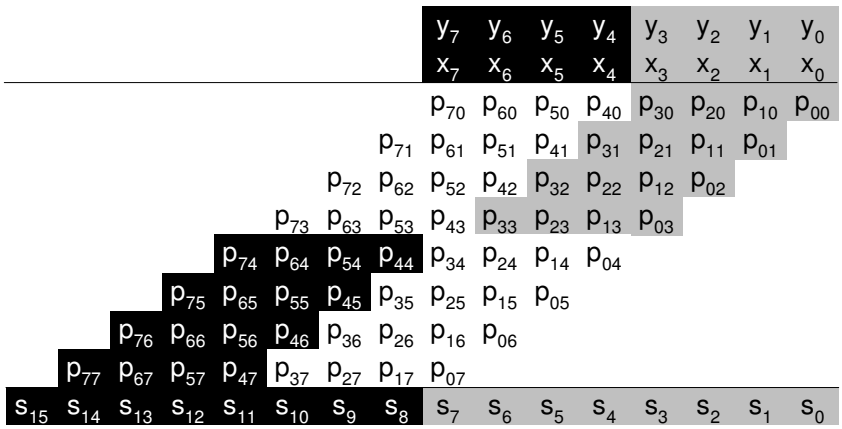


Figure 4.1: *Partial-product representation of an 8-bit multiplication with two 4-bit multiplications.*

$$p_{ij} = y_i x_j \tag{4.2}$$

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.

4.2.2 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. 4.3-4.5), which in turn are used to decode the partial products for a single row using the second operand (Eq. 4.6).

$$X1 = \overline{x_{2j-1} \oplus x_{2j}} \tag{4.3}$$

$$Z = \overline{x_{2j+1} \oplus x_{2j}} \tag{4.4}$$

$$X2 = x_{2j-1} \oplus x_{2j} \tag{4.5}$$

$$p_{ij} = \overline{(\overline{y_i \oplus x_{2j+1}} + X1)(\overline{y_{i-1} \oplus x_{2j+1}} + Z + X2)} \tag{4.6}$$

Fig. 4.2 shows which parts of the operand that are encoded and used to decode a specific row of partial products.

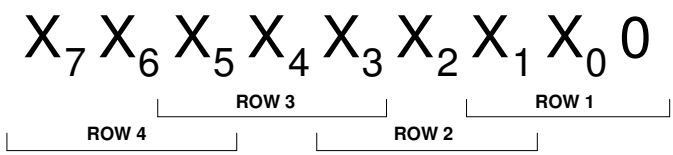


Figure 4.2: *8-bit encoding.*

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. 4.7) is treated in a special way, and *ii)* an a_i (Eq. 4.8) is added for each row of partial products.

$$p_{LSBi} = y_0(x_{2i-1} \oplus x_{2i}) \tag{4.7}$$

$$a_i = x_{2i+1}(\overline{x_{2i-1} + x_{2i}} + \overline{y_0 + x_{2i}} + \overline{y_0 + x_{2i-1}}) \tag{4.8}$$

The final result of the modified-Booth algorithm is shown in Fig. 4.3.

								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

Figure 4.3: *8-bit modified-Booth multiplication.*

4.3 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 4.2.1). 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.

By comparing the two multiplications in Fig. 4.3 and Fig. 4.4 (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. 4.3) need to define partial products that are used

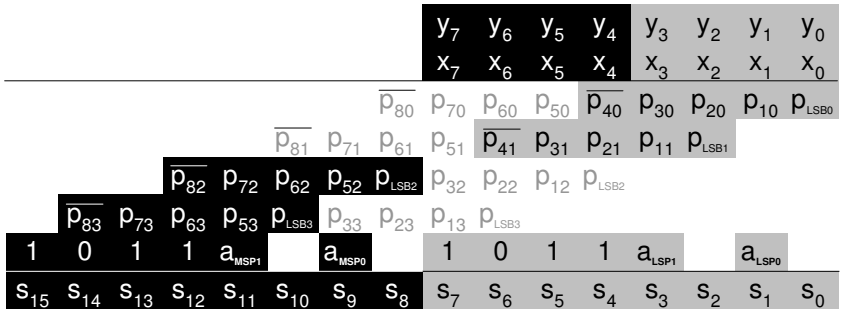


Figure 4.4: *Two 4-bit modified-Booth multiplications.*

to prevent sign extension in the low-precision 4-bit multiplication in the LSP (Fig. 4.4).

- 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.

4.4 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. 4.3 and Fig. 4.4, 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 $3N/2$) 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.

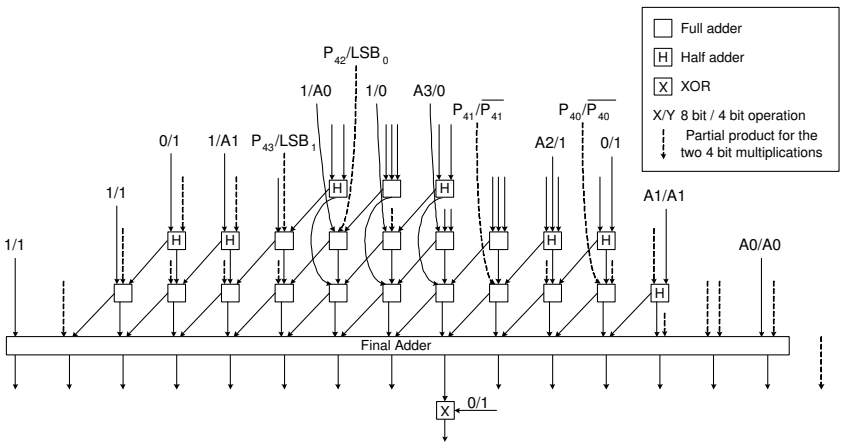


Figure 4.5: *8-bit twin-precision modified-Booth multiplier.*

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. 4.5.

With four extra transistors the decoder can set a partial product to zero (the extra transistors are encircled in Fig. 4.6); depending on the mode of operation, the *Zero* input controls whether a partial product is set to zero.

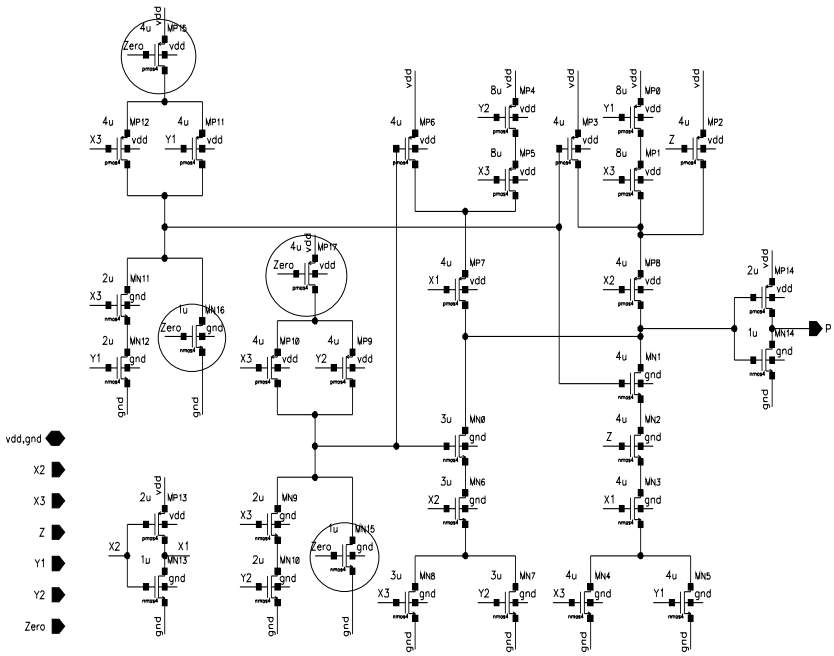


Figure 4.6: *Decoder circuit that optionally sets partial products to zero.*

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. 4.7, which can be compared to the encoding scheme for the 8-bit multiplication in Fig. 4.2.

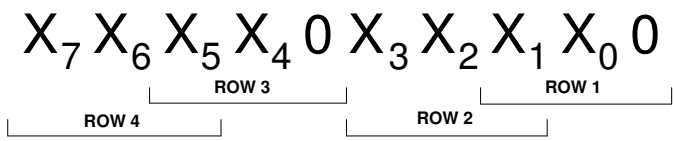


Figure 4.7: *Encoding scheme for two 4-bit multiplications.*

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 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.

4.5 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 4.1 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.

Table 4.1: *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

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. 4.9-4.16) of the multiplications has to be run with lower precision to achieve total power reduction.

$$x = \text{Number of } 8-\text{bit multiplications} \tag{4.9}$$

$$1 = \text{Total number of multiplications} \tag{4.10}$$

$$P_{conv.} = 1 * 24.8\,\text{mW} \tag{4.11}$$

$$P_{twin} = (1-x) * 25.6 + x/2 * 61.3\,\text{mW} \tag{4.12}$$

$$P_{conv.} = P_{twin} \tag{4.13}$$

$$1 * 24.8 = (1-x) * 25.6 + x/2 * 61.3 \tag{4.14}$$

$$18x = 0.8 \tag{4.15}$$

$$x \approx 4.4\% \tag{4.16}$$

Table 4.2 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

Table 4.2: *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

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.

4.6 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.

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.

Bibliography

[1] Magnus Själander, Henrik Eriksson, and Per Larsson-Edefors, "An Efficient Twin-Precision Multiplier," in *IEEE Proceedings of the 22nd International Conference on Computer Design*, October 2004, pp. 30–33.

[2] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic, *Digital Integrated Circuits, A design Perspective*, chapter 11, pp. 587–589, Number 0-13-597444-5. Prentice Hall Electronics and VLSI series, 2003.

[3] Yeh Wen-Chang and Jen Chein-Wei, "High-Speed Booth Encoded Parallel Multiplier Design," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 692–701, 2000.

[4] Jalil 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] Peter M. Kogge and Harold 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, August 1973.

[6] *Spectre user guide, Version 5.0.33.092203 – 23 Sep 2003*.

[7] Afshin Abddollahi, Massoud Pedram, Farzan Fallah, and Indradeep Ghosh, "Precomputation-Based Guarding for Dynamic and Leakage Power Reduction," in *IEEE Proceedings of the 21st International Conference on Computer Design*, 2003, pp. 90–97.

[8] *PathMill user guide, Version U-2003.03-SP1*.

PAPER IV

M. Själander and P. Larsson-Edefors

Comprehensive Evaluation of a Modified-Booth Multiplier with Logarithmic Reduction Tree

IEEE Transaction on Very Large Scale Integrated Systems

Submitted manuscript, January 20th 2006

5

PAPER IV

In studying signed multiplication schemes for operand widths from 8 to 64 bits, we conclude that a state-of-the-art modified-Booth multiplier consistently is inferior to the less complex Baugh-Wooley multiplier. This investigation is based on a regularly interconnected reduction tree that made it possible to efficiently instantiate the multipliers and to create a publicly available VHDL multiplier generator. The results from synthesis in a 0.13-μm process show that a modified-Booth multiplier on the average is 8% slower, 51% more power dissipating and 37% larger than a Baugh-Wooley multiplier.

5.1 Introduction

Thanks to the versatility of the modified-Booth (MB) algorithm [1] it has been a widely used multiplication scheme for a long time. In the MB algorithm, one of the operands is encoded into the higher-radix representation of $\{-2, -1, 0, 1, 2\}$, before being multiplied with the second operand. This method decreases the number of generated partial products to half of that of a straightforward partial-product generation scheme directly based on 2-input AND gates. When the MB algorithm is implemented sequentially, based on additions, subtractions and shifts, or when implemented based on an array of adders, it offers a clear benefit in that it has a direct effect on the total time required for the multiplication.

In 1993, Villeger *et al.* [2] presented a gate-level analysis, which showed that a multiplier built out of either 4:2 compressors or a Wallace tree has shorter logic depth than a MB multiplier, and that the 4:2-compressor implementation is preferred because of its regularity. In 2000, Yeh *et al.* [3] presented a new MB recoding scheme that reduced the logic depth of a MB multiplier, with the consequence that MB got a shorter logic depth than both a 4:2-compressor and a Wallace implementation. Although Yeh *et al.* did take their MB multiplier through synthesis, the comparison against other multiplier schemes was solely conducted at gate level, considering only unit delays. However, MB recoding intrinsically gives rise to several high fanout signals, which have a significant impact on performance. Since fanout is a property which is not at all considered in the gate-level analyses above, those studies are providing incomplete data for comparing MB multipliers against other multiplier schemes.

When a fast logarithmic reduction tree is selected, it is likely that the MB recoding part becomes a speed bottleneck. For the sake of a new comparison, it appears as if a signed multiplication scheme with lower overhead is sought for. Furthermore, we know from previous MB multiplier designs that the circuits for the reduction tree and for the MB recoding part are very different and, thus, they are hard to integrate. In view of technology scaling and related interconnect aspects, we know that efficient integration of as many parts as possible is essential

to performance. A signed multiplication algorithm which has significantly less overhead than MB recoding and which can be integrated with a reduction tree may provide a solution.

In this article we evaluate top speed, power dissipation and footprint of a state-of-the-art modified-Booth multiplier implementation. For our comparison we use a multiplier based on the Baugh-Wooley [4] algorithm, since this allows us to implement a signed representation using relatively few and simple gates without running into fanout problems. A VHDL multiplier generator that utilizes a parameterized reduction tree makes it possible to synthesize and efficiently benchmark multiplier implementations that have a wide range of operand widths. In our investigations, we use operand widths from 8 to 64 bits, with four bit increments, and still keep full control of logic mapping down to elementary gates, such as full-adder cells.

This paper is organized as follows: Section II describes the multiplication algorithms and the chosen implementations. In Section III, a gate-level analysis is conducted, comparing the modified-Booth and Baugh-Wooley implementation. Section IV describes the simulations that have been performed and the results of these; and finally the paper is concluded in Section IV.

5.2 Algorithms and Implementations

5.2.1 Algorithms for Modified Booth

The original Booth algorithm [5] for the multiplication $s = x \times y$ recodes partial products by considering two bits at a time of one of the operands (x) and encoding them into $\{-2, -1, 0, 1, 2\}$. Each such encoded higher-radix number is subsequently multiplied with the second operand (y), yielding one row of recoded partial-product bits. The advantage of Booth recoding is that the number of recoded partial products is fewer than the number of un-recoded partial products, and this can be translated into higher performance in the circuitry which reduces all partial-product bits into the final product.

The drawback of the original Booth algorithm is that the number of recoded partial products depends on input operand x, which makes this algorithm unsuitable for implementation in hardware. The modified-Booth (MB) algorithm [1] by MacSorley remedies this by looking at three bits at a time of operand x. Then we are guaranteed that only half the number of partial products will be generated, compared to a conventional partial-product generation using 2-input AND gates. Having a fixed number of partial products, the MB algorithm is suitable for hardware implementation.

								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

Figure 5.1: *Illustration of an 8-bit modified-Booth multiplication.*

A MB multiplier works internally with a two's complement representation of the partial products, in order to be able to multiply the encoded $\{-2, -1\}$ with the y operand. To avoid having to sign extend the partial products, the scheme presented by Fadavi-Ardekani [6] has been used. In the two's complement representation, a change of sign includes the insertion of a '1' at the least significant bit (LSB) position (henceforth called LSB insertion). To avoid an irregular implementation of the partial-product reduction circuitry, we draw on the idea of Yeh *et al.* [3] called modified partial-product array. Here, the impact of LSB insertion on the *two* least significant bits positions of the partial product is pre-computed. The pre-computation redefines the LSB of the partial product (Eq. 5.1 [3]) and moves the potential '1', which results from the LSB insertion, to the second least significant position (Eq. 5.2). Here Eq. 5.2 is different from that used by Yeh *et al.* [3].

$$p_{LSBi} = y_0(x_{2i-1} \oplus x_{2i}) \tag{5.1}$$

$$a_i = x_{2i+1}(\overline{x_{2i-1} + x_{2i}} + \overline{y_{LSB} + x_{2i}} + \overline{y_{LSB} + x_{2i-1}}) \tag{5.2}$$

Fig. 5.1 illustrates an 8-bit MB multiplication using sign extension prevention and the modified partial-product array scheme.

5.2.2 Modified-Booth Implementation

The implementation of the MB multiplier utilizes the regular reduction tree of the High Performance Multiplier (HPM) [7]; see Appendix. In Fig. 5.2 the reduction tree of an 8-bit MB implementation is shown. The long arrows with '1"s and a_i's show the places where these signals, which are in the lower row of the partial-product rhomboid in Fig. 5.1, have been inserted into the reduction tree. The figure does not show the MB recoding circuits.

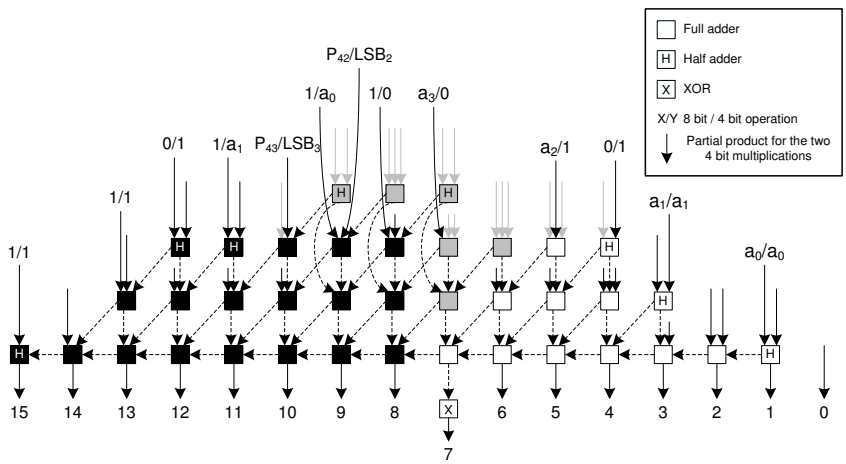
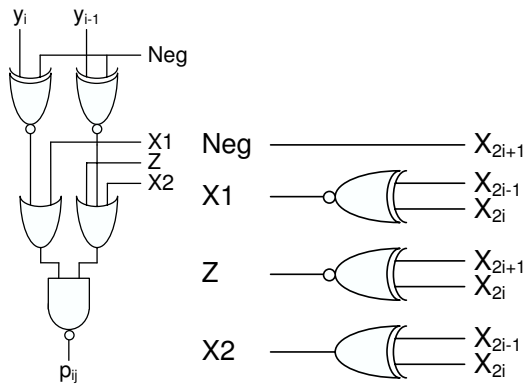


Figure 5.2: *8-bit modified-Booth multiplier using an HPM reduction tree. For simplicity, the modified-Booth recoding logic is not shown. Furthermore, a simple ripple-carry adder is used as final adder.*

The MB recoding logic can be designed in many different ways. For this comparison we have chosen the recoding scheme presented by Yeh *et al.* [3],

since this is claimed to be faster than competing schemes. The circuits for encoding and decoding are shown in Fig. 5.3.



(a) Decode (b) Encode

Figure 5.3: *Encode and decode circuit for modified Booth.*

5.2.3 Algorithms for Baugh-Wooley

The Baugh-Wooley (BW) algorithm [4] is a relatively straightforward way of doing signed multiplications; Fig. 5.4 illustrates the algorithm for an 8-bit case, where the partial-product bits have been reorganized according to Hatamian's scheme [8]. The creation of the reorganized partial-product array comprises three steps: *i)* The most significant bit (MSB) of the first $N-1$ partial-product rows and all bits of the last partial-product row, except its MSB, are inverted. *ii)* A '1' is added to the Nth column. *iii)* The MSB of the final result is inverted.

5.2.4 Baugh-Wooley Implementation

To the best of our knowledge, performance investigations in the open literature concerning BW implementations have exclusively been based on reduction arrays, in the spirit of the original paper [4]. In this investigation we have chosen

							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
						1	$\overline{p_{70}}$	p_{60}	p_{50}	p_{40}	p_{30}	p_{20}	p_{10}	p_{00}
						$\overline{p_{71}}$	p_{61}	p_{51}	p_{41}	p_{31}	p_{21}	p_{11}	p_{01}	
					$\overline{p_{72}}$	p_{62}	p_{52}	p_{42}	p_{32}	p_{22}	p_{12}	p_{02}		
				$\overline{p_{73}}$	p_{63}	p_{53}	p_{43}	p_{33}	p_{23}	p_{13}	p_{03}			
			$\overline{p_{74}}$	p_{64}	p_{54}	p_{44}	p_{34}	p_{24}	p_{14}	p_{04}				
		$\overline{p_{75}}$	p_{65}	p_{55}	p_{45}	p_{35}	p_{25}	p_{15}	p_{05}					
	$\overline{p_{76}}$	p_{66}	p_{56}	p_{46}	p_{36}	p_{26}	p_{16}	p_{06}						
p_{77}	$\overline{p_{67}}$	$\overline{p_{57}}$	$\overline{p_{47}}$	$\overline{p_{37}}$	$\overline{p_{27}}$	$\overline{p_{17}}$	$\overline{p_{07}}$							
$\overline{s_{15}}$	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

Figure 5.4: *Illustration of an 8-bit Baugh-Wooley multiplication.*

to implement the BW algorithm on the HPM reduction tree [7] instead; see Appendix.

Implementing the BW multiplier based on the HPM tree is as straightforward as the basic algorithm itself. The partial-product bits can be generated by using a 2-input AND gate for each pair of operand bits. In the case a partial-product bit should be inverted, we employ a 2-input NAND gate instead. The insertion of '1' in column N is easily accommodated by changing the half adder at top of row N, see Fig. 5.12 in Appendix, to a full adder with one of the input signals connected to '1'[1]. Finally, the inversion of the MSB of the result is done by adding an inverter. The final result of the implementation of the BW algorithm is depicted in Fig. 5.5.

5.3 Gate-Level Analysis

When comparing the Baugh-Wooley (BW) algorithm with the modified-Booth (MB) algorithm, the immediate reaction often is that the number of rows of

[1]The full-adder circuit can consequently be simplified, since one of its input signals is statically defined.

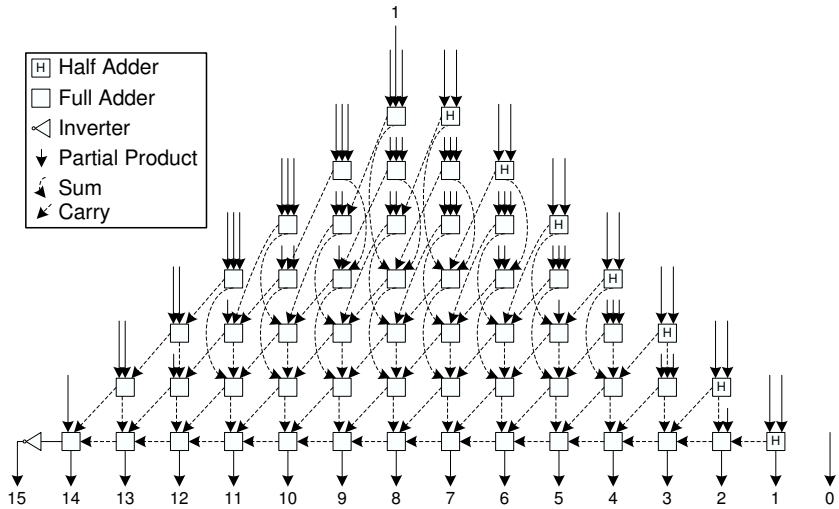


Figure 5.5: *8-bit Baugh-Wooley multiplier using an HPM reduction tree. For simplicity, the* AND/NAND *gates for partial-product generation are not shown. Furthermore, a simple ripple-carry adder is used as final adder.*

partial product bits should have a large impact on the total time for the multiplication. For sequential multipliers, where for each cycle one of the operands is added to a partial result, the observation above is true and it made it possible to construct the multiplication operator on a simple adder unit. Here, the reduction in rows from the MB algorithm has a direct effect on the cycle count for a complete multiplication.

At the time when the MB and BW algorithms were introduced in 1961 and 1973, respectively, hardware multipliers were at best built as an array of full adders. In an array multiplier, each row of partial products adds one full adder to the logic depth of the multiplier[2]; in this context the reduction in partial-product rows resulting from MB has great effect on delay.

[2]With the exception of the first two rows of partial products, which only result in one row of full adders.

However, in dedicated high-performance multipliers that are built around a reduction tree, the number of rows becomes less critical, because of the logarithmic nature of the reduction tree. With today's logarithmic reduction trees, such as TDM, Wallace, Dadda, or HPM, the logic depth increases with the maximum number of adders in a column, according to what is shown in Table 5.1. The logic depth is here defined as the maximum number of adders that has to be traversed to reach the final adder. The table is a modified version of the one Dadda [9] presented on the logic depth in a reduction tree built out of 3:2 adders, for an unsigned multiplier. To be able to evaluate the logic depth of the reduction tree for BW and MB, the logic depth is given for the maximum number of adders in a column, instead of for the operand width of the multiplier.

Table 5.1: *Logic depth through a logarithmic reduction tree, with respect to the maximum number of adders in a column.*

Maximum number of adders (A)	Logic depth
1	1
2	2
$2 < A \leq 5$	3
$5 < A \leq 8$	4
$8 < A \leq 12$	5
$12 < A \leq 18$	6
$18 < A \leq 27$	7
$27 < A \leq 41$	8
$41 < A \leq 62$	9
$62 < A \leq ...$	10

The maximum number of adders in a column of the reduction tree, for a certain operand width of BW and MB, can be obtained through the simple relation in Table 5.2.

Table 5.2: *Maximum number of adders in a column of the reduction tree for Baugh-Wooley and modified Booth.*

Operand width	Maximum number of adders	
	Baugh-Wooley	modified Booth
N	$N-2$	$N/2-1$

By combining the information on the logic depth through a reduction tree, Table 5.1, and the relation between the maximum number of adders and the operand width, Table 5.2, we can obtain the logic depth for different operand widths of the BW and MB implementations. The results are given in Table 5.3.

Table 5.3: *Logic depth through the reduction tree for Baugh-Wooley and modified Booth.*

Operand width	Logic depth		Difference in
	Baugh-Wooley	modified Booth	logic depth
8	4	3	1
16	6	4	2
32	8	6	2
40	8	7	1
48	9	7	2
60	9	8	1
64	10	8	2

As Table 5.3 shows, the logic depth of the reduction tree of MB is at best two full-adder delays shorter than for BW, for operand widths up to at least 64 bits. The shorter logic depth through the reduction tree of MB comes at the cost of a logic depth of one X(N)OR gate in the encode stage, followed by one 3-input OR, and one NAND gate in the decode stage. The HPM trees used for the two multiplier cases of this investigation have been signal-flow optimized; where possible, the sum from one full adder is connected to the carry input of the following full adder, thus achieving a three-XOR-gate logic depth when traversing two successive full adders. When comparing the logic depth of the MB and BW multipliers, it seems the MB multiplier has a slight

advantage. However, when also taking BW's simple and regular structure into consideration, it is likely that BW will be the better choice for implementation.

5.4 Simulations and Results

To validate the analysis in the previous section, a multiplier generator has been written [10]. The generator is capable of generating VHDL descriptions of both Baugh-Wooley (BW) and modified-Booth (MB) multipliers, according to the schemes presented above. For both multipliers, a Kogge-Stone [11] adder is used as the final adder that operates on the reduction tree outputs. The shape of the delay profiles, for the reduction tree outputs, are similar for BW and MB. Since the profiles differ only in absolute numbers, the final adder only adds a constant to the delay, Fig. 5.6.

By simulating exhaustive input patterns and verifying the result using Cadence NC-VHDL [12], the VHDL generator has been verified to generate functionally correct multiplier netlists of sizes up to at least 16 bits. For multipliers larger than 16 bits, the functionality has been verified for a finite random input pattern.

The VHDL descriptions have been synthesized using Synopsys Design Compiler [13] together with a commercially available 1.3 V 0.13-μm technology. To obtain delay and power dissipation data, Synopsys PrimeTime [14] and PrimePower [15] were used. We assumed a 10 fF load on the primary output signals and a cell-library buffer of medium size was chosen as driver for primary input signals. The chosen input buffer cell is specified to be capable of driving approximately 150 minimum-size XOR gates. This is of course an overkill for the smaller multipliers of this evaluation, but the same driver was used for the consistency of our analysis. For average power analysis, a clock period of 5 ns (200 MHz) and PrimePower's default values for switching activity and temperature were used.

The multipliers were not taken through place and route, but as will be shown later in this section, both multipliers have roughly the same footprint[3]. The total

[3]The BW multiplier is on average 37% smaller than MB, but on the other hand the latter has its

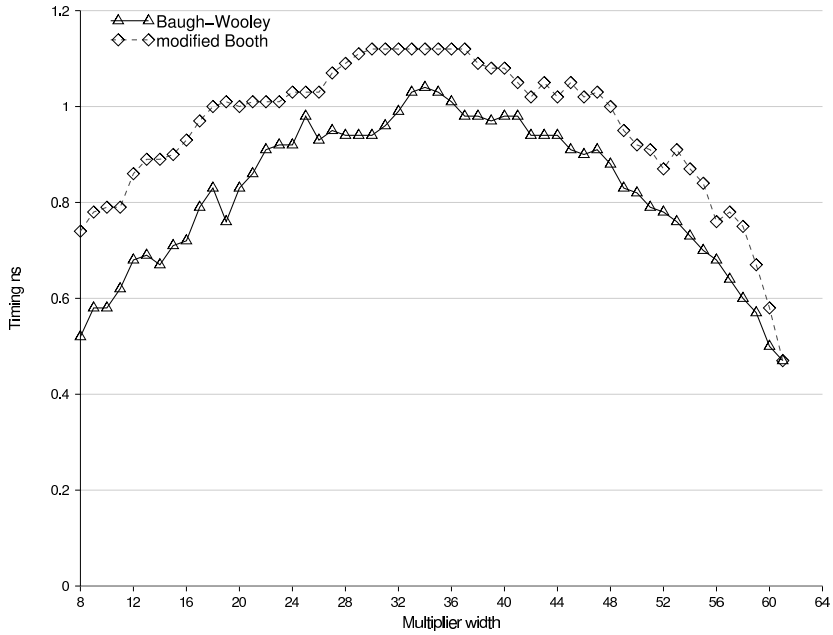


Figure 5.6: *Delay profile for the inputs to the final adder for the 32-bit Baugh-Wooley and modified-Booth cases. The delay for a specific bit is the average of the two full-adder inputs of each significance level of a final adder.*

wire length is, thus, expected to be equal for both multipliers and so is the effect of cross talk.

5.4.1 Synthesized Modified Booth Netlist

The first attempt of synthesizing the VHDL code for the generated MB multipliers exposed a serious problem with high fanout signals. By inspecting the encode and decode circuits in Fig. 5.3, we see that one of the primary input signals (x_{2i+1}) goes straight through the encoder, where it sets up the NEG signal, and drives two XNOR gates for each partial-product bit of a single row. This

wires partitioned in more segments.

means that the fanout of half of the x-inputs is at least $2N$ XNOR gates for a MB multiplier of size N. Further, the encoder outputs X1, Z, and X2 are driving N decoders, creating a need for large XOR and XNOR gates in the encode stage, which further increases the fanout of the x-inputs.

To tackle the fanout problem, we first offload the X1, Z, and X2 encoder outputs by instantiating more than one encoder for the N decoders associated with one partial-product bit row. Then, we reduce the high fanout of the NEG-signal by inserting an extra buffering inverter in each encode circuit. Finally, for each encoder, we use minimum-size inverters to buffer all three inputs x_{2i-1}, x_{2i}, and x_{2i+1}. The new encode circuit can be viewed in Fig. 5.7.

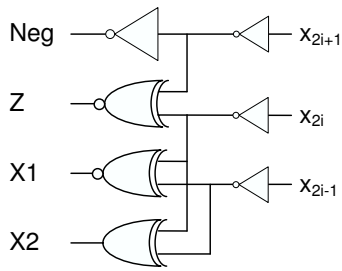


Figure 5.7: *Buffered encode circuit for fanout reduction.*

The graph in Fig. 5.8 shows the impact on total delay by limiting encoder-to-decoder fanout; the four cases represent when a maximum of 4, 8, 16, or 32 decoders have been connected to each of the new encoders of Fig. 5.7. Based on the graph, which shows that limiting encoder-to-decoder fanout yields a significant delay reduction, we chose to limit this fanout to four in our implementation. The fanout impact on power dissipation and area will be discussed later in this section.

During synthesis, the MB multiplier VHDL code to netlist mapping was constrained in that Design Compiler could not remove the minimum-size inverters that were added to the encoder as shown in Fig. 5.7. Half and full adders of the reduction tree were explicitly linked to their respective minimum-size cells and the map effort of Design Compiler was set to high.

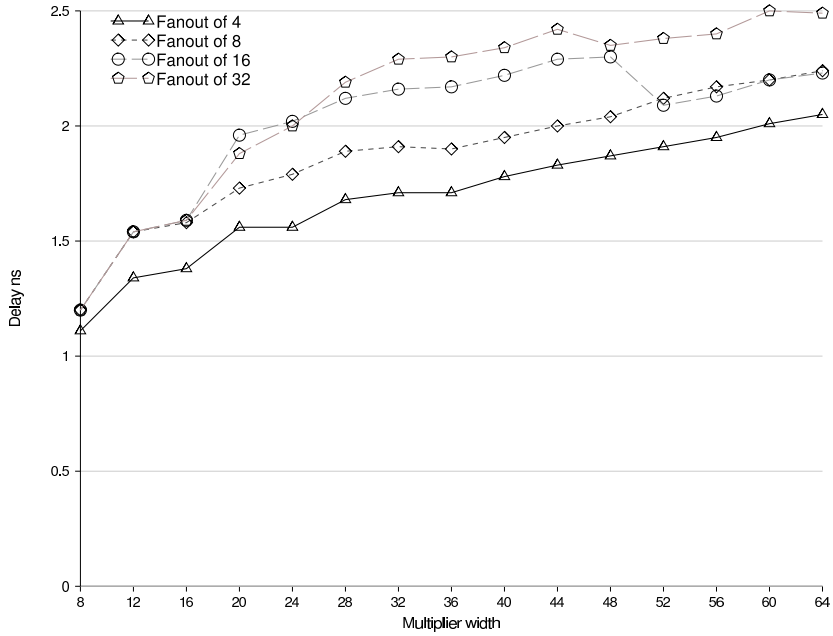


Figure 5.8: *Impact on the total delay of different encoder-to-decoder fanout.*

5.4.2 Synthesized Baugh-Wooley Netlist

The synthesis and subsequent timing analysis showed that the simplicity of the BW implementation comes with an added benefit; it does not lead to any high fanout signals. In the BW case, the signals with the highest fanout are the input signals that are connected to the input of the N 2-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 any significant performance degradation[4]. During synthesis, the BW multiplier VHDL code to netlist mapping was constrained in a similar way as for the MB netlist; i.e. half and full adders of the reduction tree were explicitly linked to their respective minimum-size cells and the map effort was set to high.

[4]This observation is based on medium-sized drivers for primary input signals.

5.4.3 Delay Comparison

To verify that BW's simple structure makes for a faster implementation, we have analyzed the delay of multipliers with operand widths from 8 up to 64 bits, with an increment of four bits. Fig. 5.9 shows the total delay for both types of multipliers. For all simulated multiplier sizes, the BW multiplier is faster than the MB multiplier. Only on two occasions, 36-bit and 52-bit width, comes the MB multiplier close to the delay of BW. On average the MB multiplier is 8% slower than BW.

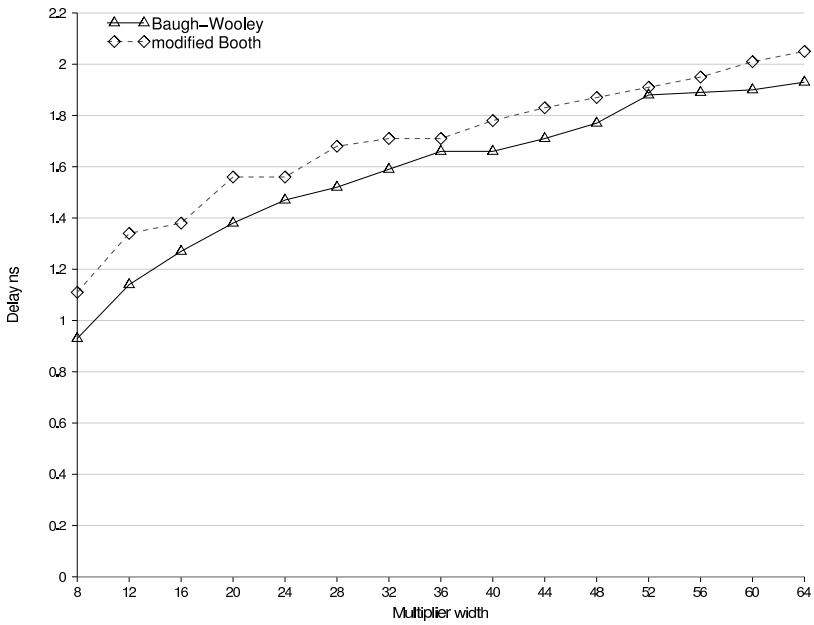


Figure 5.9: *Total delay (ns) of the Baugh-Wooley and modified-Booth multipliers.*

It should be noted that a significant effort went into improving the speed of the MB multiplier, while no attempt was made to improve the speed of the BW multiplier. As discussed earlier, the high fanout nodes of the MB recoding have been buffered to increase speed. Despite this effort, the MB multiplier remains slower than BW.

5.4.4 Power Comparison

For all operand widths, the BW multiplier has a significantly lower power dissipation than MB has; this is clearly shown in Fig. 5.10. On average the MB multiplier dissipates 51% more power than BW does.

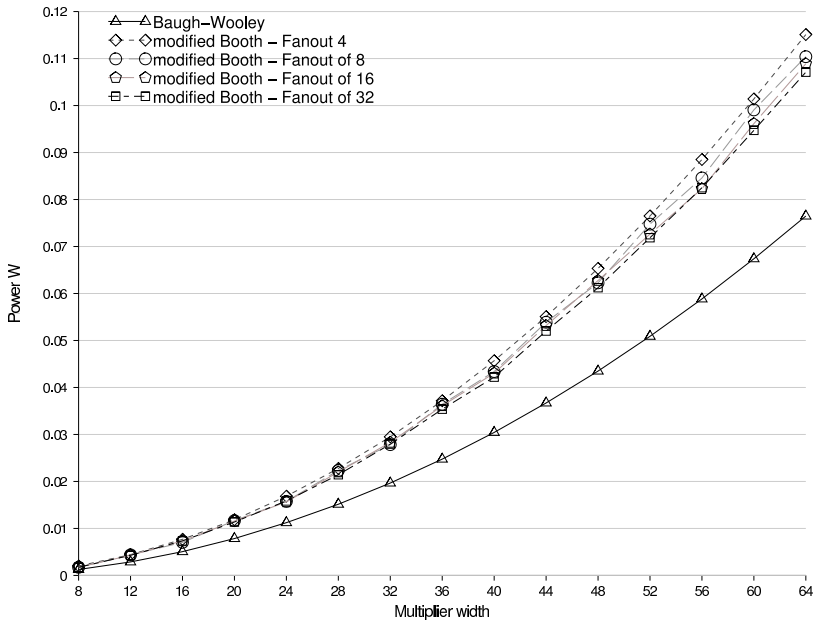


Figure 5.10: *Power (W) of the Baugh-Wooley and modified-Booth multipliers. The power for modified Booth is shown with respect to several different fanouts of the encoder.*

None of the designs were optimized for low power. As stated above, in the case of the MB multiplier, an effort was made to optimize the delay. The power dissipation for the MB multiplier could be reduced, e.g. by using a higher encoder-to-decoder fanout[5]. However, the resulting gain in power is very limited as can be seen in Fig. 5.10.

[5] A higher fanout affects the timing adversely, and accentuates the delay difference between MB and BW further.

5.4.5 Area Comparison

A comparison of the area of combinatorial cells, excluding wires, shows that the BW multiplier is the better choice. On average the MB multiplier is 37% larger than BW, with an area ratio that varies from 46% for a 8-bit multiplier to 35% for a 64-bit multiplier.

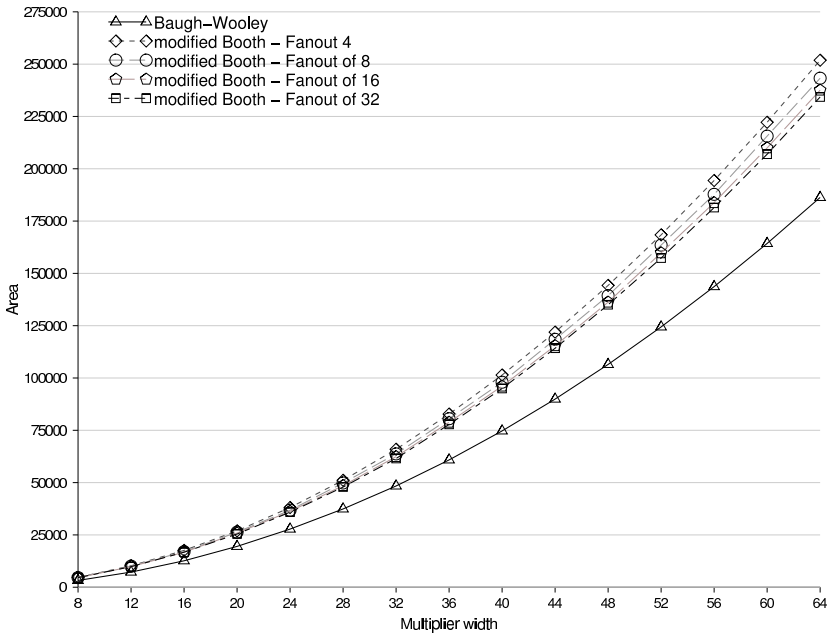


Figure 5.11: *Area of the Baugh-Wooley and modified-Booth multipliers. The area for modified Booth is shown with respect to different fanout of the encoder.*

As discussed in the power-comparison section, the multipliers have not been optimized for area or power. However, it should be observed that reducing the logic in the partial-product generation, by increasing the encoder-to-decoder fanout, has a minor effect on the area, Fig. 5.11.

5.5 Conclusions

This investigation has shown that the modified-Booth multiplier is, to say the least, not the obvious choice for high-performance multipliers. The low-complexity partial-product generation of the Baugh-Wooley algorithm makes this multiplier the better choice. A state-of-the-art modified-Booth multiplier performed worse than the Baugh-Wooley multiplier in the aspects of delay (8%), power (51%), and area (37%).

Acknowledgment

This research was funded by the Swedish Foundation for Strategic Research (SSF) through the FlexSoC project. We thank Dr. Lars 'J' Svensson for useful comments, and Dr. Henrik Eriksson for fruitful discussions in the early stage of this project.

Appendix

The High Performance Multiplier (HPM) Reduction Tree

Traditional reduction trees are irregular and difficult to implement. The High Performance Multiplier (HPM) [7] reduction tree is a *regular* reduction tree in that sum and carry signals are routed in a repeatable order, for all adder cells of the same row in the tree. This makes the HPM reduction tree suitable for HDL generators and automatic layout macros.

The summation of partial products for an array of adders is in the order of $\mathcal{O}(A-1)$, while the summation for the logarithmic reduction tree of HPM is proportional to $\mathcal{O}(\log_{\frac{3}{2}} A)$, where A is the maximum number of adders in a column.

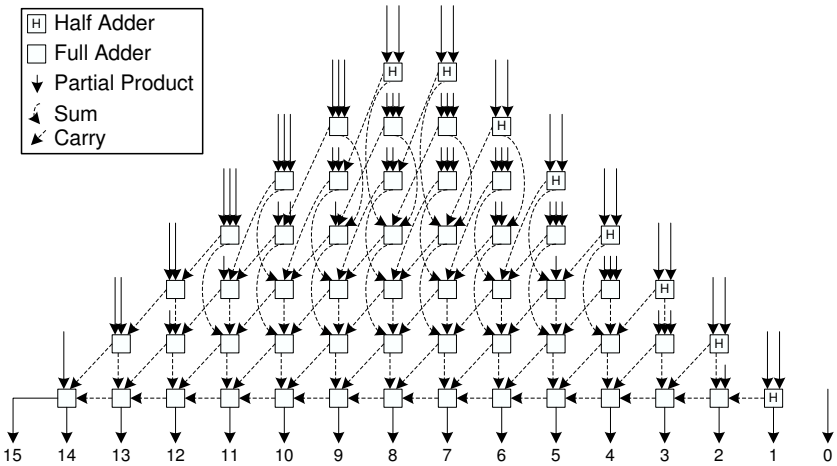


Figure 5.12: *The regular reduction tree of the HPM for an unsigned 8-bit multiplier.*

Bibliography

[1] O.L.MacSorley, "High Speed Arithmetic in Binary Computers," in *Proceedings of the IRE*, January 1961, vol. 49, pp. 67–97.

[2] David Villeger and Vojin G. Oklobdzija, "Analysis of Booth Encoding Efficiency in Parallel Multipliers Using Compressors for Reduction of Partial Products," in *Conference on Signals, Systems and Computers*, November 1993, vol. 1, pp. 781–784.

[3] Yeh Wen-Chang and Jen Chein-Wei, "High-Speed Booth Encoded Parallel Multiplier Design," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 692–701, 2000.

[4] Charles R. Baugh and Bruce A. Wooley, "A Two's Complement Parallel Array Multiplication Algorithm," *IEEE Transactions on Computers*, vol. 22, pp. 1045–1047, December 1973.

[5] Andrew D. Booth, "A Signed Binary Multiplication Technique," *Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 1951.

[6] Jalil 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.

[7] Henrik Eriksson, Per Larsson-Edefors, Mary Sheeran, Magnus Själander, Daniel Johansson, and Martin Schölin, "Multiplier Reduction Tree with Logarithmic Logic Depth and Regular Connectivity," in *IEEE International Symposium on Circuits and Systems*, May 2006.

[8] Mehid Hatamian, "A 70-MHz 8-bit x 8-bit Parallel Pipelined Multiplier in 2.5-μm CMOS," *IEEE Journal on Solid-State Circuits*, vol. 21, no. 4, pp. 505–513, August 1986.

[9] Luigi Dadda, "Some Schemes for Parallel Multipliers," *Alta Frequenza*, vol. 34, no. 5, pp. 349–356, May 1965.

[10] Magnus Själander, "HMS Multiplier Generator," `http://www.ce.chalmers.se/~hms/multiplier.html`, December 2005.

[11] Peter M. Kogge and Harold 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, August 1973.

[12] *Cadence NC-VHDL Simulator Help Version 5.1.*

[13] *Design Compiler User Guide Version W-2004.12.*

[14] *PrimeTime X-2005.06 Synopsys Online Documentation.*

[15] *PrimePower Manual Version W-2004.12.*

PAPER V

H. Eriksson, P. Larsson-Edefors, M. Sheeran,
M. Själander, D. Johansson, and M. Schölin

Multiplier Reduction Tree with Logarithmic Logic Depth and Regular Connectivity

IEEE International Symposium on Circuits and Systems
Island of Kos, Greece, May 21–24, 2006

6
PAPER V

A novel partial-product reduction circuit for use in integer multiplication is presented. The High-Performance Multiplier (HPM) reduction tree has the ease of layout of a simple carry-save reduction array, but is in fact a high-speed low-power Dadda-style tree having a worst-case delay which depends on the logarithm ($\mathcal{O}(\log N)$) of the word length N.

6.1 Introduction

The integer multiplier is a very important and wide-spread building block of digital designs. When multiplier delay is a critical design parameter, designers tend to use well-known logarithmic multipliers such as Wallace [1], Dadda [2], or TDM [3]. This class of multipliers is based on a *reduction tree*, in which

different schemes of compression of partial-product bits can be implemented, all with the result that the worst-case multiplier delay depends logarithmically on the factor word length.

To date, no regular way of inter-connecting the adding cells in such reduction trees has been proposed[1] and, thus, textbook array multipliers [5] are seriously considered [6] if layout regularity is the main concern and speed is not. It has, however, been shown by Callaway and Swartzlander [7] that not only speed is sacrificed when an array multiplier is used instead of a logarithmic one, but also *power*. When comparing netlists of the reduction trees of a 32-b Wallace, a 32-b Dadda, and a 32-b TDM multiplier with the partial-product compression array of a 32-b carry-save multiplier, the reduction trees on the average proved to be 2.3× faster *and* 3.0× more power efficient than the array [8].

We propose a new organization of the reduction tree, which is based on a partial-product compression similar to the Dadda approach. The connectivity of the adding cells in the triangle-shaped *High-Performance Multiplier (HPM)* reduction tree is completely regular. At the same time it exhibits a worst-case delay which depends on the logarithm ($\mathcal{O}(\log N)$) of the word length N.

6.2 The HPM Reduction Tree

Typically parallel multipliers comprise three parts: *primary partial-product bit generation*, performed either by simple AND gates or Booth [9] or modified Booth recoding strategies [10]; *partial-product bit compression*, using either an irregular logarithmic tree or a regular array; and *final addition*, consuming the output of the partial-product bit compression with an adder whose performance is tailored to match the output delay profile of the compression circuitry [3].

The focus of this paper is on a reduction tree that is easy to place and route *and* has a logarithmic logic depth. We will describe a connectivity strategy that enables a regular organization of the cells in the reduction tree. However, first

[1] Luk and Vuillemin [4] introduced a regular layout of a reduction tree with logarithmic depth, however, since the primary outputs of the tree are embedded in the center of the layout, it is difficult to interface the reduction tree with the final adder.

we need to select a multiplier to start from, our *model* multiplier. In principle, we can select any of the three logarithmic multipliers that were mentioned in the introduction: Wallace, Dadda, or TDM multipliers. Although it is claimed [11] to be less suitable for custom implementations than Wallace, we have selected the Dadda multiplier as our model multiplier. The main reason for this choice is that regardless of the word length N the Dadda reduction tree always makes use of a minimal amount of circuitry, M_{OPT} [3]:

$$M_{OPT} = (N-1) \cdot (N-2) \tag{6.1}$$

Here, $N-1$ cells are half adders, and the remainder are full adders.

Regarding the connectivity of cells, we are assuming a triangular reduction tree rather than a rectangular. The reason for this choice is that a triangular cell placement has a shorter total wire length than a rectangular one (see Sec. 6.2.1). Note that the unused area within a bounding box surrounding the triangular reduction tree could be used for partial-product bit generating logic and decoupling capacitance. Thus the resulting circuit footprint is rectangular also in this case.

6.2.1 Design Method

The Dadda is closely related to the Wallace multiplier in that they both have an $\mathcal{O}(\log N)$ reduction in the reduction tree. In comparison to the Dadda multiplier algorithm, the Wallace algorithm creates a shorter final pair of rows. There are claims [11, 12] that the shorter final adder makes the Wallace multiplier faster than Dadda. This is not true, since it is the delay profile from the reduction tree that matters and the delay profiles are almost identical for Wallace and Dadda multipliers.

The four steps needed to obtain an 8-b Dadda multiplier are shown in Fig. 6.1. The 64 partial-product bits are compressed into two rows, which are fed to a fast final adder. In this representation, where carry, sum, and *primary* partial-product bits are all represented as dots, it is hard to interconnect the adding cells in a regular fashion. However, if special care is taken during the assignment of adding cells, the result is the logarithmic depth High-Performance Multiplier

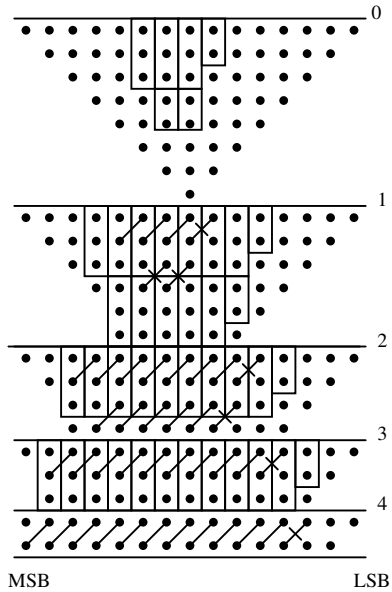


Figure 6.1: *The four steps leading to an 8-b Dadda multiplier. Two dots that are tied together by a line represent a sum and a carry. If the line is crossed, it is a half adder producing the sum and carry, otherwise it is a full adder. The cell type is also indicated by the number of bits that are encircled.*

(HPM) reduction tree shown in Fig. 6.3(a). Circuitry for primary partial-product bit generation is omitted in the figure—the very short wires represent primary partial-product bits.

The three-over-two compression is visible when inspecting Fig. 6.3(a) since every other row of adding cells takes two carry vectors and one sum vector as inputs, whereas the other rows take one carry vector and two sum vectors. The encircling scheme leading to an HPM tree is shown in Fig. 6.2.

Fig. 6.3 illustrates why a triangular cell placement is preferred for the HPM: wire routing is simple and thus it is easy to write a generator—or to make a custom design. Also, if the wires used to route the primary partial-product bits are ignored, the total wire length for an 8-b multiplier is approximately 38%

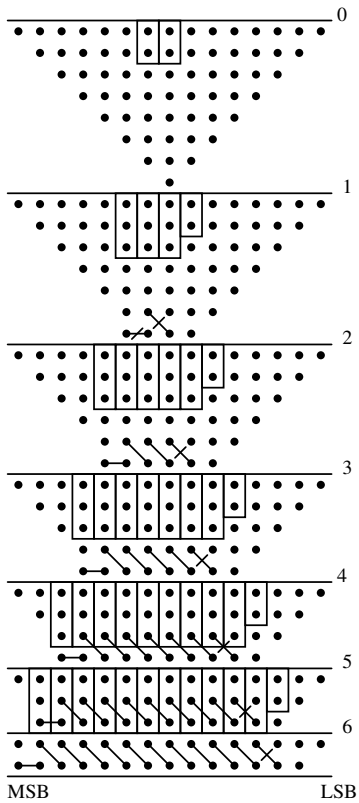
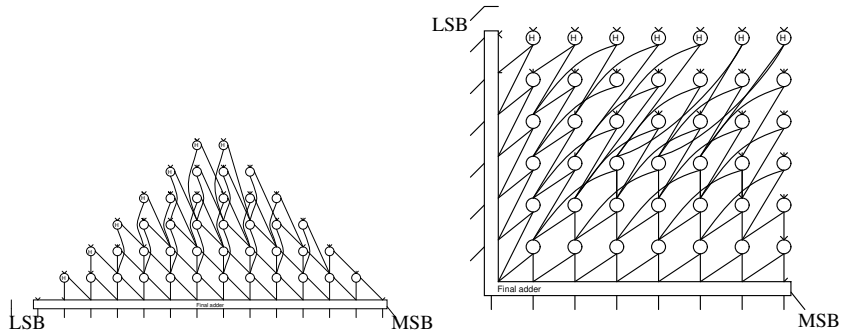


Figure 6.2: *The six steps leading to an 8-b HPM tree. Note that the six steps do not imply a linear logic depth. The logic depth is still logarithmic.*

shorter for the triangle compared to the rectangle. (Here, we assumed square-shaped adding cells and that 45 degree wires are allowed. The estimated wire length is calculated as the distance between the cells it connects.) The primary partial-product bits can be generated locally within the reduction tree, or outside it; either way more wires than just sum and carry wires need to be routed inside the reduction tree.

Modified Booth recoding could, of course, be employed to reduce the number of primary partial-product bits, but that is out of the scope of this paper.



(a) Triangular (b) Rectangular

Figure 6.3: *Different cell placements for the HPM reduction tree. The rectangular shape has larger total wire length.*

However, there is no difference between the HPM tree and a Wallace (or a Dadda) tree in the way Booth recoding is used.

6.2.2 Layout Organization

In the following, we assume that the primary partial-product bits are generated outside the tree. We have implemented both the HPM reduction tree, and for comparison, the reduction circuitry from the carry-save array (CSA) multiplier, which is well known for its regular connectivity. The CSA multiplier was slightly modified to better fit the comparison with the HPM reduction tree: the carry-save concept was used in a triangular minimal-hardware placement, rather than in the more common rectangular placement.

We have found a way to build the reduction circuitry for *both* the CSA and HPM multipliers using the *same overall structure of cells*, which is depicted in Fig. 6.4. Our task now is to design the individual cells so as to obtain the desired routing between cells.

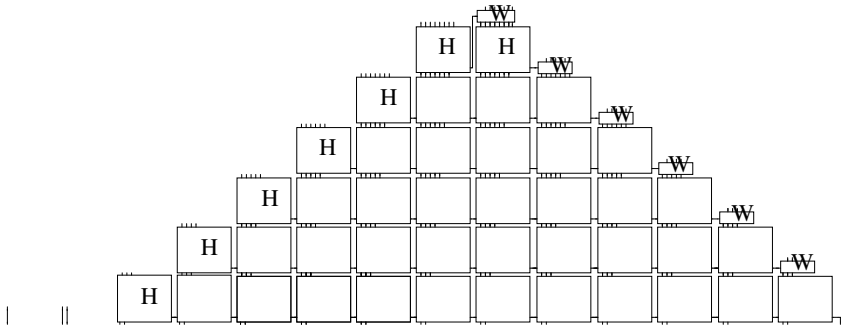
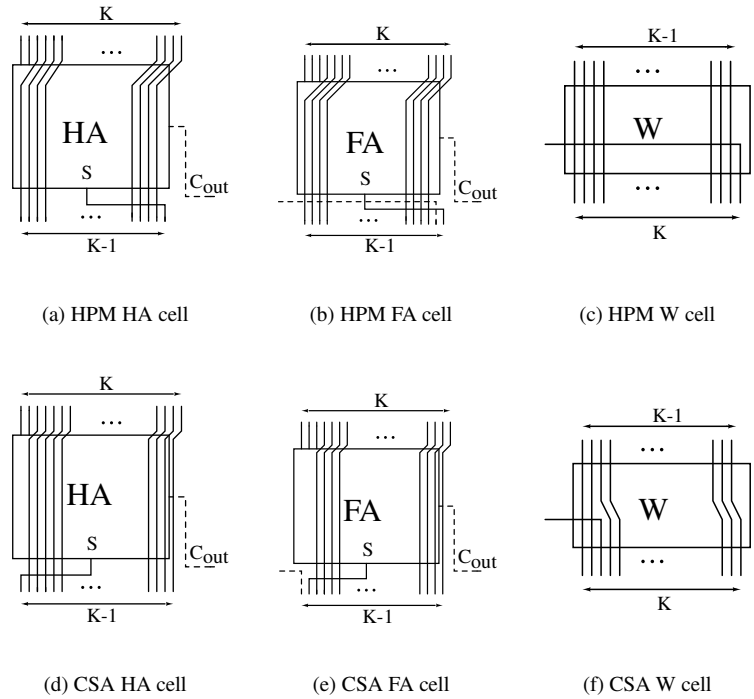


Figure 6.4: *The overall reduction structure used to build both the HPM and CSA multipliers. Boxes marked H contain half adders and wiring, unmarked boxes contain full adders and wiring, and cells marked W contain only wiring.*

We start with the cells containing half adders. Figs 6.5(a) and 6.5(d) show the over-the-cell routing patterns for the HPM and CSA respectively. K partial-product bits enter at the top, and the role of the HA cell is to reduce their number by one and to produce an output carry that is passed rightwards to the next column. A half adder is used to convert the leftmost two partial-product bits into a sum and an output carry. The output carry is passed rightwards, and the only remaining question is how to place the sum bit among the remaining $K-2$ partial-product bits, to give the desired $K-1$ outputs. For the HPM, we have already decided to process sum and carry bits as late as possible, so we place the sum bit as far to the right as possible, since the adding cells process bits from the left. For the CSA, the sum bit should be processed by the cell below this one, so we place the sum bit as far to the left as possible. The cells containing full adders are a little more complicated, because they also have an input carry bit. However, the strategy is similar: For the HPM, we process the sum bit as late as possible, as before, and we also process the input carry as late as possible, Fig. 6.5(b). Thus, these two bits are passed as far to the right as possible. For the CSA, these two bits are instead passed as far to the left as possible, so that they are processed by the cell immediately below this one, Fig. 6.5(e). Similarly, the W cells are designed to give the required connectivity, Figs 6.5(c) and 6.5(f).



(a) HPM HA cell (b) HPM FA cell (c) HPM W cell

(d) CSA HA cell (e) CSA FA cell (f) CSA W cell

Figure 6.5: *The routing patterns for full adder, half adder and wiring cells in the HPM and the CSA multipliers. In both cases, these cells are combined into the structure shown in Fig. 6.4. As long as primary partial-product bits remain to be compressed, the HPM compresses only those, ignoring sum and carry bits as long as possible. This is in contrast to the CSA, in which most FA cells consume one primary partial-product bit for each compression stage.*

As a sanity check, consider the leftmost of the two tallest columns in the overall structure shown in Fig. 6.4. This column receives 8 primary partial-product bits from the top, and 5 input carry bits from the left. The 6 cells produce a total of 6 sum bits. The following sequence indicates the order in which these 19 bits are consumed: [`pp`, `ppp`, `ppp`, `scs`, `csc`, `scs`, `cs`], starting with the two primary partial-product bits consumed by the half adder, and

ending with the carry/sum pair that is the output at the bottom of the column. The remaining groups of three are the inputs to successive full adders as we move down the column. Here, the pattern of alternating sum and carry bits, processed late, is clear. For the same column in the CSA, however, the pattern is [pp, spp, csp, csp, csp, csp, cs]. The input carry and sum for a given cell are consumed in the cell below. The reader might like to compare these patterns with the relevant columns in Fig. 6.3(a).

6.3 Evaluation

First, a 16-b HPM reduction tree was formally verified using gate-level VHDL netlists in a functional equivalence-checking tool (Formality). Then we evaluated the HPM tree in three different ways: *i)* To validate the logarithmic delay, we compared HPM to both Wallace and TDM trees. *ii)* To evaluate the relative crosstalk sensitivity of HPM, we compared it to the CSA array. *iii)* Finally, to make sure the glitch power associated with regular reduction circuits is effectively suppressed, we compared the power dissipation of HPM to the CSA array.

6.3.1 Timing Comparison (i)

We used PathMill to analyze our 0.35-μm schematic designs of HPM, Wallace (WAL) and TDM. Since we analyzed several wordlengths, we used a generator that inserted DC wire loads that had been calibrated to actual layouts. The delay values are presented in Table 6.1. Note that the delay values are for complete multipliers, when a Kogge-Stone adder [13] is used as the fast final adder. Here identical adding cells were used for all multiplier types, that is, we did not take advantage of the predictable cell placements to further increase HPM multiplier speed by circuit sizing.

All multiplier exhibit delays that are logarithms of wordlength. As expected the TDM is the fastest, but in exchange it requires the largest design effort, since it has irregular connectivity and an elaborate selection/assignment of cells. The

Table 6.1: *Simulated delay values for complete multipliers.*

Type	WAL		HPM		TDM	
N	32	54	32	54	32	54
Delay [ns]	8.22	10.6	7.94	10.5	7.80	10.2

WAL multiplier is slightly slower than the HPM and this may be due to its more complex wiring.

6.3.2 Timing Comparison (ii)

Due to its regularity, the HPM tree has some critical wires that run in parallel for a relatively long distance. Thus, crosstalk-induced delay may be a serious issue. To uncover potential problems with crosstalk in critical inter-cell wires, we used PathMill to compare the crosstalk sensitivity of a 16-b HPM to that of a 16-b CSA, which has no long wires in parallel, but many shorter. We used netlists of fully extracted 0.13-μm layouts of complete reduction trees. The results are as follows: The HPM had a 43% increase in delay when comparing a netlist *without* crosstalk to one *with* crosstalk capacitances. For the CSA the delay increase was 37%. Thus, the crosstalk of an HPM tree is not significantly worse than other trees. On the contrary, it should be noted that since wiring in the HPM tree is regular, increased wire spacing can be applied in a systematic way to reduce crosstalk.

6.3.3 Power Comparison (iii)

We also compared the power dissipation of the HPM reduction tree to the CSA reduction circuitry using HSpice. Here we used post-layout extracted 0.35-μm netlists together with random input data. In our analysis we found that a 16-b HPM reduction tree is 1.9$\times$ more power efficient than the CSA reduction array, and this is due to a smaller number of transitions [7].

6.4 Conclusion

We have introduced a new reduction tree for integer multiplication, the High-Performance Multiplier (HPM) reduction tree, which has an $\mathcal{O}(\log N)$ delay dependence on word length N. The connectivity of adding cells in the reduction tree is regular and we have shown that routing becomes trivial, which can be utilized in any type of design method; fully automatic, custom, or somewhere in between. In contrast to other logarithmic multipliers, like Wallace, the design effort for a custom-made HPM multiplier is very limited; in fact we showed that it is as low as for a textbook array multiplier. The predictable wiring resulting from the regularity of the HPM tree can enable both systematic sizing of logic circuitry and systematic wire spacing engineering so as to minimize total multiplier delay.

Acknowledgments

Thanks to Per Bjesse at Synopsys, Inc., who verified the HPM multiplier using Formality. We gratefully acknowledge equipment grants from Intel Corporation to each of our research groups.

Bibliography

[1] Christopher S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Transactions on Electronic Computers*, vol. 13, pp. 14–17, February 1964.

[2] Luigi Dadda, "Some Schemes for Parallel Adders," *Alta Frequenza*, vol. 34, no. 5, pp. 349–356, May 1965.

[3] Vojin G. Oklobdzija, David Villeger, and Simon S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," *IEEE Transactions on Computers*, vol. 45, no. 3, pp. 294–306, March 1996.

[4] W. K. Luk and J. E. Vuillemin, "Recursive Implementation of Optimal Time VLSI Integer Multipliers," in *Proceedings of VLSI'83*, 1983, pp. 155–168.

[5] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic, *Digital Integrated Circuits, A Design Perspective*, Prentice Hall, second edition, 2003.

[6] James T. Kao, Masayuki Miyazaki, and Anantha P. Chandrakasan, "A 175-mW Multiply-Accumulate Unit Using an Adaptive Supply Voltage and Body Bias Architecture," *Journal on Solid-State Circuits*, vol. 37, no. 11, pp. 1545–1554, November 2002.

[7] Thomas K. Callaway and Earl E. Swartzlander, Jr., "Optimizing Multipliers for WSI," in *Proceedings of the Fifth Annual IEEE International Conference on Wafer Scale Integration*, 1993, pp. 85–94.

[8] Henrik Eriksson, *Efficient Implementation and Analysis of CMOS Arithmetic Circuits*, Ph.D. Thesis, Chalmers University of Technology, 2003.

[9] Andrew D. Booth, "A Signed Binary Multiplication Technique," *Quarterly Journal Mechanical and Applied Mathematics*, vol. 4, pp. 236–240, 1951.

[10] Yeh Wen-Chang and Jen Chein-Wei, "High-Speed Booth Encoded Parallel Multiplier Design," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 692–701, July 2000.

[11] Michael J. S. Smith, *Application-Specific Integrated Circuits*, Addison-Wesley Publishing Company, first edition, 1997.

[12] K'Andrea C. Bickerstaff, Michael Schulte, and Jr. Earl E. Swartzlander, "Reduced Area Multipliers," in *Proceedings of the International Conference on Application-Specific Array Processors*, 1993, pp. 478–489.

[13] Peter M. Kogge and Harold 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, August 1973.