CHALMERS

Technical Report No. 08-8

The Case for HPM-Based Baugh-Wooley Multipliers

MAGNUS SJÄLANDER
PER LARSSON-EDEFORS

Department of Computer Science and Engineering
Division of Computer Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, 2008

The Case for HPM-Based Baugh-Wooley Multipliers

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

March 4, 2008

Abstract

The modified-Booth algorithm is extensively used for high-speed multiplier circuits. Once, when array multipliers were used, the reduced number of generated partial products significantly improved multiplier performance. In designs based on reduction trees with logarithmic logic depth, however, the reduced number of partial products has a limited impact on overall performance. The Baugh-Wooley algorithm is a different scheme for signed multiplication, but is not so widely adopted because it may be complicated to deploy on irregular reduction trees. We use the Baugh-Wooley algorithm in our High Performance Multiplier (HPM) tree, which combines a regular layout with a logarithmic logic depth. We show for a range of operator bit-widths that, when implemented in 130-nm and 65-nm process technologies, the Baugh-Wooley multipliers exhibit comparable delay, less power dissipation and smaller area foot-print than modified-Booth multipliers.

1 Introduction

Multiplication is an important arithmetic operation and multiplier implementations date several decades back in time. Multiplications were originally performed by iteratively utilizing the ALU's adder. As timing constraints became stricter with increasing clock rates, dedicated multiplier hardware implementations such as the array multiplier were introduced. Since then ever more sophisticated methods on how to implement multiplications have been proposed. One of the more popular implementations is that of the modified-Booth recoding scheme together with a logarithmic-depth reduction tree and a fast final adder. Modified-Booth recoding has the advantage of reducing the number of generated partial products by half, compared to partial-product generation based on 2-input AND-gates. This fact decreases the size of the reduction circuitry, which commonly is a logarithmic-depth reduction tree, e.g. Wallace, Dadda or TDM. Since such reduction trees are infamous for their irregular structures, which make them difficult to place and route during the physical layout of a multiplier, a decreased size of the reduction circuit eases the implementation and improves the performance of the multiplier.

Modified-Booth implementation strategies are commonly motivated by the need for fast multipliers. However, with the ongoing integration trend for which power dissipation is an ever pressing concern, modified Booth is no longer the obvious implementation choice. Already in 1997 Callaway *et al.* showed that, for a 2 μm process technology, a Wallace multiplier is more energy efficient than a modified-Booth multiplier [1]. Even though power has become a bigger concern since then, the modified-Booth multiplier is still prevailing as the main implementation choice for high-speed multipliers.

In this paper we compare a multiplier based on the modified-Booth algorithm against one based on the less used Baugh-Wooley algorithm. As will be shown in Sec. 8, a Baugh-Wooley multiplier implemented in a 130-nm and 65-nm process technology is more power and energy efficient than a modified-Booth multiplier of equal bit-width. This efficiency comes with only an insignificant increase in delay. We will subsequently show that the high power dissipation makes modified Booth a poor implementation choice for high-speed multipliers.

2 Modified-Booth Multiplication

The original Booth algorithm [2] 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 [3] 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. Since it has a fixed number of partial products, the MB algorithm is suitable for hardware implementation.

An 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, we are using the scheme presented by Fadavi-Ardekani [4]. 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 called modified partial-product array [5]. 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. 1 in [5]) and moves the potential '1', which results from the LSB insertion, to the second least significant position (Eq. 2). Note that our Eq. 2 is different from the corresponding equation used in [5]. Fig. 1 illustrates an 8-bit MB multiplication using sign extension prevention and the modified partial-product array scheme.

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

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

							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: Illustration of an 8-bit modified-Booth multiplication.

2.1 Modified-Booth Implementation

The implementation of the MB multiplier utilizes the regular reduction tree of the High Performance Multiplier (HPM) [6]. In Fig. 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. 1, have been inserted into the reduction tree. The figure does, however, not show the MB recoding circuits.

The MB recoding logic can be designed in many different ways. For the subsequent comparison we have chosen to implement two different, recent recoding schemes. The first recoding scheme is the one presented by Yeh *et al.* [5], since this is claimed to be faster than many previous schemes. The circuits for encoding and decoding using this scheme are shown in Fig. 3.

We have also chosen to implement the recoding scheme of Hsu *et al.* [7]. We chose to implement this recoding scheme because it is a recent one, showing promising results. The encoding and decoding circuits are shown in Fig. 4.

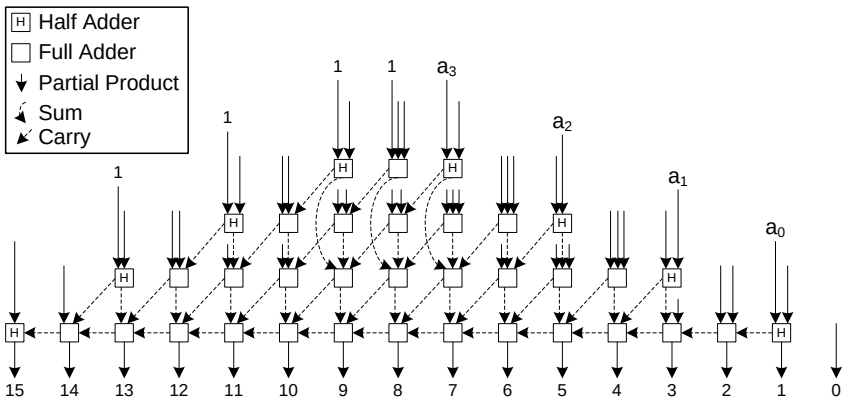
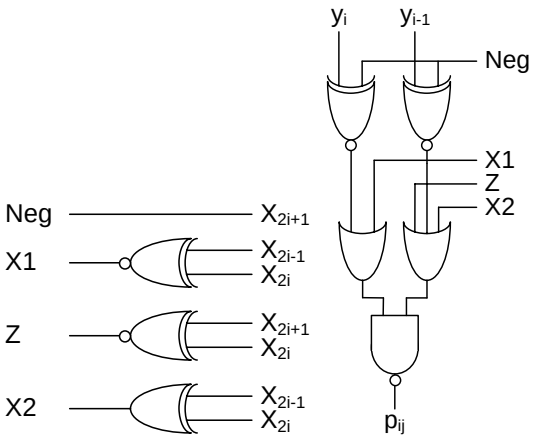
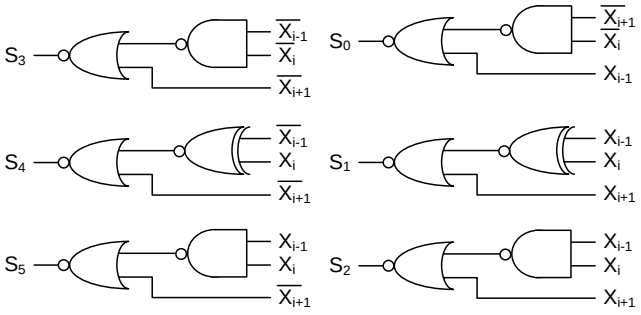


Figure 2: 8-bit modified-Booth multiplier using an HPM reduction tree. For simplicity, the modified-Booth recoding logic is not shown. Furthermore, a simple ripple-carry adder is used as final adder.

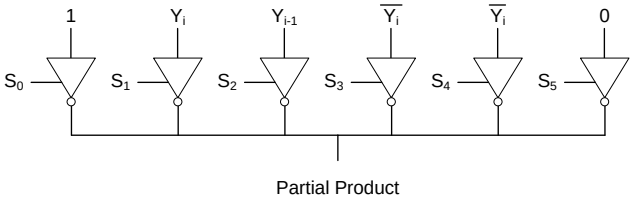


(a) Encode (b) Decode

Figure 3: Modified-Booth recoding circuits according to Yeh *et al.*.



(a) Encode



(b) Decode

Figure 4: Modified-Booth recoding circuits according to Hsu *et al.*.

3 Baugh-Wooley Multiplication

The Baugh-Wooley (BW) algorithm [8] is a relatively straightforward way of doing signed multiplications; Fig. 5 illustrates the algorithm for an 8-bit case, where the partial-product bits have been reorganized according to Hatamian's scheme [9]. 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.

							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: Illustration of an 8-bit Baugh-Wooley multiplication.

3.1 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 [8]. In this investigation, however, we have chosen to implement the BW algorithm on the HPM reduction tree [6] instead.

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

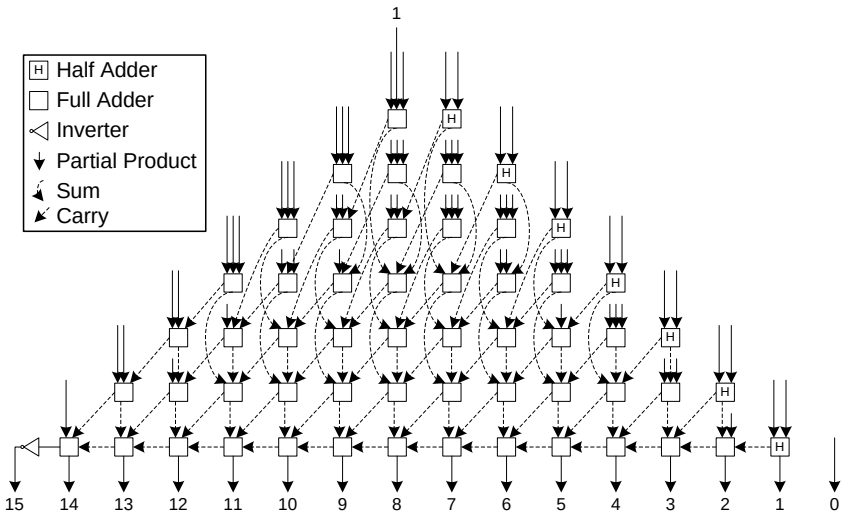


Figure 6: 8-bit Baugh-Wooley multiplier using an HPM reduction tree. For simplicity, the AND/NAND gates for partial-product generation are not shown. Furthermore, a simple ripple-carry adder is used as final adder.

of row N 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. 6.

4 An Initial Gate-Level Study

A widespread belief is that the number of rows of partial-products bits has large impact on the total time to carry out a multiplication. For sequential multipliers, where for each cycle one of the operands is added to a partial result, the number of partial products to be added is indeed critical. Here, the reduction in rows resulting from Booth encoding has a direct and significant effect on the cycle count for a complete multiplication.

At the time when the modified-Booth (MB) and Baugh-Wooley (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-product bits 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 timing.

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 gate structure of the tree. With today's logarithmic-depth reduction trees, such as TDM, Wallace, Dadda, and HPM, the logic depth increases with the logarithm of the maximum number of adders in a column. The logic depth for the HPM is shown in Table 1. The logic depth is here defined as the maximum number of adders that have to be traversed to reach the final adder in an HPM tree built out of 3:2 adders (full adders). 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 1: Logic depth through the HPM tree, with respect to the maximum number of adders in a column.

Maximum number of adders (A)	Logic depth
$A = 1$	1
$A = 2$	2
$3 \leq A \leq 4$	3
$5 \leq A \leq 7$	4
$8 \leq A \leq 11$	5
$12 \leq A \leq 17$	6
$18 \leq A \leq 26$	7
$27 \leq A \leq 40$	8
$41 \leq A \leq 61$	9
$62 \leq A \leq 92$	10

The maximum number of adders in a column of the HPM reduction tree, for a certain operand width of BW and MB, can be obtained through the simple relation in Table 2.

Table 2: Maximum number of adders in a column of the HPM tree based on the partial-product generation according to Baugh-Wooley and modified Booth.

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

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

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

By combining the information on the logic depth through the HPM reduction tree, Table 1, and the relation between the maximum number of adders and the operand width, Table 2, we can obtain the logic depth for different operand widths of the BW and MB implementations. The results are given in Table 3.

Table 3: Logic depth through the HPM reduction tree based on the partial-product generation according to 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
54	9	7	2
60	9	8	1
64	10	8	2

As Table 3 shows, the logic depth of the reduction tree designed for modified Booth is at best two full-adder delays shorter than that designed for Baugh-Wooley, for operand widths up to at least 64 bits. The shorter logic depth through the MB reduction tree comes at the cost of a partial-product generation circuit that is more complex than that of BW, whose partial-product generation circuit utilizes only a simple 2-input AND-gate for each generated partial-product bit.

The conclusion of this initial gate-level study is that it is certainly not obvious that the deployment of the modified-Booth algorithm onto a fast reduction tree yields a faster implementation than a corresponding Baugh-Wooley implementation.

5 Multiplier Evaluation Setup

To evaluate the different multiplier designs a multiplier generator [10] was created. This generator is capable of generating gate-level VHDL netlist descriptions of modified-Booth (MB) and Baugh-Wooley (BW) multipliers of various operand sizes, according to the schemes presented in the previous sections. Both the MB recoding scheme of Yeh *et al.* and Hsu *et al.* have been implemented. All netlists are based on the regular reduction tree of a High Performance Multiplier (HPM) [6], based on 3:2 adder (full adder) cells. A Kogge-Stone [11] adder is used as the final adder, placed on the reduction tree outputs.

By exhaustively simulating all input patterns and verifying the result using Cadence NC-VHDL [12], the VHDL generator was verified to generate functionally correct multiplier netlists of sizes up to at least 16 bits. For multipliers larger than 16 bits, the functionality of each multiplier size was verified for a random pattern of one million vectors.

The VHDL descriptions were synthesized using Synopsys Design Compiler [13] together with a commercially available 1.2 V 130-nm process technology. The synthesized netlist was taken through place and route using Cadence Encounter [14]. To create a common input and output interface to the multipliers and to enable higher place and route quality from the EDA tools, which do not handle purely combinatorial circuits well, registers were placed on the inputs and outputs of the multipliers. Delay estimates were obtained after RC extraction from the placed and routed netlists. Power estimates were derived from the same netlists, using value change dump (VCD) data from simulations of 10,000 random input vectors. All delay, power, and area figures that we present, include the input and output registers on the multipliers and are for the worst case corner at 1.08V.

Initially, the timing constraints were systematically set so as to push the limits of the timing that can be achieved for each generated VHDL description. This guarantees that a very fast implementation is obtained, however, the power dissipation becomes prohibitively high, due to excessive buffering and resizing of gates. Therefore, for each implementation, the timing was subsequently relaxed with 100, 300, and 600 ps, respectively, relative to the fastest timing obtained. Correspondingly, power estimates were obtained for each timing constraint.

6 Modified-Booth Multiplier Evaluation

Before we can compare a modified-Booth (MB) multiplier with a Baugh-Wooley (BW) multiplier, we first have to ascertain which of the previous two MB recoding schemes is best. The first scheme was that of Yeh *et al.* [5], as shown in Fig. 3, while the second scheme was that of Hsu *et al.* [7], as shown in Fig. 4. The original circuit of Hsu uses transmission gates in the decoding circuit. However, with the cell library used for this evaluation we are limited to tristated inverters, which may hamper the efficiency of this scheme.

6.1 Yeh Recoding

If we consider the fanout for the recoding circuits of the MB multiplier using the Yeh recoding scheme, we notice that each decoder needs to drive N decoders, for a multiplier of size N. To find a trade off between the fanout of the x-inputs that drive the encoders and the fanout of the encoder output signals, we instantiate more than one encoder for the N decoders associated with one partial-product bit row. To reduce the load of the x-inputs, we use minimum-size inverters to buffer all three inputs of the encode circuit, as shown in Fig. 7.

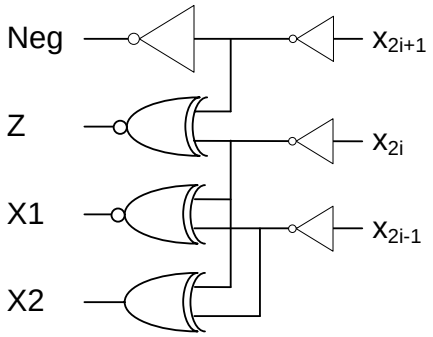


Figure 7: Buffered encode circuit for fanout reduction.

To find a good trade off between the fanout of the x-inputs and the outputs of the encoding circuit, we varied the maximum number of decode circuits that are connected to a single encode circuit. The netlists of 32- and 64-bit multipliers with a maximum of 12, 16, or 20 decoders per encoder, were taken through synthesis, placement, and route. The delay and energy-per-operation can be found in Table 4. As can be seen from the table, the minimum delay that can be achieved differs only with 30 ps (less than 1%) for the 64-bit case. However, the difference in energy-per-operation is more than 4 pJ (3%). Taking energy into consideration, a maximum fanout of 16 decoders per encoder appears to offer a good trade off between delay and energy.

Table 4: Delay and energy dependency on fanout for the MB multiplier using a Yeh recoder.

	Maximum Number of Decoders per Encoder		
Width	12	16	20
16	2.82 (ns) 9.24 (pJ)	2.79 (ns) 9.04 (pJ)	2.79 (ns) 9.04 (pJ)
32	3.69 (ns) 32.5 (pJ)	3.68 (ns) 32.6 (pJ)	3.65 (ns) 31.1 (pJ)
64	4.51 (ns) 135 (pJ)	4.52 (ns) 133 (pJ)	4.54 (ns) 137 (pJ)

6.2 Hsu Recoding

When we conduct the fanout investigation for the Hsu recoding scheme, we discover that the fanout for the x-inputs is reduced by adding minimum-sized inverters to all inputs of the encode circuits. The encode circuit is changed accordingly so that the logical function is preserved. To reduce the fanout of the encode circuits, the maximum number of decoders connected to a single encoder is limited, and to reduce the fanout of the y-inputs, we insert a number of inverters in parallel to buffer each input, so that each inverter drives a limited number of decoders.

The netlists of 32- and 64-bit multipliers with various fanout configurations, were taken through synthesis, placement, and route. The result for the 32-bit and the 64-bit multiplier implementations is given in Table 5 and Table 6, respectively. The choice of fanout configuration is not as straightforward as for the Yeh recoding scheme. For the 32-bit implementations we see that the configuration with 12 decoders per encoder and 10 decoders per inverter for the *y*-input yields the fastest implementation. However, this recoding configuration dissipates 9% more energy-per-operation than the implementation with the lowest energy-per-operation. Now, since the configuration with the lowest energy-per-operation also happens to be the second fastest, this is the most efficient configuration of all. When we consider also the 64-bit implementations, it is confirmed that 16 decoders per encoder and 10 decoders per inverter for the *y*-input is a good choice for configuration.

Table 5: Delay and energy dependency on fanout for 32-bit modified-Booth multipliers using a Hsu recoder. A row shows the maximum number of decoders per inverter for the *y*-inputs.

	Maximum Number of Decoders per Encoder		
	12	16	20
6	3.80 (ns) 39.1 (pJ)	3.84 (ns) 38.9 (pJ)	3.83 (ns) 39.3 (pJ)
10	3.84 (ns) 39.8 (pJ)	3.79 (ns) 37.1 (pJ)	3.95 (ns) 38.9 (pJ)
14	3.70 (ns) 40.4 (pJ)	3.80 (ns) 38.5 (pJ)	3.83 (ns) 38.9 (pJ)

Table 6: Delay and energy dependency on fanout for 64-bit modified-Booth multipliers using a Hsu recoder. A row shows the maximum number of decoders per inverter for the *y*-inputs.

	Maximum Number of Decoders per Encoder		
	12	16	20
6	4.84 (ns) 166 (pJ)	4.94 (ns) 156 (pJ)	4.83 (ns) 152 (pJ)
10	4.79 (ns) 166 (pJ)	4.80 (ns) 153 (pJ)	5.21 (ns) 158 (pJ)
14	4.70 (ns) 175 (pJ)	4.92 (ns) 163 (pJ)	4.85 (ns) 168 (pJ)

6.3 Recoding Scheme Comparison

Already from the results of Tables 4, 5, and 6 it is clear that the Yeh recoding scheme outperforms the recoding scheme of Hsu. This is confirmed by Fig. 8 that shows the delay and power for 16-, 32-, 48-, and 64-bit multipliers with the two different recoding schemes. Fig. 8 shows the power dissipation for different delay constraints. The leftmost point for each graph in the figure depicts the shortest possible delay that is achieved using our evaluation setup. The second, third and fourth point is when the timing constraint is relaxed with 100 ps, 300 ps, and 600 ps, respectively, relative the shortest possible delay. From the figure it is clear that the Yeh recoding scheme is the best scheme both in terms of delay and power.

We believe that the Hsu recoding scheme somewhat suffers from the fact that our investigation is based on tristated inverters, rather than transmission gates. At the same time we doubt that the improvement gained from using transmission-gate cells would be so significant that the Hsu recoding scheme would outperform the Yeh scheme.

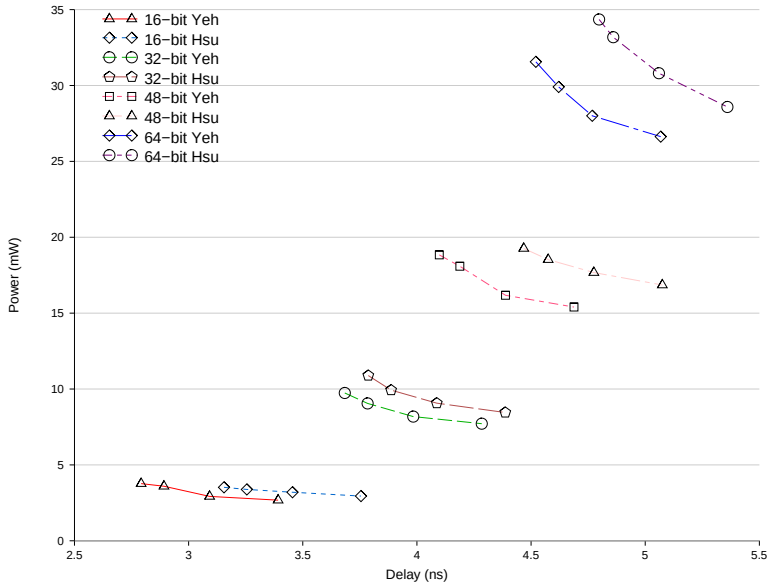


Figure 8: Delay and power for modified-Booth multipliers of various sizes, using two different recoding schemes.

7 Baugh-Wooley Multiplier Evaluation

The partial-product generation in the Baugh-Wooley (BW) implementation is much simpler than the one used in the modified-Booth multipliers. This difference in complexity becomes apparent also when we deal with fanout. In the BW multiplier each input drives N 2-input AND-gates for an N-bit multiplier. The effort to drive the AND-gates can simply be shared between a number of inverters that connect to the multiplier primary inputs. Table 7 shows the result of varying the maximum number of AND-gates connected to an inverter from 6 to 14 gates. The table shows that connecting 10 AND-gates to each inverter yields both a fast and energy-efficient solution.

Table 7: Delay (ns) and energy (pJ) dependency on fanout for the Baugh-Wooley multiplier.

	Maximum Number of Decoders per Inverter		
Width	6	10	14
16	2.91 (ns) 6.47 (pJ)	2.94 (ns) 6.27 (pJ)	2.94 (ns) 6.07 (pJ)
32	3.75 (ns) 24.5 (pJ)	3.63 (ns) 25.1 (pJ)	3.64 (ns) 23.7 (pJ)
64	4.65 (ns) 95.1 (pJ)	4.53 (ns) 95.4 (pJ)	4.68 (ns) 94.5 (pJ)

8 Comparison of Baugh-Wooley and Modified-Booth Multipliers

We begin the multiplier comparison of Baugh-Wooley (BW) and modified-Booth (MB), using the Yeh recoding scheme, by considering their efficiency in terms of delay and power[3]. Fig. 9 shows the delay and power for 16-, 32-, 48-, and 64-bit multipliers, using *i)* a timing constraint that achieves the fastest implementation and *ii)* three different relaxed timing constraints: 100 ps, 300 ps, and 600 ps, respectively, slower than the fastest timing obtained. The figure shows that the MB implementation can be up to 150 ps faster than the BW implementation. However, for 32 bits the BW implementation outperforms that of the MB. In contrast to the timing, the power dissipation is consistently and significantly lower for the BW implementations. The power dissipation of the MB implementations is 25–40% higher than that of a BW multiplier of the same size, which operates at the same speed.

[3]The power is given for a frequency that corresponds to the inverse of the delay.

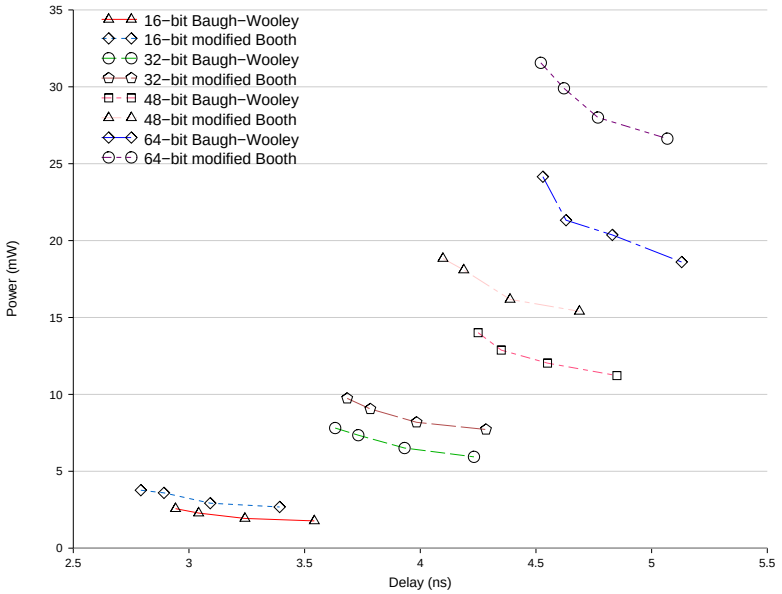


Figure 9: Delay and corresponding power dissipation for 16-, 32-, 48-, and 64-bit instances of modified-Booth and Baugh-Wooley multipliers.

A useful metric when considering both delay and power is the energy that is required to complete a single multiplication operation. Since we are now dealing with single-cycle multipliers, the energy-per-operation is obtained as the product of power and delay for each point in the graph of Fig. 9. Fig. 10 shows the result of such a calculation for sizes from 8 to 64 bits: Among the four timing constraints, only the smallest energy-per-operation obtained for each size is plotted. The lowest energy-per-operation is commonly achieved at a timing that is 100 ps to 300 ps slower than the fastest possible. From the graph it is clear that the BW multiplier is a much more efficient implementation in terms of energy.

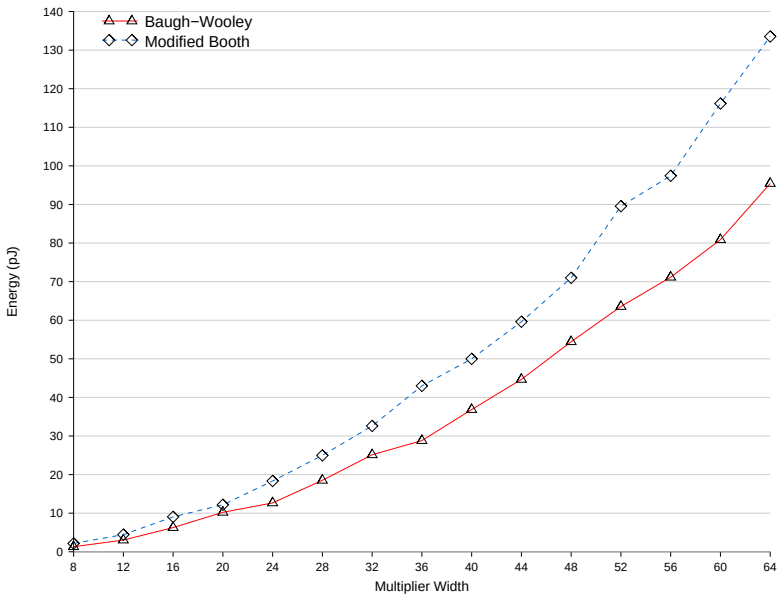


Figure 10: Energy-per-operation for modified-Booth and Baugh-Wooley multipliers.

For some embedded systems, speed can be as crucial as power dissipation. To compare the shortest delay that can be achieved, Fig. 11 shows the result for sizes of 8 to 64 bits. The graph shows that the BW multiplier can match the performance of the MB multiplier for sizes of up to 44 bits and even for larger multipliers the difference is no more than 150 ps, i.e. less than 4%.

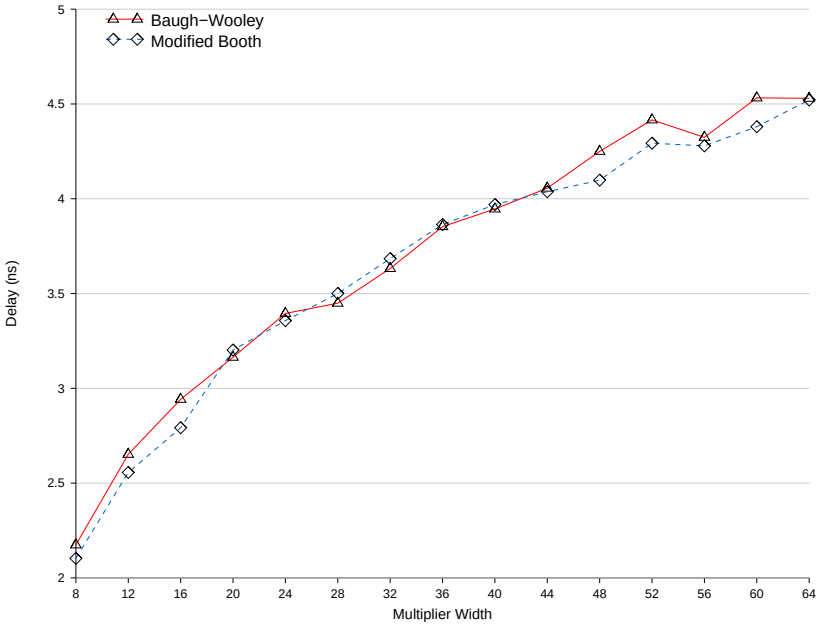


Figure 11: The shortest delay for modified-Booth and Baugh-Wooley multipliers.

If we consider area, which is an important factor for efficient multiplier implementations, the BW multiplier also in this respect outperforms the MB multiplier. Fig. 12 shows the area of the different multiplier implementations and for all sizes the BW implementation consistently is the smallest. The BW implementation is about 20% smaller than that of a MB implementation of the same operand bit-width.

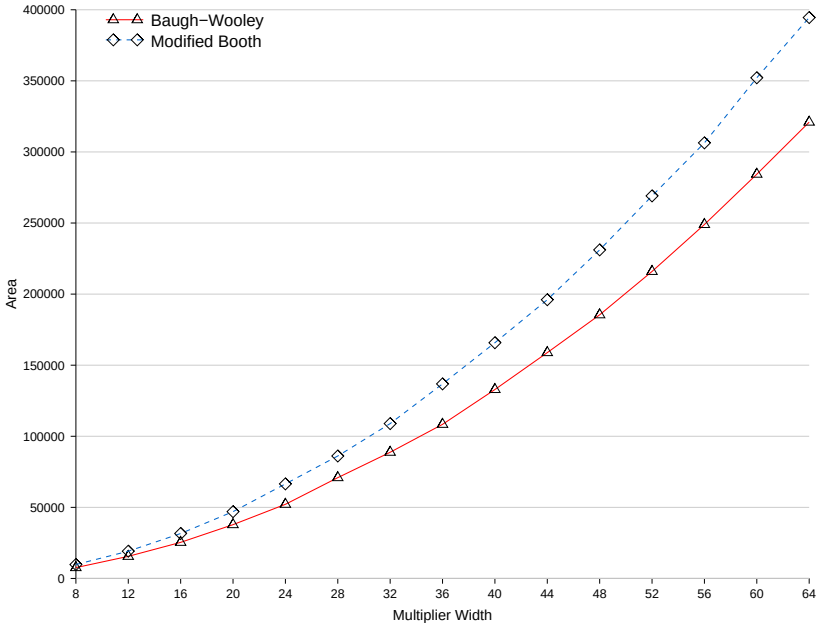


Figure 12: The area (μm^2) for modified-Booth and Baugh-Wooley multipliers.

8.1 Dissecting the Timing

When we take a closer look at the timing contribution of *i)* the partial-product generation, *ii)* the reduction tree, and *iii)* the final adder to the total delay for MB and BW multipliers, we see that the delay that is gained by having a smaller reduction tree for the MB implementation is offset by the delay of the MB recoding circuitry. Tables 8 and 9 show that the partial-product generation of the MB multipliers are about two times as slow as that of generation in the BW multipliers. This comes as no surprise, since for BW there is only one 2-input AND-gate in the critical path, while for the MB multiplier the critical path goes through the encoder followed by the decoder. Taking fanout into consideration, the primary inputs of a BW multiplier need to drive only N minimum-sized 2-input AND-gates, while for the MB multiplier the x primary inputs first have to drive $N/2$ encoders, which in turn drive N decoders.

Table 8: The delay for the partial-product generation, the reduction tree, and the final adder for two 32-bit multipliers.

	Baugh-Wooley		Modified Booth	
	Increment	Total	Increment	Total
Partial Product	0.459 (ns)	0.459 (ns)	0.953 (ns)	0.953 (ns)
Reduction Tree	2.092 (ns)	2.551 (ns)	1.679 (ns)	2.632 (ns)
Final Adder	1.081 (ns)	3.632 (ns)	1.052 (ns)	3.684 (ns)

Table 9: The delay for the partial-product generation, the reduction tree, and the final adder for two 48-bit multipliers.

	Baugh-Wooley		Modified Booth	
	Increment	Total	Increment	Total
Partial Product	0.592 (ns)	0.592 (ns)	1.101 (ns)	1.101 (ns)
Reduction Tree	2.439 (ns)	3.031 (ns)	1.770 (ns)	2.871 (ns)
Final Adder	1.219 (ns)	4.250 (ns)	1.265 (ns)	4.136 (ns)

9 Implementation Aspects of the Reduction Tree

The synthesis of the reduction trees used for the results in the previous sections was limited to minimum-sized full-adder (3:2) cells. It would be possible to achieve higher speed by increasing the drive strength of the reduction tree's gates or by using larger (4:2, 7:3, or 9:2) counter cells. This would mainly favor the Baugh-Wooley (BW) implementation since this, in comparison to the modified-Booth (MB) implementation, has a larger critical-path portion inside the reduction circuit. A faster reduction tree circuit, thus, would have a proportionally larger impact on the total delay of the BW implementation. On the other hand, an increased drive strength of the reduction tree's gates would have a negative impact on the power dissipation. In this respect, a BW implementation would experience a relatively higher increase in power compared to an MB implementation. How the power would be affected by larger counters is more difficult to predict without knowing the power of the counter cell. If one counter cell would be less power hungry than the collection of full-adder cells that it replaces, the BW implementation would benefit from using counters.

By varying the full-adder cell drive strength in the reduction circuit, we found that even for the drive strength[4] that yields the highest speed, a BW implementation is still 20-30% more energy efficient than the MB implementation; the exact value of the gain depends on the operand bit-width of the multiplier. Fig.13 shows the delay and power for the full-adder drive strength that obtained the best timing. In a comparison with Fig. 9, we see that the delay is improved at a cost in power.

[4]The full-adder cell comes in five different drive strengths in our cell library.

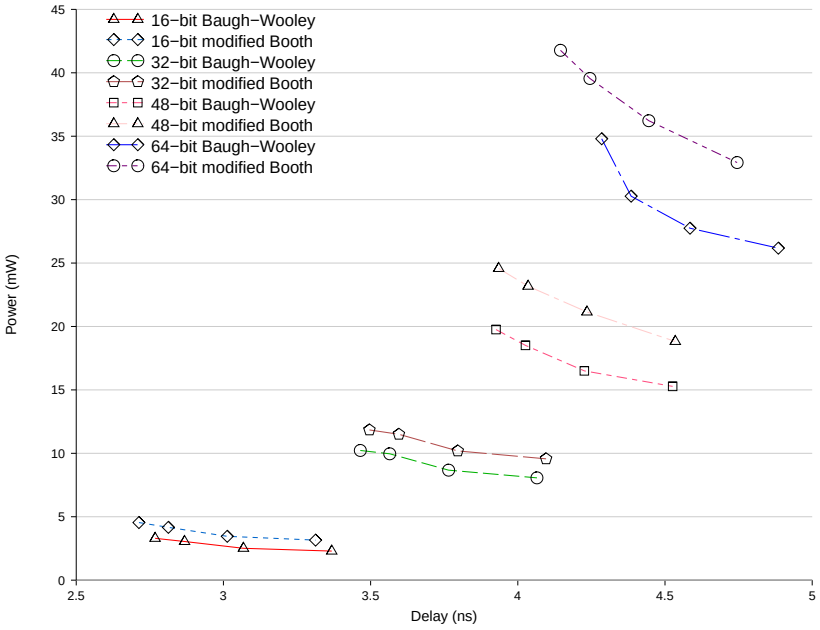


Figure 13: Delay and power for a 16-, 32-, 48- and 64-bit Baugh-Wooley and modified-Booth multipliers, for the drive strength of the full-adder cells in the reduction tree that obtains the lowest delay.

10 Implementation in a 65-nm Process Technology

We had limited access to a commercial 65-nm low-power process technology and used it to implement a 32-bit Baugh-Wooley (BW) and modified-Booth (MB) multiplier with standard threshold voltage cells. For the 65-nm design flow, Cadence Encounter [15] is used for synthesis, placement, and routing. For this process technology we were not able to do the same fanout investigations and buffer insertion, as described for the 130-nm process. The synthesis was restricted to use only full-adder cells. However, the placement tool is capable of doing re-synthesis and has in some cases implemented the full-adder functionality as a set of logic cells. Timing and power estimates are for the worst-case 125-degrees corner at 1.1 V. The timing is for lowest delay possible and the power dissipation was estimated using value change dump (VCD) data from simulations with 10,000 random input vectors, as for the 130-nm process.

Table 10: Delay, power, energy, and area for 32-bit Baugh-Wooley and modified-Booth multipliers in a 65-nm process.

	Delay (ns)	Power (mW)	Energy (pJ)	Area (μm^2)
Baugh	2.59 (100%)	23.4 (100%)	60.6 (100%)	48.1k (100%)
Booth	2.50 (97%)	37.5 (160%)	93.8 (155%)	52.1k (108%)

Table 11: Delay, power, energy, and area for 32-bit Baugh-Wooley and modified-Booth multipliers in a 130-nm process.

	Delay (ns)	Power (mW)	Energy (pJ)	Area (μm^2)
Baugh	3.63 (100%)	7.81 (100%)	28.4 (100%)	88.8k (100%)
Booth	3.68 (101%)	9.74 (125%)	35.8 (126%)	108.9k (123%)

The results for 32-bit BW and MB multipliers in the 65-nm process are shown in Table 10. The high power and energy dissipation of the MB multiplier does not justify the small performance advantage; 90 ps (3%) faster than the BW multiplier. Furthermore, the MB multiplier is also 8% larger in terms of area.

For reference, the corresponding data (those for shortest delay) for the 32-bit 130-nm implementations is shown in Table 11. The relationship between the BW and MB multiplier does not change much in terms of shortest delay. However, in relation to BW, the power and energy for the MB multiplier increase significantly with process scaling. Regarding area, the area penalty for the MB implementation is reduced from 23% in 130-nm to 8% in 65-nm.

Considering the results for the 65-nm and 130-nm it is clear that at least for a 32-bit multiplier, a MB implementation has higher power dissipation, larger area and only a negligible delay improvement compared to a BW implementation.

11 Partial-Product Generation and Final Adders

The usage of the modified-Booth (MB) algorithm makes the reduction tree flatter. This has the effect that the timing profile of the remaining pair of partial products, which are to be summed up by the final adder, are also flatter than compared to a Baugh-Wooley (BW) multiplier. There have been extensive research [16, 17] in adapting the final adder to the timing profile of the partial-product pair from the reduction tree in order to reduce power as well as area. As the timing profile becomes flatter, the room for exploiting timing slacks becomes more limited: A faster final adder is required for the less significant portion of the final result.

For the investigations conducted in this paper, a fast Kogge-Stone adder has been used as final adder. However, the final adder could have been optimized for each specific implementation. In this context, the BW implementations would have gained, in terms of power and area, more from an optimization of the final adder than the MB implementations.

12 Conclusion

The modified-Booth algorithm, which is commonly used today, makes multiplier design complex and a significant design effort is needed to obtain an efficient implementation. The design decision of using the modified-Booth algorithm is most likely based on the notion that a smaller reduction circuitry achieves a better implementation. This was once true, but the introduction of logarithmic-depth reduction trees, and particularly the regular structure of the HPM reduction tree, has made the size of the reduction circuit less of a concern when designing a multiplier. The logic depth through the HPM reduction tree differs by only one or two full adders for a modified-Booth and Baugh-Wooley implementation of the same operand bit-width. Considering that the critical path of a modified-Booth multiplier is located in its encoder and decoder, it is difficult to envision a modified-Booth implementation that can be much faster than a Baugh-Wooley implementation, regardless of the recoding scheme used. Taking power, energy per operation, and area into consideration, it is clear that the gain by reducing the reduction circuitry is lost in the recoding circuitry, making a modified-Booth implementation perform worse than a Baugh-Wooley implementation.

References

[1] T. K. Callaway and J. Earl E. Swartzlander, "Power-Delay Characteristics of CMOS Multipliers," in *Proceedings of the 13th IEEE Symposium on Computer Arithmetic*, June 1997, pp. 26–32.

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

[3] O.L.MacSorley, "High Speed Arithmetic in Binary Computers," in *Proceedings of the IRE*, vol. 49, no. 1, January 1961, pp. 67–97.

[4] J. Fadavi-Ardekani, "MxN Booth Encoded Multiplier Generator Using Optimized Wallace trees," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, no. 2, pp. 120–125, 1993.

[5] W.-C. Yeh and C.-W. Jen, "High-Speed Booth Encoded Parallel Multiplier Design," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 692–701, July 2000.

[6] 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*, May 2006.

[7] S. K. Hsu, S. K. Mathew, M. A. Anders, B. R. Zeydel, V. G. Oklobdzija, R. K. Krishnamurthy, and S. Y. Borkar, "A 110 GOPS/W 16-bit Multiplier and Reconfigurable PLA Loop in 90-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, pp. 256–264, January 2006.

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

[9] M. 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.

[10] M. Själander, "HMS Multiplier Generator," http://www.sjalander.com/research/multiplier, December 2008.

[11] P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Transactions on Computers*, vol. 22, no. 8, pp. 786–793, August 1973.

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

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

[14] *Cadence Encounter User Guide Version 4.1*.

[15] *Cadence Encounter User Guide Version 6.2*.

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

[17] J. Liu, S. Zhou, H. Zhu, and C.-K. Cheng, "An Algorithmic Approach for Generic Parallel Adders," in *IEEE International Conference on Computer Aided Design*, December 2003, pp. 734–740.