International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347 - 6435 (Online)), Volume 2, Special Issue 3, February 2014)

International Conference on Trends in Mechanical, Aeronautical, Computer, Civil, Electrical and Electronics Engineering (ICMACE14)

Delay Optimised 16 Bit Twin Precision Baugh Wooley Multiplier

Vivek. V. Babu [1], S. Mary Vijaya Lense [2]

[1] *II ME-VLSI DESIGN & The Rajaas Engineering College Vadakkangulam, Tirunelveli*
[2]*Assistant Professor & Department of Electronics and Communication Engineering The Rajaas Engineering College*

[1]vivek.vbabu@gmail.com
[2]marylense@yahoo.com

***Abstract*- Multiplication is indeed the most crucial operation in digital signal processing (DSP). Its implementation requires large hardware resources and significantly affects the size, performance, and power consumption of a DSP system. Technique for 16 bit integer multiplication is implemented in this project .Baugh-Wooley algorithm is an algorithm used for the multiplication ,delay and power dissipation can't be reduced further in this type. Twin precision technique is noteworthy for its low power dissipation. Multiplier is adapted to bitwidth of the operands to be computed to obtain the reduced power dissipation. The technique also results in an increased computational throughput, by allowing several narrow-width operations to be computed in parallel. Using Twin-precision technique with Baugh-Wooley algorithm, we achieve significant optimized delay and good power reduction.The project describe how to apply the twin-precision technique also to signed multiplier schemes, such as Baugh Wooley . It is shown that the twin-precision delay penalty is small (5% to 10% and that a significant reduction in power dissipation (40% to 70% can be achieved,when operating on narrow-width operands.**

***Index term*- Baugh Wooley algorithm,Twin Precision technique**

I. Introduction

Multiplication is a complex arithmetic operation, which is reflected in its relatively high signal propagation delay, high power dissipation, and large area requirement. When choosing a multiplier for a digital system, the bit width of the multiplier is required to be at least as wide as the largest operand of the applications that are to be executed on that digital system. The bit width of the multiplier is, therefore, often much larger than the data represented inside the operands, which leads to unnecessarily high power dissipation and unnecessary long delay.

This resource waste could partially be remedied by having several multipliers, each with a specific bit width, and use the particular multiplier with the smallest bit width that is large enough to accommodate the current multiplication. Such a scheme would assure that a multiplication would be computed on a multiplier that has been optimized in terms of power and delay for that specific bit width. However, using several multipliers with different bit widths would not be an efficient solution, mainly because of the huge area overhead. It has been shown in many studies that more than 50% of the instructions are instructions where both operands are less than or equal to 16 bits. Such operations are 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. It is shown that the power reduction of an operand-guarded integer unit was 54% to 58%, which accounts for a total power reduction of 5–6% for an entire data path.

Narrow-width operands have also been used to increase instruction throughput, by computing several narrow-width operations in parallel on a full-width data path. It is showed a 7% speedup for a simple 4-bit ALU, which excluded the multiplier, in parallel with four simple 16-bit ALUs that share a 64-bit routing.

II. Baugh Wooley Algorithm

The Baugh Wooley algorithm is a relative straightforward way of performing signed multiplications .illustrates the algorithm for an 8-bit case, where the partial-product array has been reorganized according to the scheme of Hatamian.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347 - 6435 (Online)), Volume 2, Special Issue 3, February 2014)

International Conference on Trends in Mechanical, Aeronautical, Computer, Civil, Electrical and Electronics Engineering (ICMACE14)

The creation of the reorganized partial-product array comprises three steps: i) the most significant partial product of the first N-1 rows and the last row of partial products except the most significant have to be negated, ii) a constant one isadded to the th column, iii) the most significant bit (MSB) of the final result is negated.Multiplication involves 2 basic operations: the generation of the partial product and their accumulation .Therefore, there are Possible ways to speed up the multiplication: reduces the complexity, and as a result reduces the time needed to accumulate the partial products.Both solutions can be applied simultaneously.

Baugh-Wooley Twos Compliment Signed Multiplier:Twos Compliments is the most popular method in representing signed integers in Computer sciences.It is also an operation of negation(Converting positive to negative numbers or vice versa) in computers which represent negative numbers using twos compliments. Its use is so wide today because it does not require the addition and subtraction circuitry to examine the signs of the operands to determine whether to add or subtract.Twos compliment and one's compliment representations are commonly used since arithmetic units are simpler to design. Baugh-Wooley Multiplier is used for both unsigned and signed number multiplication. Signed Number operands which are represented in 2s complemented form. Partial Products are adjusted such that negative sign move to last step, which in turn maximize the regularity of the multiplication array. Baugh-Wooley Multiplier operates on signed operands with 2s complement representation to make sure that the signs of all partial products are positive. Power consumption in Baugh-Wooley multipliers is minimum compared to other conventional multiplier units. So it clears that the signed binary multiplication through Baugh- Wooley multiplication is suited for large multiplier implementation. The improvements in constraint can be used to make Baugh-Wooley multiplier more efficient .The fan-out of the multiplier architectures are also given which directly gives the possibility of the multiplier to form large circuits. This can be extended tothe pipelined multiplier architecture also to verify the parameters. Latency and speed are the important factors with pipelining under consideration.The synthesis results of 4-bitpipelined multipliers . The pipeline constraint increases the speedof the multiplier considerably with an increase in power consumption.

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

$$P_{ij}=y_i x_j$$

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. 2.2

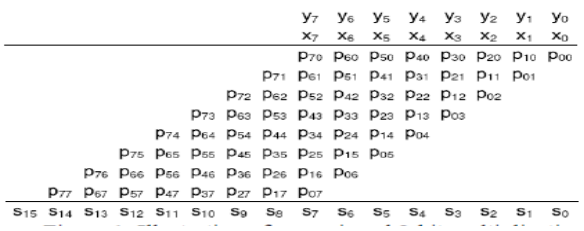
y7 y6 y5 y4 y3 y2 y1 y0
x7 x6 x5 x4 x3 x2 x1 x0
p70 p60 p50 p40 p30 p20 p10 p00
p71 p61 p51 p41 p31 p21 p11 p01
p72 p62 p52 p42 p32 p22 p12 p02
p73 p63 p53 p43 p33 p23 p13 p03
p74 p64 p54 p44 p34 p24 p14 p04
p75 p65 p55 p45 p35 p25 p15 p05
p76 p66 p56 p46 p36 p26 p16 p06
p77 p67 p57 p47 p37 p27 p17 p07
s15 s14 s13 s12 s11 s10 s9 s8 s7 s6 s5 s4 s3 s2 s1 s0

Fig.2.2 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 2. 3

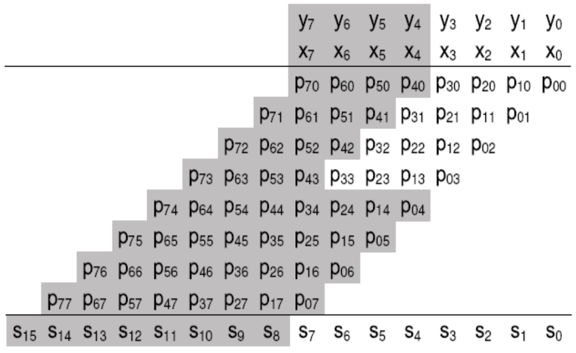
y7 y6 y5 y4 y3 y2 y1 y0
x7 x6 x5 x4 x3 x2 x1 x0
p70 p60 p50 p40 p30 p20 p10 p00
p71 p61 p51 p41 p31 p21 p11 p01
p72 p62 p52 p42 p32 p22 p12 p02
p73 p63 p53 p43 p33 p23 p13 p03
p74 p64 p54 p44 p34 p24 p14 p04
p75 p65 p55 p45 p35 p25 p15 p05
p76 p66 p56 p46 p36 p26 p16 p06
p77 p67 p57 p47 p37 p27 p17 p07
s15 s14 s13 s12 s11 s10 s9 s8 s7 s6 s5 s4 s3 s2 s1 s0

Fig.2.3 8 bit multiplication precision operand 4 bit

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347 - 6435 (Online)), Volume 2, Special Issue 3, February 2014)

International Conference on Trends in Mechanical, Aeronautical, Computer, Civil, Electrical and Electronics Engineering (ICMACE14)

Fig. 2.2 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. Figure 2.2 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. Since partial products of the same column are summed together, it would not be wise to use any of the partial products that are in the same column as the multiplication that is already computed. Looking closer at the 4-bit multiplication marked in white in Fig. 2,2 one can also observe that the column at position S7 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 S7. Altogether this makes only the partial products in the most significant part of the partial-product array available for a second multiplication. In order to be able to use the partial products in the most significant part, there has to be a way of setting their values. For this we can use the most significant bits of the operands, since these are not carrying any useful information. 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. 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 (NFULL) of the multiplication, 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 NLSP , shown in white, and the multiplication in the Most Significant part (MSP) with size MSP , shown in black.

$$NFULL = NLSP + NMSP$$

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.

$$N_{FULL} \geq \sum_{i=1}^{K} Ni$$

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.

III. Twin Precision Technique

Initially we present the twin-precision technique using an illustration of unsigned binary multiplication.In an unsigned binary multiplication each bit of one of the operands, called the multiplier, is multiplied with the second operand, called multiplicand. 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.2.12 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 f the partial-product array and the most significant bits of the final result will only consist of zeros. An illustration of an 8-bit multiplication, where the precision of the operands is four bits, is shown in Fig.2.12 Fig.2.13 shows that large parts of the partial-product array only consist of zeros and are, thus, not contributing any useful information to the final result. What if these partial products could be utilized for a second, concurrent multiplication. Since partial products of the same column are summed together, it would not be wise to use any of the partial products that are in the same column as the multiplication that is already computed. Looking closer at the 4-bit multiplication marked in white in Fig.2.11, one can also observe that the column at position S7 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 S7.

In order to be able to use the partial products in the most significant part, there has to be away of setting their values. For this we can use the most significant bits of the operands, since these are not carrying any useful information. If we are only looking at the upper half of the operands, the partial products generated from these bits are the ones shown in black in Fig.2.12..

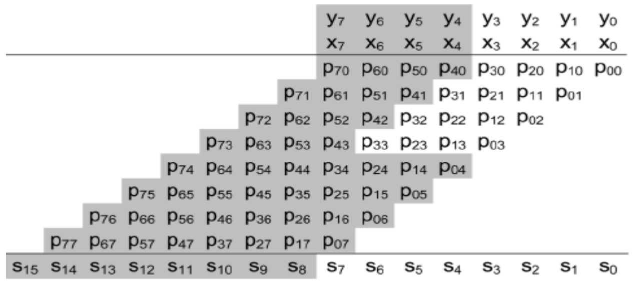
y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0
x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
p_{70} p_{60} p_{50} p_{40} p_{30} p_{20} p_{10} p_{00}
p_{71} p_{61} p_{51} p_{41} p_{31} p_{21} p_{11} p_{01}
p_{72} p_{62} p_{52} p_{42} p_{32} p_{22} p_{12} p_{02}
p_{73} p_{63} p_{53} p_{43} p_{33} p_{23} p_{13} p_{03}
p_{74} p_{64} p_{54} p_{44} p_{34} p_{24} p_{14} p_{04}
p_{75} p_{65} p_{55} p_{45} p_{35} p_{25} p_{15} p_{05}
p_{76} p_{66} p_{56} p_{46} p_{36} p_{26} p_{16} p_{06}
p_{77} p_{67} p_{57} p_{47} p_{37} p_{27} p_{17} p_{07}
s_{15} s_{14} s_{13} s_{12} s_{11} s_{10} s_9 s_8 s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0

Fig 3.2 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.

By setting the other partial products to zero, it is possible to perform two multiplications within the same partial-product array, without changing the way the summation of the partial-product array is done. How the partial products, shown in gray, can be set to zero will be presented in the implementation section later on. Assume, for now, that there is a way of setting unwanted partial products to zero: Now it suddenly becomes possible to partition the multiplier into two smaller multipliers that can compute multiplications in parallel. In the above illustrations the two smaller multiplications have been chosen such that they are of equal size. This is not necessary for the technique to work. Any size of the two smaller multiplications can be chosen, as long as the precision of the two smaller multiplications together are equal or smaller than the full precisionn (N)of the multiplication 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 Nlsp, shown in white, and the multiplication in the Most Significant Part (MSP) with size , shown in black.

It is functionally possible to partition the multiplier into even more multiplications. For example, it would be possible to partition a 64-bit multiplier into four 16-bit multiplications.

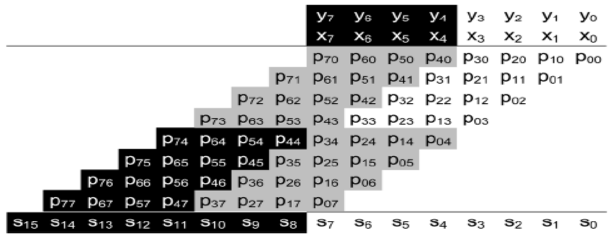
y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0
x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
p_{70} p_{60} p_{50} p_{40} p_{30} p_{20} p_{10} p_{00}
p_{71} p_{61} p_{51} p_{41} p_{31} p_{21} p_{11} p_{01}
p_{72} p_{62} p_{52} p_{42} p_{32} p_{22} p_{12} p_{02}
p_{73} p_{63} p_{53} p_{43} p_{33} p_{23} p_{13} p_{03}
p_{74} p_{64} p_{54} p_{44} p_{34} p_{24} p_{14} p_{04}
p_{75} p_{65} p_{55} p_{45} p_{35} p_{25} p_{15} p_{05}
p_{76} p_{66} p_{56} p_{46} p_{36} p_{26} p_{16} p_{06}
p_{77} p_{67} p_{57} p_{47} p_{37} p_{27} p_{17} p_{07}
s_{15} s_{14} s_{13} s_{12} s_{11} s_{10} s_9 s_8 s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0

Fig.3.1. Illustration of an unsigned 8-bit multiplication, where a 4-bit multiplicationIllustration 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.,

Given a number of low-precision multiplications, their total size needs to be smaller or equal to the full-precision multiplicationThe basic operation of generating a partial product is that of a 1-bit multiplication using a two-input AND gate, where one of the input signals is one bit of the multiplier and the second input signal is one bit of the multiplicand. The summation of the partial products can be done in many different ways, but for this investigation we are only interested in parallel multipliers that are based on 3:2 full adders.2 For this first implementation an array of adders will be used because of its close resemblance to the previously used illustration of a multiplication; previous section we assumed that there is a way of setting unwanted partial products to zero. This is easily accomplished by changing the two-input AND gate to a three-input AND gate, where the extra input can be used for a control signal. Of course, only the AND gates of the partial products that have to be set to zero need to be changed to a three-input version. During normal operation when a full-precision multiplication is executed the control signal is set to high, thus all partial products are generated as normal and the array of adders will sum them together and create the final result.

3.1Twin Precision On Baugh Wooley Multiplication

It is not as easy to deploy the twin-precision technique onto a BWmultiplication as it is for the unsigned multiplication, where only parts of the partial products need to be set to zero.

To be able to compute two signed multiplications, it is necessary to make a more sophisticated modification of the partial-product array. Fig.2.13 illustrates an 8-bitBWmultiplication, in which two 4-bit multiplications have been depicted in white and black. When comparing the illustration of Fig.2.12 with that of Fig.2.13 one can see that the only modification needed to compute the4-bit multiplication in the MSP of the array is an extra sign bit 1 in column . For the 4-bit multiplication in the LSP of the array, there is a need for some more modifications. In the active partial-product array of the 4-bit LSP multiplication (shown in white), the most significant partial product of all rows, except the last, needs to be negated. For the last row it is the opposite, here all partial products, except the most significant, are negated. Also for this multiplication a sign bit 1 is needed, but this time in column .

Finally the MSB of the results needs to be negated to get the correct result of the two 4-bit multiplications. To allow for the full-precision multiplication of size to coexist with two multiplications of size in the same multiplier, it is necessary to modify the partial-product generation and the reduction tree. For the -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 -bit multiplication is to be computed and to low, when the full precision multiplication is to be computed. To compute the -bit multiplication in the LSP of the array, certain partial products need to be negated. This can easily be accomplished by changing the two-input AND gate that generates the partial product to a two-input NAND gate followed by an XOR gate. The second input of the XOR gate can then be used to invert the output of the NAND gate. When computing the -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 multiplication the same signal is set to high making the XOR work as an inverter.

When comparing the illustration of Fig.3.1.1 with that of Fig.3.1.2 one can see that the only modification needed to compute the4-bit multiplication in the MSP of the array is n extra sign bit 1 in column s1 . For the 4-bit multiplication in the LSP of the array, there is a need for some more modifications.

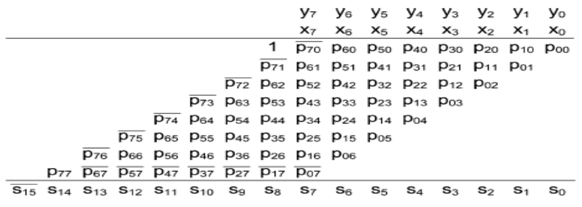
y7 y6 y5 y4 y3 y2 y1 y0
x7 x6 x5 x4 x3 x2 x1 x0
1 p70 p60 p50 p40 p30 p20 p10 p00
p71 p61 p51 p41 p31 p21 p11 p01
p72 p62 p52 p42 p32 p22 p12 p02
p73 p63 p53 p43 p33 p23 p13 p03
p74 p64 p54 p44 p34 p24 p14 p04
p75 p65 p55 p45 p35 p25 p15 p05
p76 p66 p56 p46 p36 p26 p16 p06
p77 p67 p57 p47 p37 p27 p17 p07
s15 s14 s13 s12 s11 s10 s9 s8 s7 s6 s5 s4 s3 s2 s1 s0

Fig.3.1.1 Illustration of a signed 8-bit multiplication, using the BaughWooley

In the active partial-product array of the 4-bit LSP multiplication (shown in white), the most significant partial product of all rows, except the last, needs to be negated. For the last row it is the opposite, here all partial products, except the most significant, are negated. Also for this multiplication a sign bit 1 is needed, but this time in column s4. Finally the MSB of the results needs to be negated to get the correct result of the two 4-bit multiplications.

To allow for the full-precision multiplication of size n to coexist with two multiplications of size 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 he array all that is needed is to add a control signal that can be set to high, when the n2/-bit multiplication is to be computed and to low, when the full precision multiplication is to be computed. To compute the n/2- 20 bit multiplication in the LSP of the array, certain partial products need to be negated. This can easily be accomplished by changing the two-input AND gate that generates the partial product to a two-input NAND gate followed by an XOR gate.

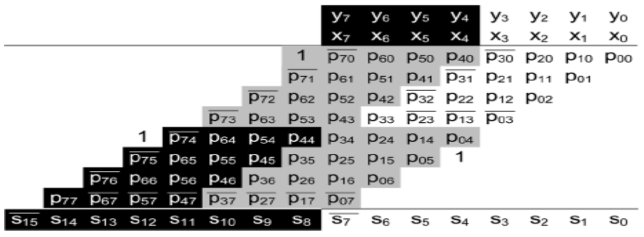
y7 y6 y5 y4 y3 y2 y1 y0
x7 x6 x5 x4 x3 x2 x1 x0
1 p70 p60 p50 p40 p30 p20 p10 p00
p71 p61 p51 p41 p31 p21 p11 p01
p72 p62 p52 p42 p32 p22 p12 p02
p73 p63 p53 p43 p33 p23 p13 p03
1 p74 p64 p54 p44 p34 p24 p14 p04
p75 p65 p55 p45 p35 p25 p15 p05 1
p76 p66 p56 p46 p36 p26 p16 p06
p77 p67 p57 p47 p37 p27 p17 p07
s15 s14 s13 s12 s11 s10 s9 s8 s7 s6 s5 s4 s3 s2 s1 s0

Fig:3.1.2 Illustration of a signed 8-bit multiplication, using the BaughWooley

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 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 three-input AND gates as for the unsigned case.

IV. RESULT

Delay optimized multiplier can be obtained with this technique. Higher order multiplication ie 16bit multiplication is possible in this Twin Precision Baugh Wooley Multiplier. Since the delay penalty and the power dissipation can be reduced the multipler can be made faster. The result of the comparison of the twin-precision implementations with their conventional counterparts is that a twin-precision implementation of BaughWooley performs equal in terms of delay for the 16- and 48-bit case and is only 160 ps slower for the 32-bit case. When we consider power, the twin-precision implementation dissipates 8conventional16-, 32-, and 48-bit BW implementation.

V. CONCLUSIONS

The presented twin-precision technique allows for flexible architectural solutions, where the variation in operand bitwidth that is common in most applications can be harnessed to decrease power dissipation and to increase throughput of multiplications. It turns out that the BaughWooley algorithm implemented on a HPM reduction tree is particularly suitable for a twin-precision implementation.

Due to the simplicity of the implementation, only minor modifications are needed to comply with the twin-precision technique. This makes for an efficient twin-precision implementation, capable of both signed and unsigned multiplications. Currently a lot of research is done on reconfigurable architectures, where the architecture can be adapted to the applications that are being executed. Some of these proposed architectures can adapt their arithmetic logic units to operate on different bitwidths, depending on the application .The twin-precision technique, which offers xibility at a low implementation overhead, makes it possible to efficiently deploy these flexible architectures.

REFERENCES

[1] Baugh.C.R and Wooley B.A. A Two"s Complement Par allel Array Multiplication Algorithm. IEEE Transactions on Computers, 22:1045", December 1973.–701, Jul. 2000.

[2] Benini.L, Micheli.G.D, Maci.Ai, E.Macii, Poncino.M, and Scars.Ri, "Glitching power minimization by selective gate freezes," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 8, no. 3, pp.287²97, June 2000.

[3] Eriksson.H. Efficient Implementation and Analysis of CMOS Arithmetic Circuits. PhD thesis, Chalmers University of Technology, 2003.

[4] Huang.Z and Ercegovac.M.D. Two-Dimensional Signal Gating for Low-Power Array Multiplier Design. In Proceedings of the IEEE International Symposium on Circuits and Systems pages II–IJ vol.1, 2002.

[5] LakshmiNarayanan.G and Venkataramani.B ,"Optimization Techniques for FPGA-Based Wave Pipelined DSP Blocks" IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.13. No 7. pp 783-792, July 2005.

[6] Oklobdzija.V.G, Villeger.D, and Liu .S.S,A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic approach IEEE Transactions on Computers, 45C):294˜6 march 1996.

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