Efficient Utilization of Time for Fast Multipliers Using Twin Precision Technique in DSP

M.Sahithi[1], Naseema Shaik[2], S. Dayasagar Chowdary[1], M. Jyothi[1], J.Poornima[1] and K. Rajasekhar[1]

[1] *M.Tech students, Department of ECE, K L University, Vijayawada, AP, India*
mannesahithi@gmail.com

[2] *Asst Prof. Department of ECE, K L University, Vijayawada, AP, India*
aseem.dishad@gmail.com

Abstract

We present the twin-precision technique for multipliers. The twin-precision technique can reduce the power dissipation by adapting a multiplier to the bit width of the operands being computed. The technique also enables an increased computational throughput, by allowing several narrow-width operations to be computed in parallel. We describe how to apply the twin-precision technique also to signed multiplier schemes, such as Baugh–Wooley and modified-Booth multipliers. By using this we can reduce power and we can do the multiplication in less amount of time.

Keywords: *Twin Precision technique, Baugh Wooley algorithm, Modified Booth algorithm.*

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.

The basic multiplication principle is two fold i.e. evaluation of partial products and accumulation of the shifted partial products. It is performed by the successive additions of the columns of the shifted partial product matrix. The 'multiplier' is successfully shifted and gates the appropriate bit of the 'multiplicand'. The delayed, gated instance of the multiplicand must all be in the same column of the shifted partial product matrix. They are then added to form the product bit for the particular form. Multiplication is therefore a multi operand operation. To extend the multiplication to both signed and unsigned numbers, a convenient number system would be the representation of numbers in two's complement format.

In this paper 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 bit-width 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.

In this paper we present mainly the comparisons of algorithms which are mainly used for multiplication. Here we present the array multiplier, Modified booth algorithm and Baugh Wooley algorithm. In this our main aim is to show Baugh Wooley is the efficient technique for getting multiplication in less amount of time. It also used to get high throughput with less power. We structured our code in VHDL.All these algorithms are designed by using Twin Precision [1] technique.

GENERAL MULTIPLICATION

Fast multipliers are essential parts of digital signal processing systems. The speed of multiply operation is of great importance in digital signal processing as well as in the general purpose processors today, especially since the media processing took off. In the past multiplication was generally implemented via a sequence of addition, subtraction, and shift operations. Multiplication can be considered as a series of repeated additions. The number to be added is the multiplicand, the number of times that it is added is the multiplier, and the result is the product. Each step of addition generates a partial product. In most computers, the operand usually contains the same number of bits. When the operands are interpreted as integers, the product is generally twice the length of operands in order to preserve the information content. This repeated addition method that is suggested by the arithmetic definition is slow that it is almost always replaced by an algorithm that makes use of positional representation. It is possible to decompose multipliers into two parts. The first part is dedicated to the generation of partial products, and the second one collects and adds them.

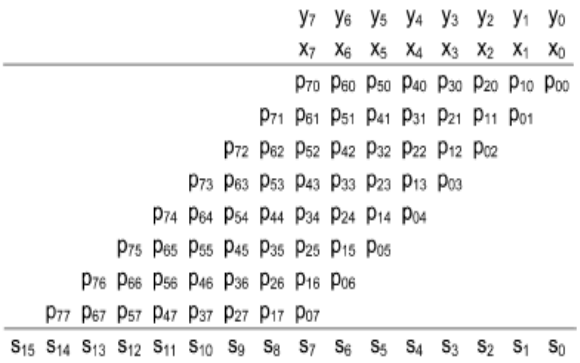
We present a twin-precision multiplier that in normal operation mode efficiently performs N-bit multiplications. For applications where the demand on precision is relaxed, the multiplier can perform N/2-bit multiplications while expending only a fraction of the energy of a conventional N-bit multiplier. For applications with high demands on throughput, the multiplier is capable of performing two independent N/2-bit multiplications in parallel.

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 bit-width, and let them share the same routing. 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 fan-out of the signals that drive the inputs of the multipliers. Higher fan-out 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. This can be done in several stages, creating a larger multiplier out of smaller for each extra 4:2 reduction stage. The desired bit-width 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 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.

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). 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.



$$\mathbf{p}ij = \mathbf{y}i\mathbf{x}j \quad \text{---------- (1)}$$

Figure 1: *Illustration of an unsigned 8-bit multiplication.*

IMPLEMENTATIONS AND ALGORITHMS

A. Array Multiplier

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 adders1. 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, 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 summations 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. 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.

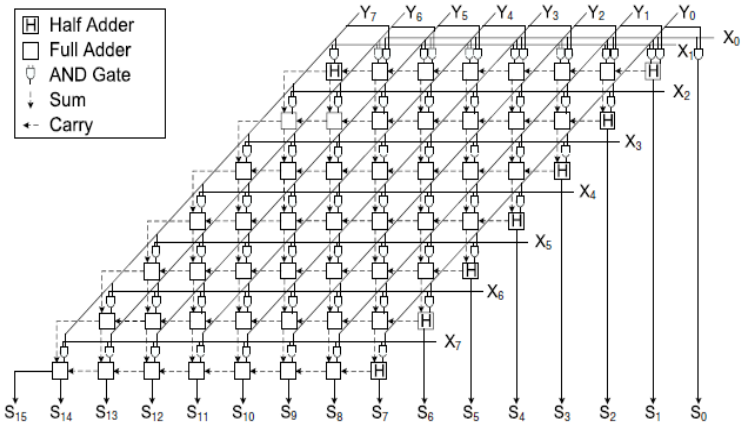


Figure .2: *Block diagram of an unsigned 8-bit array multiplier*

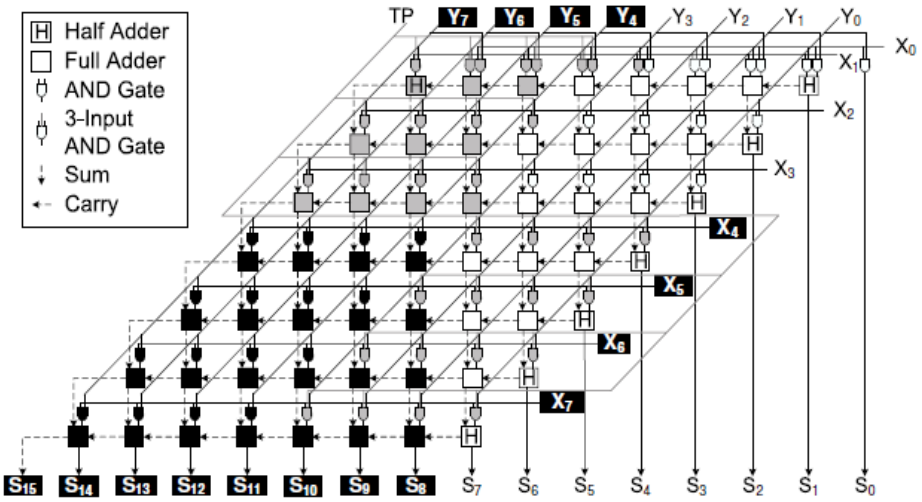


Figure.3: *Block diagram of an unsigned 8-bit twin-precision array multiplier.*

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.

The above figure.3 shows the array multiplier implementation using Twin precision technique. It shows the clear difference between

B. A Baugh Wooley Implementation Using Twin Precision

In this section a twin-precision multiplier based on the Baugh-Wooley [2] (BW) algorithm will be presented. The BW algorithm is a relative straightforward way of doing signed multiplications. Fig. 4 illustrates the algorithm for an 8-bit case, where the partial product array has been reorganized according the scheme of Hatamian [3] . The creation of the reorganized partial-product array comprises three steps:

i. *T*he 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.

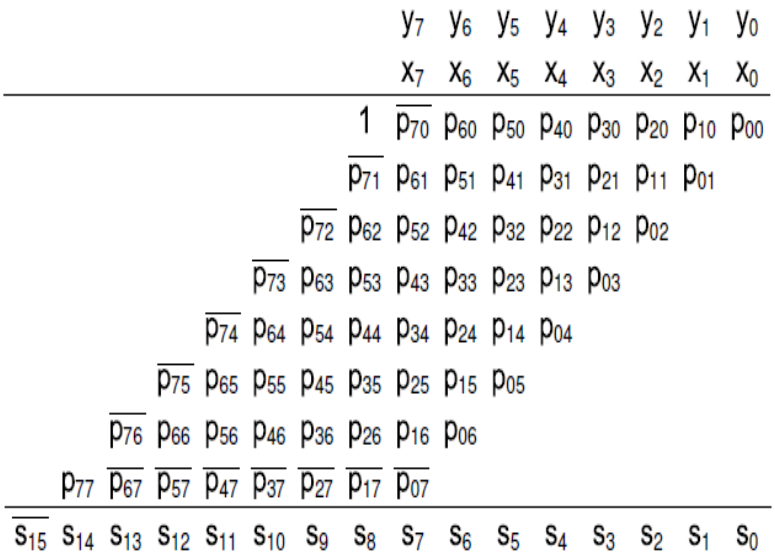


Figure.4: *Illustration of an 8-bit Baugh-Wooley multiplication.*

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. 4. with that of Fig. 5.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 S12. 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 S4. Finally the MSB of the result needs to be negated to get the correct result of the two 4-bit multiplications.

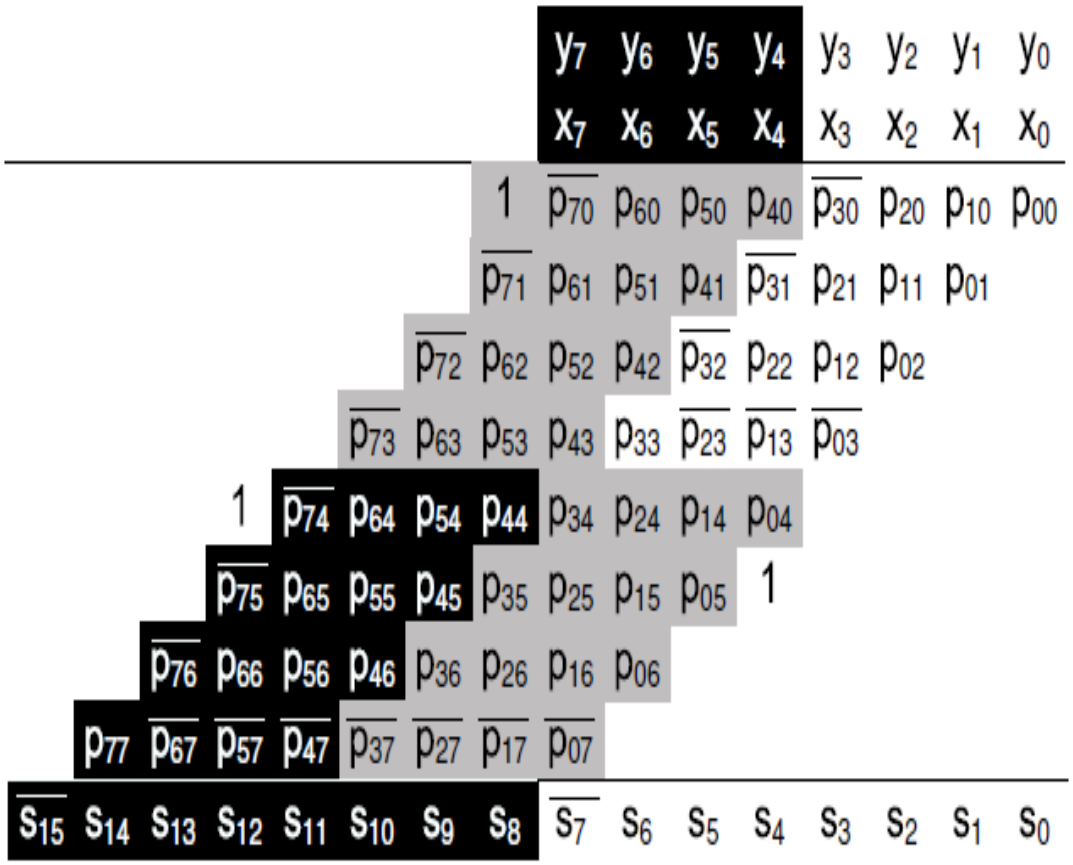


Figure.5. *Illustration of a signed 8-bit multiplication, using the Baugh-Wooley algorithm, where a 4-bit multiplication, shown in white, is computed in parallel with a second 4-bit multiplication, shown in black.*

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.

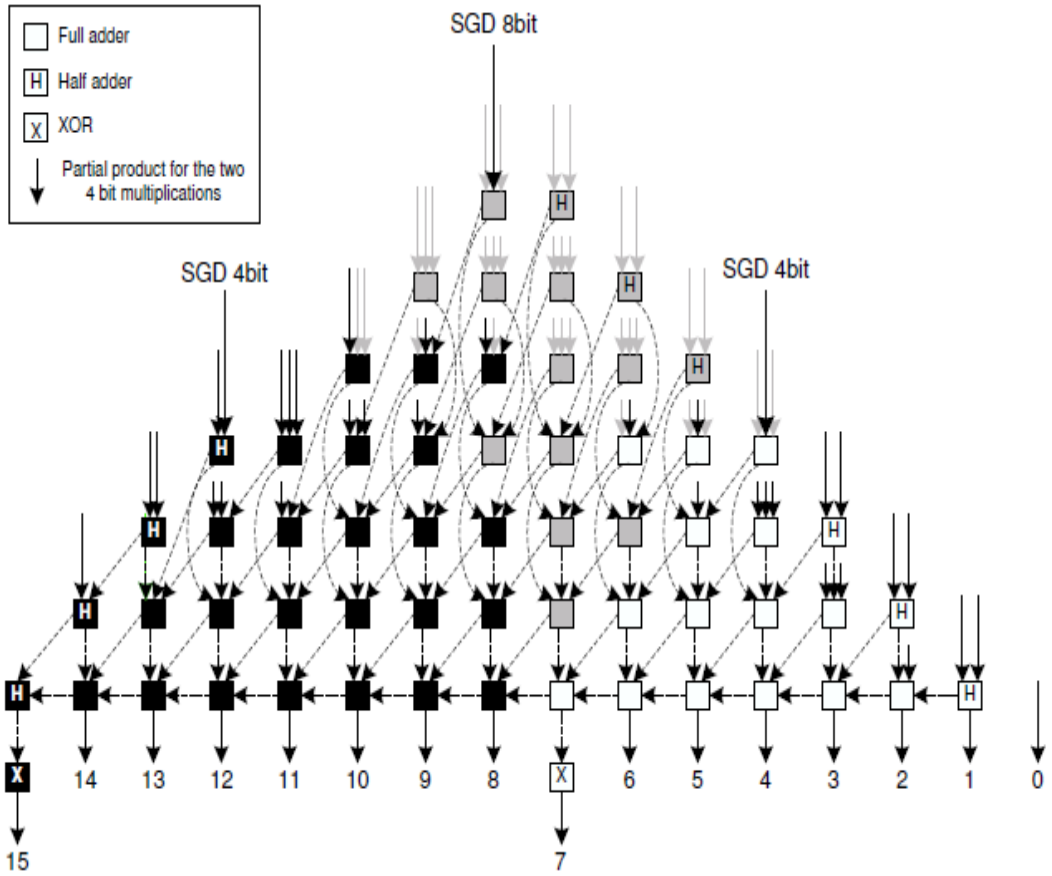


Figure.6. *Block diagram of a signed 8-bit multiplication, using the Baugh-Wooley algorithm, where a 4-bit multiplication, shown in white, is computed in parallel with a second 4-bit multiplication, shown in black.*

Fig. 6. 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. 3 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.

C. Modified Booth Algorithm

Modified Booth [4] (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.

The original-Booth algorithm is a way of coding the partial products generated during a S = x * 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 unrecorded 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 Modified Booth [5] algorithm 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.

A MB multiplier works internally with two's complement representation of the partial products, in order to be able to multiply the encoded {-2; -1g with the y operand. To avoid having to sign extend the rows of recoded partial products, the sign-extension prevention scheme presented by Fadavi- Ardekani [6] 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 [7] we draw on the idea of Yeh *et al., [*8] 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.

Implementing twin-precision together with the MB algorithm is not as straightforward as for the BW implementation. It is not possible to take the partial products from the full-precision MB multiplication and use only the partial products that are of interest for the 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. 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, 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 pLSBi, 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.

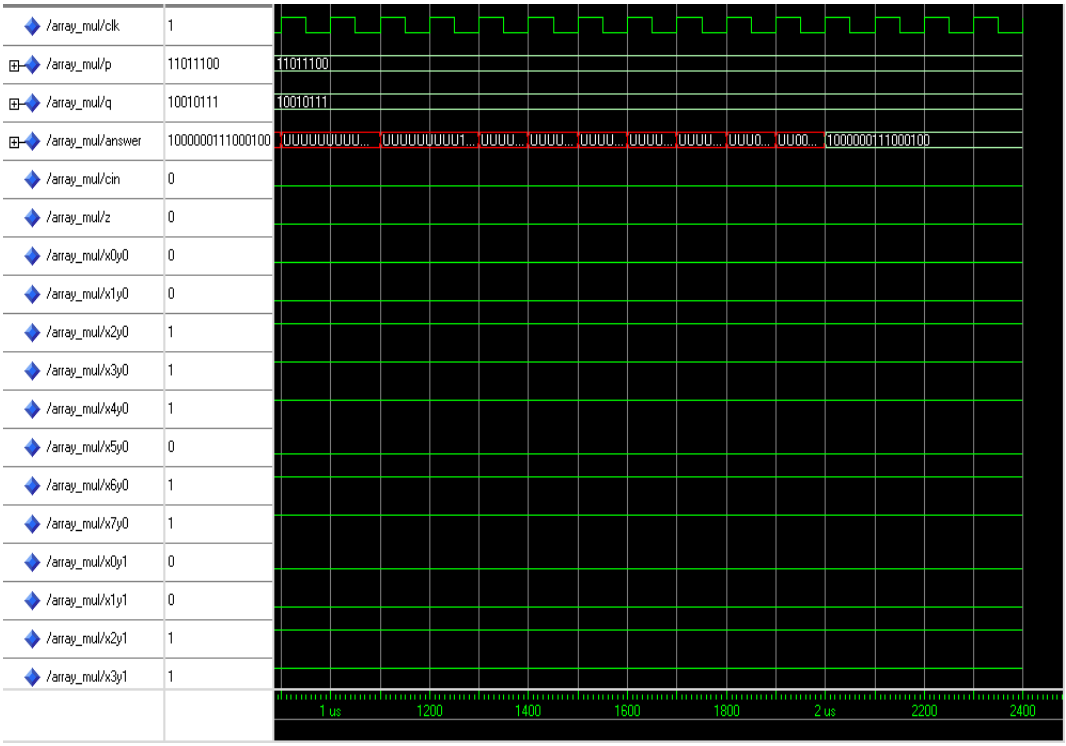


Figure.7.*Twin array Multiplier*

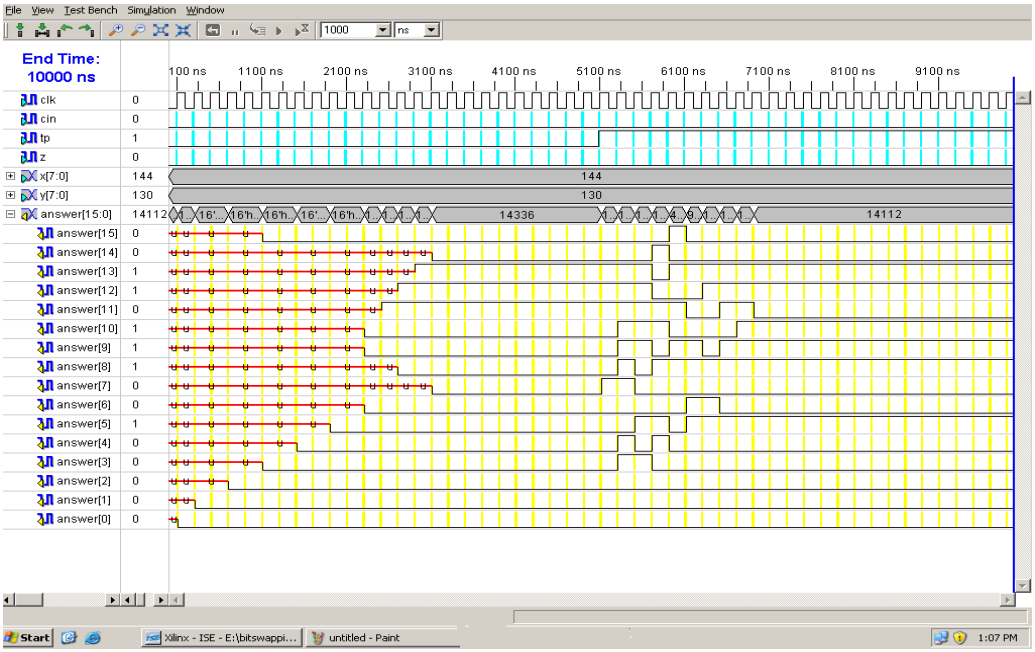


Figure.8. *Modified Booth Multiplier Using Twin Precision*

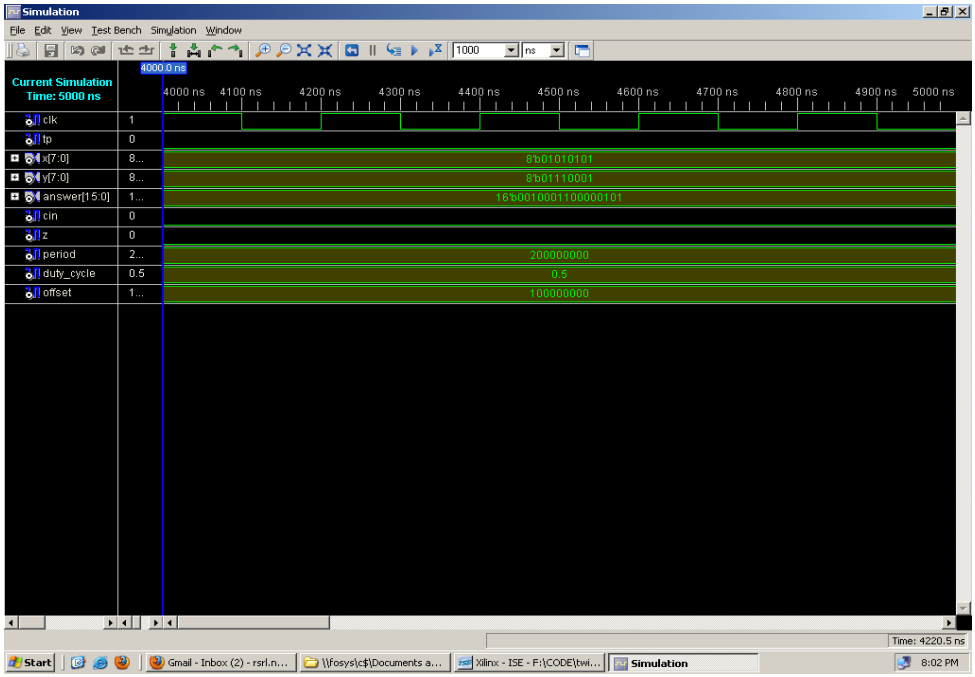


Figure 9. *Baugh Wooley algorithm using Twin precision technique*

All these are implemented using Xilinx [9] ISE Simulator.

	Baugh wooley	Modified Booth	Array multiplier
Efficiency	High	Moderate	Low
Speed	High	High	Low
Throughput	High	Less compared to Baugh Wooley	Low
Adders	Less in number	Less	less
Signed or Unsigned	Both	Signed	Unsigned
Complexity	Less	More compared to Baugh Wooley	More
Area Used	Less	Partially less	More

Table 1.*Comparisions of the algorithms used in multiplication.*

III. SIMULATION RESULTS

From the above table.1 we have seen that Baugh Wooley is more efficient and less complex when compared to other designs that we are used in this paper. One of the goals of the twin-precision technique is to keep the performance degradation of the multiplier's full-precision operation at a minimum. A clear trend is that a BW implementation is more power efficient than a MB implementation. However, a MB implementation can, in some cases, exhibit higher maximum speed. The MB twin

precision implementation has the poorest performance, both in terms of delay and power, of the four compared implementation choices. The twin-precision multiplier requires slightly more area than its conventional counterpart. However, the twin-precision implementation based on the BW algorithm is smaller than the commonly used conventional MB implementation.

CONCLUSION

From the above figures we have seen the multiplication using Array multiplier, Modified Booth and Baugh Wooley algorithms. The presented twin-precision technique allows for flexible architectural solutions, where the variation in operand bit width that is common in most applications can be harnessed to decrease power dissipation and to increase throughput of multiplications. 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 implementation of Baugh Wooley algorithm using Twin Precision Technique gives high throughput, highly efficient, more speed when compared to Array multiplier design and Modified Booth algorithms. So Baugh Wooley design using Twin Precision is less complex and easily designed and is more efficient.

REFERENCES

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

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

M. Hatamian, "A 70-MHz 8-bit □ 8-bit parallel pipelined Multiplier in 2.5-□m CMOS," *IEEE J. Solid-State Circuits*, vol. 21, no. 4, pp.505–513, Aug. 1986.

A. D. Booth, "A signed binary multiplication technique, Quarterly J. Mechan. Appl. Math" vol.4, no.2, pp.236240, 1951.

O. L. MacSorley, "High speed arithmetic in binary computers, Proc.Inst.RadioEng" vol.49, no.1, pp.67–97, Jan.1961.

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

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.

W.-C. Yeh and C.-W. Jen, "High-speed Booth encoded.

Parallel multiplier design," *IEEE Trans.Comput.*, vol.49, no. 7, spp. 692–701, Jul. 2000.

Xilinx, Inc. Xilinx Libraries Guide, 1999.