Asirvatham and Ramachandran *EURASIP Journal on Advances in Signal Processing* (2015) 2015:18
DOI 10.1186/s13634-015-0197-y

EURASIP Journal on Advances in Signal Processing
a SpringerOpen Journal

RESEARCH Open Access

An optimised twin precision multiplier for ASIC environment

Rosi Asirvatham* and Seshasayanan Ramachandran

Abstract

In this paper, we present the performance of twin precision technique in reduced computation modified booth (RCMB) multiplier to achieve double throughput, and an algorithm is proposed. Twin precision technique is the efficient way to obtain double throughput in the multipliers. We describe how to apply twin precision technique to RCMB multipliers. Implementation of twin precision in RCMB multiplier requires lesser changes to be made in partial product array for obtaining double throughput. Multiplexers usually do the signal selection for N and $N/2$ bit multiplication. In RCMB multiplier, $[N/2] + 1$ partial product are reduced to $N/2$ rows. Our idea of implementing twin precision technique to RCMB results in less utilisation of multiplexers of about $[N/2] + 3$ which gave a way for optimization in the twin precision (TP) multiplier. Thereby, we have achieved the drastic reduction in multiplexer utilisation of about 40% to 50% (for $N = 8$ to 128) compared to the existing twin precision modified booth multiplier. In our proposed optimised TP modified booth multiplier this reduction in multiplexers gave a way for overall reduction in area, power and delay. Lesser utilisation of multiplexer results in the area reduction of about 5% to 18%, delay of 5% to 20% and a considerable reduction in power of 8% to 32% were noticed in the proposed TP booth multiplier for N = 8 to 128. Our proposed optimised TP multiplier is implemented in FFT complex multiplication which is taken as an application case study and achieves better performance (area, delay and power) compare to prior TP multiplier. All our evaluation are made using cadence RTL compiler using TSMC 180 nm library.

Keywords: Twin precision; Multiplexer; Throughput; RCMB; FPGA; ASIC

1 Introduction

Multiplication is an influential arithmetic operation in processor and digital signal-processing application, and thus it plays a foremost role in digital computation. In multiplication, the processing delay is directly proportional to the critical path. In order to design an efficient multiplier, limits such as multiplier speed, power and area have to be thought thoroughly. In this paper, we narrowed our research work towards achieving double throughput in signed multipliers with lesser hardware complexity. Twin precision (TP) is the technique that can be exploited to obtain the dual output [1] in multipliers, and this technique was implemented for both signed and unsigned multipliers [2].

Achieving double throughput multipliers in application-specific integrated circuit (ASIC) environment is a challenging task where re-programmability is not possible like in field-programmable gate array (FPGA). Two technologies such as FPGA and ASIC have their own pros and cons [3]. One of the biggest advantages considered in FPGA is re-programmability. Modern FPGA has many variable precision embedded multipliers [4]. Though FPGA has variable precision embedded multipliers, blindly we cannot say it is more advantageous than ASIC multipliers. Because further optimization in FPGA embedded multipliers are not possible. In ASIC, re-programmability is not possible but increased throughput in multipliers is achieved efficiently by twin precision technique.

Initially, double throughput in very large scale integration (VLSI) multipliers is achieved by using several multipliers and at least two share the same route,

* Correspondence: arosyme@gmail.com
Department of Electronics and Communication Engineering, College of Engineering, Guindy, Chennai 600025, India

which is adopted in [5,6]. However, these methods have several disadvantages like increased multiplier area and high fan-out that increases the overall delay and area. This method uses multiplexers to connect the active multiplier to the output. We can say twin precision is a subset of subword parallelism (SWP) if the lower precision multiplications stop within two levels, i.e., if a 16 bit multiplier performs two 8-bit multiplication, then it is TP multiplication. SWP is capable of performing four 4-bit multiplications or two 8-bit multiplications in the same multiplier architecture, and we referred it as multiple SWP. A subword is a lower precision of data contained within a word. By exploiting SWP in signed multipliers, multiple lower precision multiplications can be performed. So unlike TP, the throughput wont stop within two. But the other case, we have to consider here is when multiple (above two) SWP is performed on modified booth multiplier many changes are to be made for obtaining MB algorithm in all levels of lower precision multiplication. While obtaining multiple (above two) SWP in signed multiplier which adopts modified booth algorithm, steps like sign extension and inversion of bits and carry suppression are to be made in all levels of lower precision operation. And multiplexers are needed at all lower precision multiplication to select appropriate partial products because sign extension and inversion of the most significant bits (MSB) in partial product rows will vary for multiple SWP and full precision multiplication. Though SWP offers more flexibility like multiple throughputs, their hardware complexity increases with an increase in multiplexer utilisation for implementing a signed multiplication algorithm like modified booth (MB). Much architecture based on SWP has been implemented in [7-9]. These architectures are specially designed for media processing.

In [2], double throughput is realised effectively by adopting TP technique in MB multiplier. MB algorithm [10] is a widely used signed algorithm since it has reduced partial product row. The possibility of combining N and $N/2$ bit (b) multiplication in the same N b tree multiplier is called as TP multiplier where N is the bit width of the multiplier. Here we can split the partial product bits of the N b multiplier in such a way that $N/2$ b multiplication can be performed in the least significant part (LSP) of the multiplier in parallel with another $N/2$ b multiplication in the most significant part (MSP). And this is done in the partial product reduction tree without inclusion of any additional logic as explained in [1]. Multiplexers (muxes) are generally employed in TP multiplier to select appropriate partial product for N and $N/2$ b multiplication.

While implementing TP technique in the signed multipliers like MB [2], multiplexers are involved for the selection of partial products during N and $N/2$ b multiplication. This addition of multiplexers give rise to an area overhead, and thereby significant delay is added in the TP multiplier. By making the calculation of multiplexer utilisation, the TP implementation in MB multiplier [2] utilises $N+3$ multiplexers to select the partial products. The unwanted partial products that are not in use for $N/2$ bit multiplication are set to zero by modifying two input AND gate which produces partial products to three input AND gate and the third input is the control signal. Since the area overhead in TP multiplier is caused by the multiplexers, we aimed at optimising the TP multiplier by reducing the multiplexer utilisation. Multiplexer will get reduced only when the changes to be made in $N/2$ b multiplication are lesser. Our goal of obtaining optimised TP MB multiplier is achieved when implementing the TP technique in reduced computation modified booth (RCMB) that consists of $[N/2]$ rows and not $[N/2]+1$ rows as in modified booth algorithm and uses the simple sign extension prevention scheme which requires lesser changes to be made in TP implementation, i.e., for $N/2$ b multiplication.

In this paper, our goal of obtaining optimised double throughput signed multiplier has achieved without inclusion of complex logic, which reflects in reduced area, delay and power. We have implemented twin precision in an efficient manner with less hardware constraint compared to previous implementation, and a suitable algorithm is proposed. Implementing TP in RCMB [11] has achieved reduced area, delay and power compared to prior TP technique applied in MB algorithm. We have analysed that implementing TP technique in RCMB results in better performance, and it has been discussed in the rest of our paper.

2 TP implementation in modified booth

MB algorithm is a widely used algorithm for signed multipliers, and it holds an advantage of producing half the number of partial products. In this algorithm, X is multiplied by Y using modified booth encoding (MBE) scheme [12]. This encoding scheme groups Y into three bits as shown in Figure 1 and encodes it into one of the following {−2, −1, 0, 1, 2}, and the MB decoder generate partial products by multiplying encoded signal with the multiplicand Y. In the TP implementation made in MB, [2] adopts modified booth encoder and decoder are shown in Figure 2(a) & (b). Different methodologies of implementing MB recoding logic are explained in [13,14].

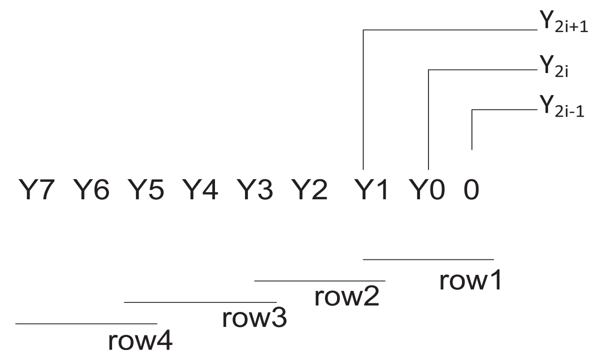


Figure 1 Encoding logic of 8-bit modified booth.

To multiply the encoded {−2, −1} with multiplicand X, two's complement representation of partial products is in need to indicate the change in sign bit. A '1' should be inserted in the least significant bit (LSB) of each partial product row for sign change which is a normal procedure in two's complement representation. While doing so, an irregular partial product array will be obtained, and this can be overwhelmed by the idea proposed in [15]. The idea is to precompute the two LSB positions of recoded partial product row by the insertion of 1 during the sign change. Actually, precomputation calculates the addition of LSB with potential '1' whose sum is the new LSB and carry is inserted in the second least significant bit position. Precomputation works according to the Equation 1 and its carry is calculated as given by Equation 2, which is adopted in [2].

$$\mathrm{PLSB}i = X_0(Y_{2i-1} \oplus Y_{2i}) \tag{1}$$

$$a_i = Y_{2i+1}(Y_{2i-1} + Y_{2i} + X_{LSB} + Y_{2i} + X_{LSB} + Y_{2i-1}) \tag{2}$$

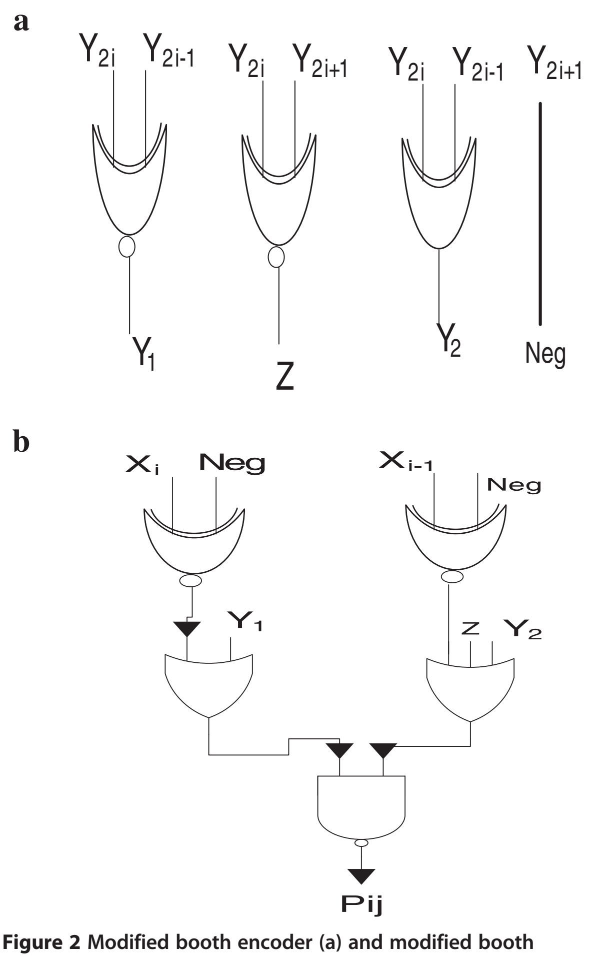


Figure 2 Modified booth encoder (a) and modified booth decoder (b).

TP technique facilitates the increase in computational throughput by allowing narrow width operation in parallel. Double throughput was obtained by the TP technique and was implemented in signed algorithm such as Baugh Woolley (BW) and MB [2]. Normally, TP multiplier either performs *N* b multiplication or *N*/2 b multiplication. If TP multiplier performs *N* bit multiplication, then its output sum (S) bits are from S0 to S15 as demonstrated in Figure 3, and when *N*/2 bit multiplication is performed, the output sum bits are from S0 to S7 and S8 to S15, i.e., twin output is taken as illustrated in Figure 4 and *N* = 8 for the illustrated example.

Implementation of TP technique is not similar for BW and MB algorithm. Figure 3 illustrates the MB multiplication for 8 × 8 which uses the sign extension scheme proposed by Fadavi [16] and uses recoding logic proposed in [15]. Achieving double throughput in MB algorithm requires appropriate selection of partial product signals in *N* and *N*/2 b multiplication, i.e., looking after sign extension prevention in LSB and MSB multiplication is to be taken care when *N*/2 b multiplications are performed. Simply we can say the steps performed for *N* b multiplication has to be carried over for *N*/2 b multiplication. Apart from booth encoding and decoding, the steps (changes) to be performed in *N*/2 LSB and MSB multiplication for MB algorithm [2] are precomputed LSB (PLSB) and its carry ai has to be calculated separately. Likewise inversion of MSB in each partial product row and sign bit pattern 1 s and 0 s will differ for *N*/2 LSB and MSB multiplication. Multiplexers will make these changes in *N* b partial product array by selecting appropriate partial products for *N* and *N*/2 b multiplication.

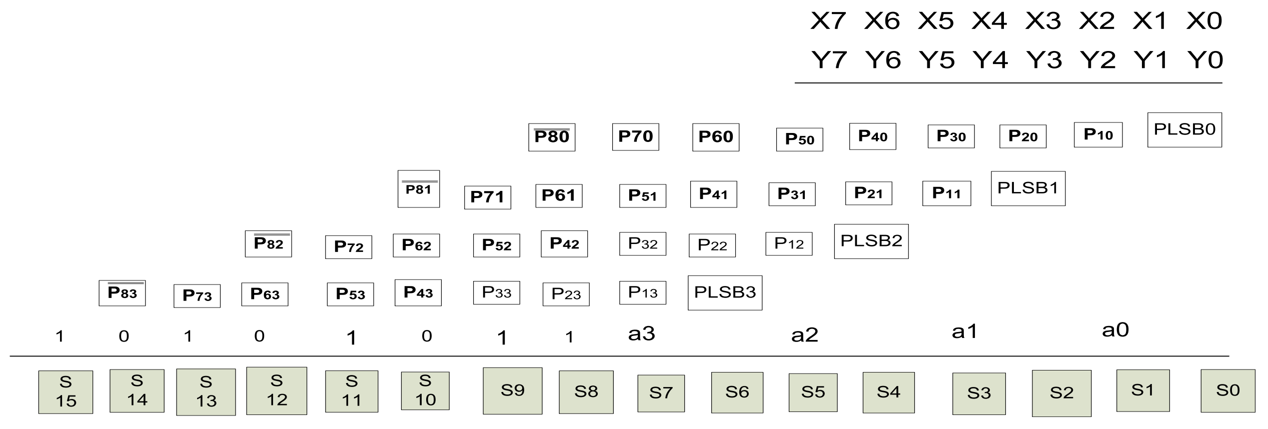

X7 X6 X5 X4 X3 X2 X1 X0
Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
P80 P70 P60 P50 P40 P30 P20 P10 PLSB0
P81 P71 P61 P51 P41 P31 P21 P11 PLSB1
P82 P72 P62 P52 P42 P32 P22 P12 PLSB2
P83 P73 P63 P53 P43 P33 P23 P13 PLSB3
1 0 1 0 1 0 1 1 a3 a2 a1 a0
S15 S14 S13 S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1 S0

Figure 3 **Signed 8-bit multiplication using modified booth algorithm-*N* bit multiplication.**

Since the partial product generation of MB multiplication is based on the encoding and decoding logic, it is not possible to use the results of full precision (N b multiplication) for narrow width ($N/2$ b) MB multiplication. So changes are to be made in the recoded N b partial product array to obtain double throughput. Though the MB algorithm reduces N number of partial product rows to $N/2$ rows, the inclusion of sign bit 1 s in the last row which is due to negative decoding of MBE makes partial product strength to $[N/2] + 1$. In [2] $(N/2 + 1)$th row is due to sign bit and potential carry (ai, $i = 0$ to $(N/2 - 1)$) of PLSB. The twin precision technique implemented in MB multiplication [2] consisting of $[N/2] + 1$ rows which has been illustrated in Figure 4 and double throughput is achieved, and Figures 3, 4, 5, and 6 are inspired from [2].

To obtain TP in modified booth, the partial products that are shaded (grey colour) in Figure 4 are needed when multiplier performs $N/2$ b multiplication whereas the non-shaded partial products are set to zero. The partial products are set to zero by connecting the MB decoder output to two input AND gate and the other input of the AND is the control signal. Based on this control signal, either MB decoded signal or zero will appear as the output. Multiplexers are needed to select the appropriate partial products for N and $N/2$ b multiplications. Encoding scheme for $N/2$ b multiplication is shown in Figure 5.

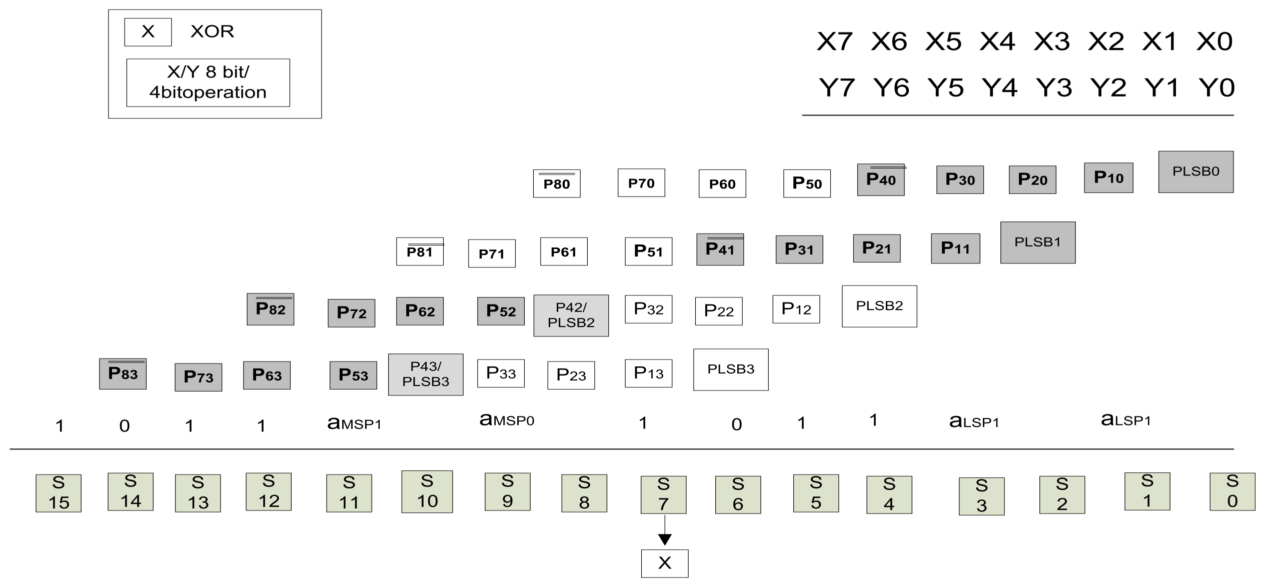

X XOR
X/Y 8 bit/ 4bitoperation
X7 X6 X5 X4 X3 X2 X1 X0
Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
P80 P70 P60 P50 P40 P30 P20 P10 PLSB0
P81 P71 P61 P51 P41 P31 P21 P11 PLSB1
P82 P72 P62 P52 P42/ PLSB2 P32 P22 P12 PLSB2
P83 P73 P63 P53 P43/ PLSB3 P33 P23 P13 PLSB3
1 0 1 1 aMSP1 aMSP0 1 0 1 1 aLSP1 aLSP1
S 15 S 14 S 13 S 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
X

Figure 4 **Implementation of twin precision in modified booth multiplier-*N*/2 multiplication.**

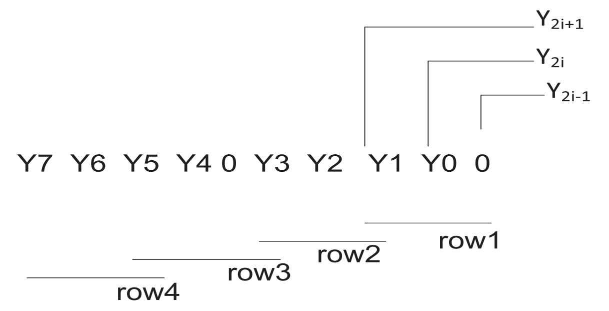


Figure 5 MB encoding scheme for *N*/2 multiplication.

By comparing Figures 3 and 4, the changes to be made to obtain twin precision are

- Partial products – P40 and P41 during normal 8 × 8 multiplications (Figure 3) need to describe sign extension prevention for *N*/2 LSB multiplication in MSB (Figure 4).
- Partial products – P42 and P43 in normal (*N* bit) multiplication has to be replenished as PLSB2 and PLSB3 for *N*/2 MSB bit multiplication.
- aMSP0 and aMSP1 are to be added for *N*/2 MSB multiplication.
- The pattern 1 s and 0 s are different for *N* and *N*/2 bit multiplication.

In Figure 6, the partial products are mapped to high-performance multiplier (HPM) reduction tree [17] and the shaded regions indicate that they are involved in *N*/2 b multiplication. Another issue that has to be looked through in *N*/2 b multiplication is that the multiplication in the LSB should not interfere with MSB multiplication, and so an encoding logic in Figure 5 has been followed. The 1 s and 0 s pattern in the last row of the partial product array is for the sign extension prevention and the most significant '1' is to invert the S2*N*-1 bit. So while performing *N*/2 LSB multiplication, the most significant '1' present in the S7 column produces a carry, and when it propagates, it will affect the *N*/2 MSB multiplication result. In order to avoid this problem, output of the S7 is passed to an exclusive OR (XOR) gate and the inversion is made, and it is illustrated in Figure 6 where the most significant '1' is made to zero and S7 output is passed to the XOR gate.

Apart from setting the unwanted partial products to zero in *N*/2 b multiplication, the selection of partial products for *N* and *N*/2 b multiplication are done by multiplexers. These multiplexers are added to the TP multiplier design to get the dual output, and thereby a significant delay is added. This delay increases linearly as the bit width (*N*) of multiplier increases because for a higher bit width, more multiplexers are utilised. The number of multiplexers depends upon the changes (steps) or selection of appropriate signals in each column to produce sum bit (S). For an 8 × 8 MB multiplier which adopts sign extension scheme [16] and recoding logic [15], the changes that have to be made for implementing TP in each column to

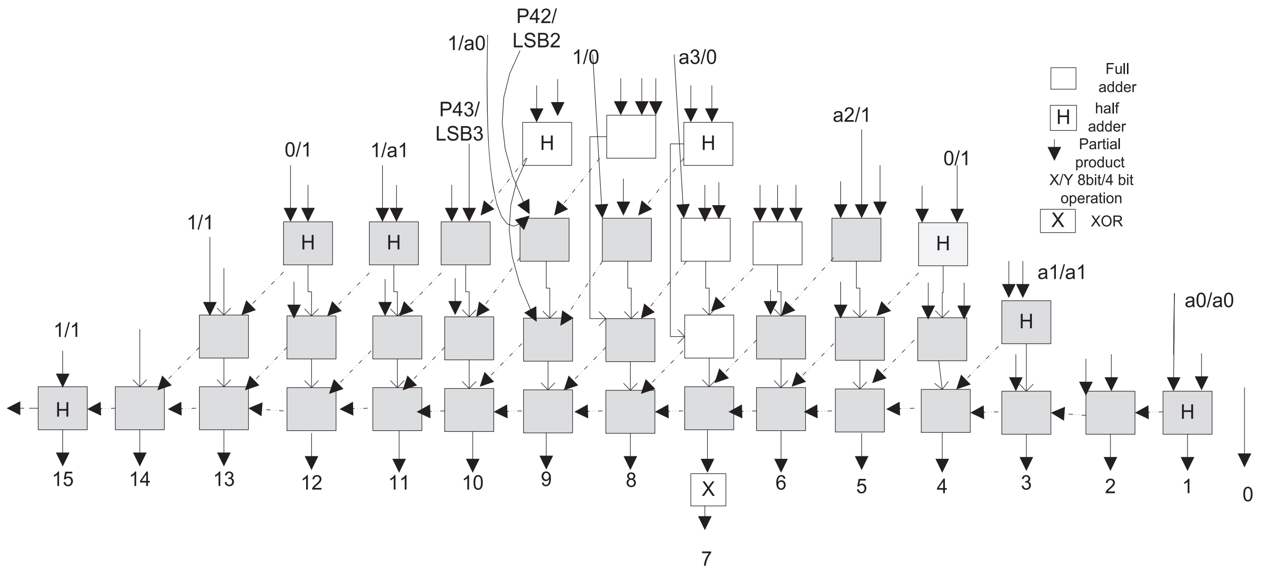


Figure 6 TP implementation of MB in HPM.

produce sum bits S0 to S15 are elaborated in Table 1. The signal selection x/y in Table 1 represents partial product (P) for N bit/ P for $N/2$ bit multiplication.

By comparing Figures 3 and 4, the changes needed to perform in each column of 8×8 MB multiplication for TP are tabulated in Table 1. Depending on the multiplication performed (N or $N/2$ b), the appropriate changes that have to be made in the partial product array are executed by multiplexers. In N b multiplication (Figure 3), the S4 column usually has no sign bit, but when $N/2$ LSB multiplication (Figure 4) is performed, '1' has to be included as sign bit and the inversion of MSB bit (P40) has to be performed. So in the S4 column, two multiplexers are required to make these two changes. As explained earlier, the MSB bit of each partial product row has to be inverted so P41 bit has to be inverted for $N/2$ LSB multiplication . In the seventh column, the potential carry (a3) in N bit partial product array has to be replaced as sign bit 1 for $N/2$ LSB multiplication. Likewise the changes such as precomputed LSB (PLSB2, PLSB3), the potential carry (aMSP0, aMSP1), and sign bit pattern 1 s and 0 s made for $N/2$ MSB multiplication are tabulated in Table 1.

Totally, for an 8×8 TP MB multiplication, we need 11 multiplexers. For various bit width, the multiplexer (mux) utilisation is calculated. From the analysis made in the MUX utilisation for implementation of TP in MB [2], it is clear that this method

Table 1 Changes needed for twin precision

Sum bit	Twin 2009
	Signal selection
S0	No change
S1	No change
S2	No change
S3	No change
S4	1/0 and P40/$\bar{P}$40
S5	1/a2
S6	P41/$\bar{P}$41
S7	0/a3
S8	PLSB2/P42 and 0/1
S9	aMSP0/1
S10	PLSB3/P43
S11	aMSP1/1
S12	1/0
S13	No change
S14	No change
S15	No change

inquires $N + 3$ multiplexers for selection of appropriate partial products, where N is the bit width of multiplier.

3 Implementation of TP in RCMB

In this study, we have designed an optimised TP multiplier. Optimization is achieved by implementing TP technique in RCMB and an algorithm is proposed which is applicable for $N = 8 \times$ (multiples of eight). Implementation of TP in RCMB reduces the mux utilisation to $(N/2 + 3)$. Muxes are normally employed in TP multiplier for partial product selection in N and $N/2$ b multiplication. In prior work [2], the implementation of TP involves $N + 3$ multiplexers for partial product selection in TP multiplier. By applying our proposed algorithm, the TP can be obtained in the RCMB multiplier.

RCMB algorithm [11] is the classic twos complement that uses a radix-4 MBE scheme. Partial product row in RCMB (Figure 7) consists of partial product (Pi,j) that are generated based on booth encoding and decoding, negk signals ($k = 0$ to $((N/2) - 1)$) are added in the LSB position of each partial product row for generating twos complement and 1 s in the leftmost part of partial product rows that are for sign extension in twos complement representation. Figure 1 in [11] describes the gate level diagram of partial product generation. The maximum height of the partial product in MB is $(N/2 + 1)$. This is reduced to $N/2$ rows in RCMB algorithm which is the biggest advantage. The neg$(N/2 - 1)$ bit which actually lies in the $(N/2 + 1)$th row of partial product is added to the MSB part of first row partial product represented as qij, i.e., in Figure 7, we have $\bar{q}90$ q90 q80 q70 q60 = 0 0, $\bar{P}80$ P70 P60 + 0 1 1 0 neg3 (refer to Figure 6 in [11]). And by this way, the height of partial product is reduced to $N/2$. Finally the partial products are mapped to half and full adders in HMM tree [17] structure.

In our proposed algorithm, we have assumed the partial product rows of RCMB as reduced computation matrix set (R_s). This set is divided equally and named it as upper and lower bound as shown in Figure 8. This set consists of partial product Pij, negk signals ($k = 0$ to $((N/2) - 1)$) and 1 s for sign extension. This set consists of i rows and j columns where $i = (1$ to $N/2)$ and $j = (1$ to $2N)$ and represented as $R_s\{i,j\}$. The N b procedure (steps) has to be followed for $N/2$ b multiplication. The changes or steps to be performed for $N/2$ (LSB and MSB) bit multiplication in N bit architecture are inversion of MSB in each partial product row which has to be made for twos complement method, addition of 1 s for sign extension prevention, generation of negk

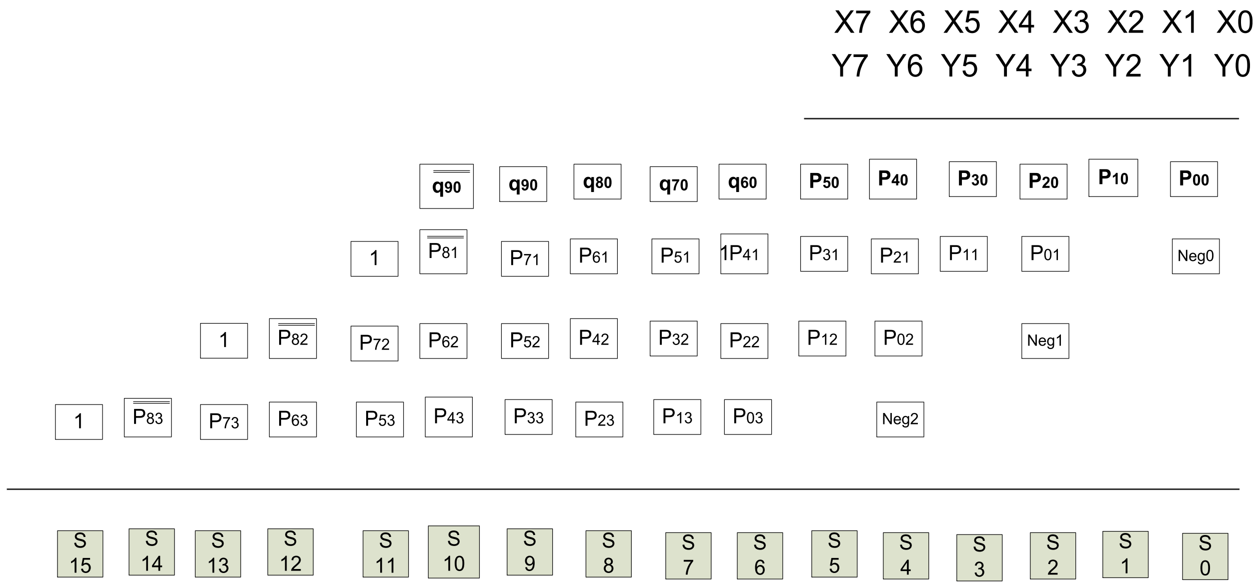

X7 X6 X5 X4 X3 X2 X1 X0
Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
q90 q90 q80 q70 q60 P50 P40 P30 P20 P10 P00
1 P81 P71 P61 P51 1P41 P31 P21 P11 P01 Neg0
1 P82 P72 P62 P52 P42 P32 P22 P12 P02 Neg1
1 P83 P73 P63 P53 P43 P33 P23 P13 P03 Neg2
S15 S14 S13 S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1 S0

Figure 7 **8 × 8 multiplication using reduced computation modified booth (RCMB).**

signals in the LSB position for generating twos complement. While performing $N/2$ LSB multiplication, the changes are to be in the upper bound, and for $N/2$ MSB multiplication, changes are made in the lower bound. Multiplexers are employed in the corresponding places of N b architecture where the changes are to be made, and depending upon the multiplication, appropriate partial products are selected. Each bit (element) in R_s is denoted as Bx, and $x = 1$ to $N + 3$ denotes relative bit position in each row.

In this paper, we have implemented TP technique in RCMB multiplier which has $N/2$ partial rows. The sign extension prevention scheme proposed in [18] is used in the RCMB. In our proposed algorithm, much of the work has to be done in the upper bound compared to the lower bound. Inversion of most significant bit in partial product rows and adding of sign bit 1 s are two major steps followed to perform twin precision. In prior work of implementing twin precision technique [2] to obtain double throughput, more changes (steps) are to be performed in partial product array than proposed implementation of TP in RCMB. This in turn increases the need of multiplexers. Therefore, the area overhead for the implementation of twin precision in MB algorithm is higher than our proposed method. While obtaining twin precision in RCMB algorithm, the $N/2$ rows will be increased to $(N/2) + 1$ that uses sign extension prevention scheme in [18], and MB recoding logic proposed [19]. When the TP technique is implemented in RCMB, the multiplexer utilisation reduces drastically, and its impact on area, delay and power is analysed.

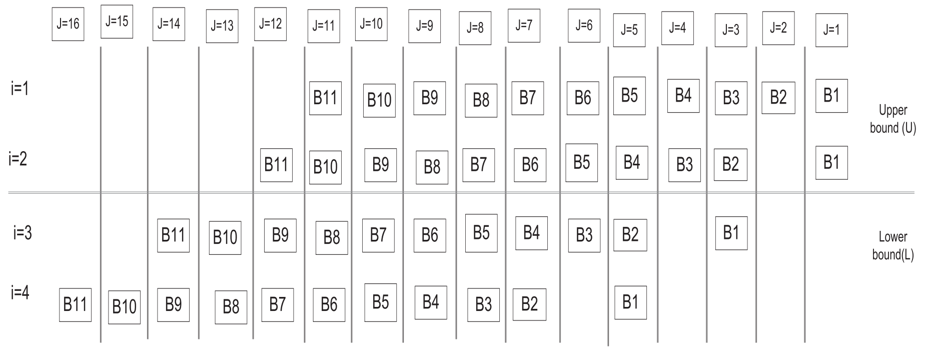

J=16 J=15 J=14 J=13 J=12 J=11 J=10 J=9 J=8 J=7 J=6 J=5 J=4 J=3 J=2 J=1
i=1
B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1
i=2
B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1
Upper bound (U)
i=3
B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1
i=4
B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1
Lower bound(L)

Figure 8 **Reduced computation matrix RS for 8 × 8 multiplication.**

3.1 Algorithm for obtaining twin precision in RCMB

1. $R_s\{i,j\}$ - reduced computation matrix set

   $i = 1\ to\ \frac{N}{2}\ ; j = 1\ to\ 2N;$
   N-bit width

2. For $N = 8$, upper bound $= U\{i_1,i_2\}$; lower bound $= L\{i_3,i_4\}$
3. $(Pi, j \in (i, j))$; Partial product is an element of *i* and *j*
4. $B_T = N + 3$; B_T - total bits (*BT*) in a row
5. Inversion and addition of sign bit 1s are needed for TP

   (i) Inversion of MSB bit while performing *N*/2 LSB multiplication is given as $B[X_i]$. X_i value gives relative bit position in the *i*th row so that inversion of MSB can be made correctly

   $$\text{for}\begin{cases} i = 1 & X = X_i = B_T - C_N \\ i = 2\ to\ \frac{N}{4} & X = X_i = X_{(i-1)} + 2 \end{cases}$$

   $B[X_i]$-bit (*B*) position (*X*) in the corresponding *i* row
   $C_N = \frac{N}{2} + 2$
   C_N - constant which depends on *N* value

   (iia) Addition of 1s in corresponding columns (*j*) are made for following iterations

   Iteration (*I*) = 1 to (*N*/4) + 1

   $$\text{for}\begin{cases} I = 1 & j_{ou(I)} = j_{ou(1)} = \left(\frac{N}{2}\right) + 1 \\ I = 2 & j_{ou(I)} = j_{ou(2)} = \left(\frac{N}{2}\right) + 2 \\ I = 3\ to\ (N/4) + 1 & j_{ou(I)} = j_{ou(I-1)} + 2 \end{cases}$$

   $J_{ou(I)}$ ones (*O*) in upper (*U*) bound in corresponding column (*j*) for corresponding iteration (*I*)

   (iib)

   $$OB_{L(j)} = \left(\frac{N}{2} + 1 + N\right)$$

   $OB_{L(j)}$ - addition of one in the lower bound in the corresponding column *j*

6. Negative bits (negk) have to be inserted in the LSB part of each partial product row for *N*/2 MSB multiplication. For this, we need to know the column (*j*) with respect to row (*i*). *k* varies from 0 to (*N*/2 − 1).

   $$\text{negk} \rightarrow j_{(i)}$$

   $$\text{for}\begin{cases} i = \frac{N}{4} + 1, k = N/4\ j_{(i)} = N + 1 \\ i = \left(\frac{N}{4} + 2\ to\ N/2\right), k = \left(\left(\frac{N}{4} + 1\right)\ to\ \left(\frac{N}{2} - 1\right)\right)\ j_{(i)} = j_{(i-1)} + 2 \end{cases}$$

   i = row, *j* = column
   negk bits where *k* = 0 to (*N*/2) − 1

The total bits in any row in R_s will be equal and Equation 3 gives the formula to calculate total bits in a row. We have generalised the partial product (*Pij*), 1 s that are for twos complement implementation and negk bit in each row as a bit (*B*).

$$B_T = N + 3 \tag{3}$$

Proof

$$\text{For}\, N = 8 \quad B_T = 8 + 3 = 11$$

$B_T(i)$ - total bits (*BT*) in a row

For an 8 × 8 RCMB multiplier illustrated in Figure 8, $B_T = 11$ and this is equal for all the rows. When we perform *N*/2 LSB multiplication in *N* b tree, the most significant bit in each partial product row has to be inverted (negated). To perform this, we need to know the bit position (*Xi*) in the upper bound with respect to corresponding row (*i*) and given as *B*[*Xi*] in Equation 4, where *B* is the bit and *X* denotes the bit position. In the lower bound, there is no need for inversion of bits for *N*/2 MSB multiplication because during *N* bit multiplication itself, the inversion was made. The inversion of bits that has to be implemented in the upper bound in corresponding bit position of *i*th row is described in Equation 4.

For rows $i = 1$ to *N*/4, the inversion of bits *B*[*Xi*] are done. For initial condition $i = 1$, [*Xi*] is the difference of total bits $B_T(i)$ in row and constant C_N.

$$for\begin{cases} i = 1 & X = X_i = X_1 = B_T - C_N \\ i = 2\,to\,\dfrac{N}{4} & X = X_i = X_{(i-1)} + 2 \end{cases} \tag{4}$$

$$C_N = \frac{N}{2} + 2$$

B[*Xi*] - bit position in the corresponding *i* row
C_N - constant, which depends on *N* value
Proof
For $N = 8$

$$i = 1 \quad X = X_i = B_T - C_N$$
$$X_1 = 11 - 6 = 5$$
$$i = 2\,to\,\frac{N}{4} X = X_i = X_{(i-1)} + 2$$
$$X = X_i = X_2 = 5 + 2 = 7$$

Moreover, for the succeeding values of *i* that is up to *N*/4, the inversion bit position is the sum of previous iteration value and 2. For $N = 8$, the values of $X_1 = 5$ and $X_2 = 7$, so the fifth and seventh bit has to be inverted in the first and second row as implemented in Figure 9. The changes are made in the upper bound for *N*/2 LSB multiplication and in the lower bound for *N*/2 MSB multiplication.

Next step in the algorithm to perform TP is the addition of sign bit 1 s for sign extension in the upper and lower bound and has to be included in the suitable column. For this, much work has to be performed in the upper bound compared to the lower bound. Because for *N* b multiplication, RCMB algorithm already has sign extension bit 1 s in the most significant bit position of the lower bound. So in our algorithm for obtaining TP, addition of 1 s for sign extension in corresponding column (*j*) is performed in the upper bound according to Equation (5). The addition of sign bit 1 s and 0 s pattern that is used for sign extension prevention will vary for *N* and *N*/2 b multiplication. It is not necessary to find the bit position for the insertion of 1 s as we do for inversion of bits to get the correct sum bit (*S*). While performing inversion (negation) of bits in the upper bound, bit position is needed for the corresponding *i* because if the inversion is made in the wrong bit position, then we will wind up with incorrect sum bit and thereby will end up with the erroneous output. But in the case of adding 1 s, it is just an addition of extra signal, so finding out the corresponding column (*j*) is sufficient. Total 1 s to be added in upper bound for *N*/2 LSB multiplication depends on iteration (*I*).

$$for\begin{cases} I = 1 & j_{ou(I)} = j_{ou(1)} = \left(\dfrac{N}{2}\right) + 1 \\ I = 2 & j_{ou(I)} = j_{ou(2)} = \left(\dfrac{N}{2}\right) + 2 \\ I = 3\,to\,(N/4) + 1 & j_{ou(I)} = j_{ou(I-1)} + 2 \end{cases} \tag{5}$$

$j_{ou(I)}$ - ones (*O*) in the upper (*U*) bound for iteration (*I*)
Proof
$N = 8$ Iteration $I = 1$ to 3

$$I = 1 \qquad j_{ou(1)} = \left(\frac{8}{2}\right) + 1 = 5$$
$$I = 2 \qquad j_{ou(2)} = \left(\frac{8}{2}\right) + 2 = 6$$
$$I = 3 \qquad j_{ou(3)} = j_{ou(3-1)} + 2 = 6 + 2 = 8$$

So in the fifth, sixth and eighth columns, the 1 s are inserted

Iteration (*I*) varies from 1 to ($N/4 + 1$). For addition of 1 s in the upper bound, the iteration (*I*) lies from 1 to $(N/4) + 1$ and for a first two iteration, i.e., $i = 1$ and 2, the column where 1 has to be added is $\left(\frac{N}{2}\right) + 1$ and $\left(\frac{N}{2}\right) + 2$. And for the subsequent values of *I*, it is the sum of previous iteration value and 2. For $N = 8$, the $I = 1$ to 3, and according to Equation 5, the columns are calculated for the insertion of 1 s. From proof, it is clear that the 1 s are added in the fifth, sixth and eighth columns. In Figure 9, the ‘1’ that is inserted in the fifth column is joined with partial product represented as P02/1 (Figure 10, Table 2) which implies

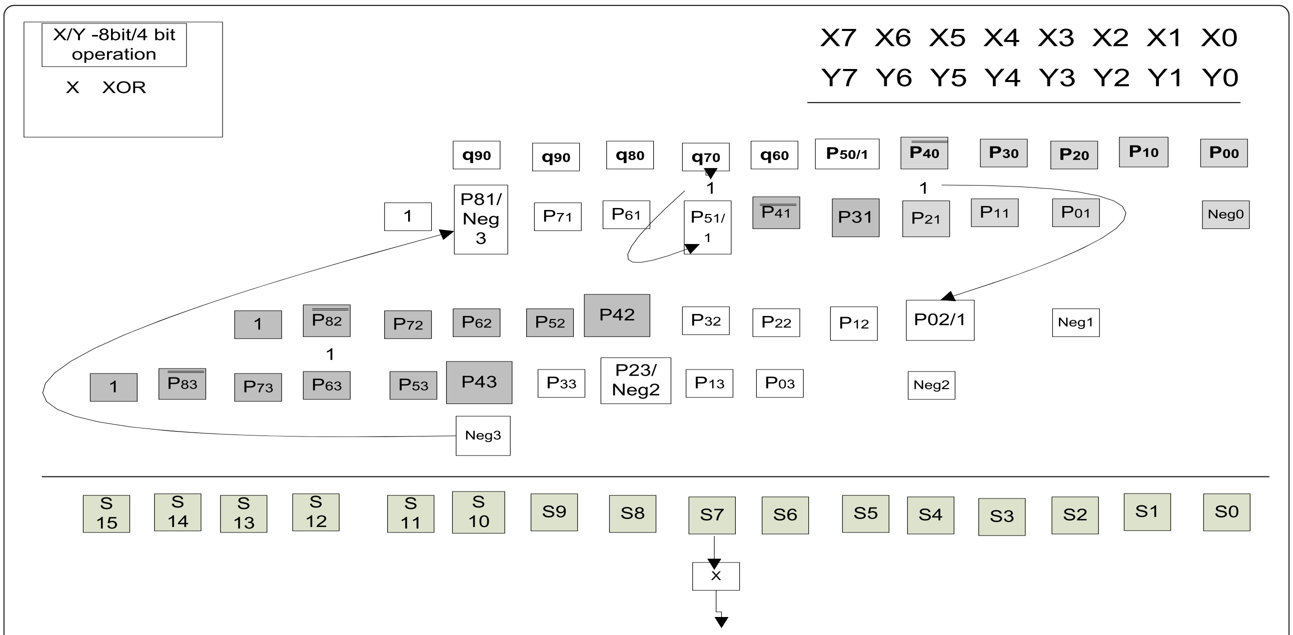

X/Y -8bit/4 bit operation
X XOR
X7 X6 X5 X4 X3 X2 X1 X0
Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
q90 q90 q80 q70 q60 P50/1 P40 P30 P20 P10 P00
1 P81/Neg3 P71 P61 P51/1 P41 P31 P21 P11 P01 Neg0
1 P82 P72 P62 P52 P42 P32 P22 P12 P02/1 Neg1
1 P83 P73 P63 P53 P43 P33 P23/Neg2 P13 P03 Neg2
Neg3
S15 S14 S13 S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1 S0
X

Figure 9 Implementation of our proposed algorithm in RCMB to obtain twin output.

during N bit multiplication P02 is used whereas for $N/2$ LSB multiplication, the '1' is utilised. And this selection is made by multiplexers. Likewise '1' in the sixth column is joined with P50. '1' that is inserted in the eighth column is to invert the *SN-1* bit. If 1 is added in the eighth column, it generates a carry and affects the $N/2$ MSB multiplication. To avoid this problem, the MSB bit of $N/2$ LSB multiplication is passed through an XOR gate after the final addition and the inversion of *SN-1* bit made.

In the lower bound, 1 has to be added in only one column for TP implementation as per Equation 6. Because in N b multiplication of RCMB, the sign bit 1 s are already present for each partial product row, so there is no need of extra effort to insert 1 s in the lower bound for $N/2$ MSB multiplication.

$$OB_{L(j)} = \left(\frac{N}{2} + 1 + N\right) \tag{6}$$

Proof
$N = 8$

$$OB_{L(j)} = \left(\frac{N}{2} + 1 + N\right) = (8/2 + 1 + 8) = 13$$

In thirteenth column, 1 is added

$OB_{L(j)}$ - one (O) in the lower bound (BL) in corresponding column (j)

From the proof of Equation 6, it is clear 1 has to be added in the 13th column. All the steps in the algorithm are made in RCMB multiplier and mapped in HPM

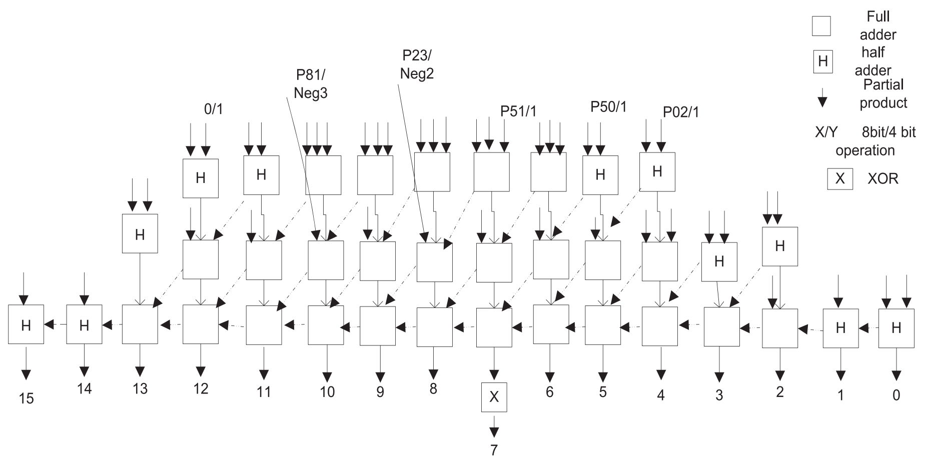


Figure 10 RCMB in HPM.

Table 2 Changes to be made to obtain TP

Sum bit	Twin 2009 [2]	Proposed twin signal selection
S15	No change	No change
S14	No change	No change
S13	No change	No change
S12	1/0	1
S11	aMSP1/1	No change
S10	PLSB3/P43	P81/neg3
S9	aMSP0/1	No change
S8	PLSB2/P42 and 0/1	*P*23/neg2
S7	0/a3	*P*51/1
S6	P41/$\bar{P}$41	P41/$\bar{P}$41
S5	1/a2	*P*31/1
S4	1/0 and *P*40/$\bar{P}$40	*P*02/1 and *P*40/$\bar{P}$40
S3	No change	No change
S2	No change	No change
S1	No change	No change
S0	No change	No change

reduction tree [17] shown in Figure 10. For N b multiplication, the sum (S) bit is from S0 to S15, and for $N/2$ LSB and MSB multiplication, sum bits are S0 to S7 and S8 to S15.

In RCMB [11], the $N/2+1$ rows are reduced to $N/2$ rows by the addition of last row sign bit, i.e., (neg(($N/2$) − 1)) bit to MSB part of the first row partial product. In our chosen example in Figure 7, it is neg3 bit which leads to $(N/2)+1$ rows added to the first row whose partial product are represented as qij. We exploited TP technique in RCMB multiplier and suitable algorithm is proposed. In our proposed TP implementation in RCMB when we perform $N/2$ MSB multiplication, the first row ($i=3$) of lower bound is not modified like N bit multiplication because the complexity of the design will get increased.

For N bit multiplication the neg3 bit is added to the first row whereas for $N/2$ bit MSB multiplication, the sign bit (neg3) is not added to the first row ($i=3$). But for $N/2$ MSB multiplication, P23 is replaced by the sign bit neg3 by multiplexer. This way of replacing the partial product as neg bit for $N/2$ multiplication maintains the partial product array to $(N/2)$ rows without the first row modification for $N/2$ MSB multiplication.

Proof

$N = 8$

$$negk \rightarrow j_{(i)}$$
$$i = \frac{N}{4}+1, k = N/4 \quad j_{(i)} = N+1$$

$i=3,\ k=2\ j_{(3)} = 8+1 = 9$ neg2 → 9 (neg2 bit is added to ninth column)

$$i = \left(\frac{N}{4}+2, \frac{N}{4}+3\, to\, N/2\right), k = \left(\left(\frac{N}{4}+1\right) to \left(\frac{N}{2}-1\right)\right)$$
$$j_{(i)} = j_{(i-1)}+2$$
$$i = N/2+2 = 4,\ k = N/4+1 = 3$$

$j_{(4)} = j_{(4-1)}+2 = 9+2 = 11$ neg3 → 11 (neg3 bit is added to eleventh column)

Note: $i=1$ to $N/2$ (first step in the 'Algorithm for obtaining twin precision in RCMB' section). For $N=8$, i varies from 1 to 4. Since i value ends with 4, it is not necessary to find $j_{(i)}$ for values of i after $N/4+2$ (i.e.,) for $i = N/4+3 = 5$ where we do not have $i=5$ (fifth row) in 8 × 8 ($N=8$) RCMB multiplier.

The negk ($k=0$ to $(N/2-1)$) bits are usually used to generate twos complement in RCMB algorithm. These bits are added to the LSB part in each partial product row. During N bit multiplication, negk bits are added in the LSB part of each partial product row. So for $N/2$ LSB multiplication, we do not face any problem in adding the neg(0 to $(N/4-1)$) bits in the LSB part, i.e., in the upper bound. But for $N/2$ MSB multiplication, the neg($N/4$ to $(N/2-1)$) bits have to be added in the corresponding LSB part of each partial product row in the lower bound. By Equation 7, this can be performed by adding neg($N/4$ to $(N/2-1)$) bits to the corresponding column. From proof of Equation 7, it is clear the neg2 and neg3 are added in the 9th and 11th columns. In Figure 9, the neg2 bit is joined with P23, i.e., for $N/2$ MSB multiplication; instead of P23, the neg2 bit will be selected by the multiplexer. Likewise in the tenth column, neg3 bit is joined with P81.

And by implementing TP technique in RCMB, we found that our approach requires less mux utilisation compared to previous implementation [2]. Figure 9 depicts the implementation of all the above steps for N and $N/2$ bit multiplication. When the multiplier performs

$$\text{for}\begin{cases} i = \frac{N}{4}+1, k = N/4 & j_{(i)} = N+1 \\ i = \left(\frac{N}{4}+2, \frac{N}{4}+3\, to\, N/2\right), k = \left(\left(\frac{N}{4}+1\right) to \left(\frac{N}{2}-1\right)\right) & j_{(i)} = j_{(i-1)}+2 \end{cases} \tag{7}$$

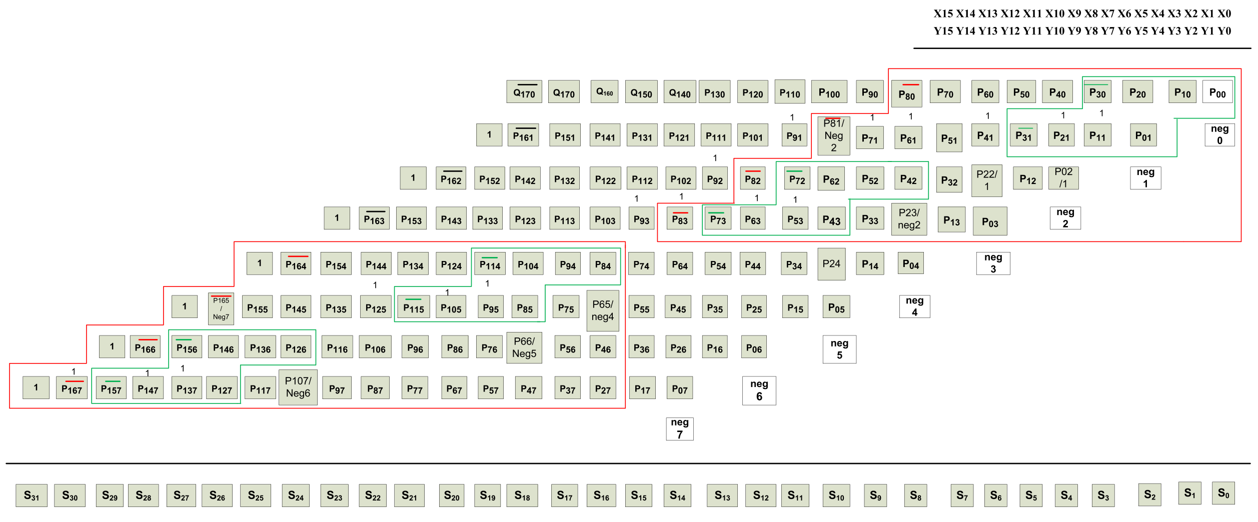
N/2 b multiplication, the unwanted partial products that are not shaded in Figure 9 are set to zero and the multiplexers are implemented where ever the changes are required. The columns where the changes have to be made in RCMB to obtain twin output using our algorithm are S4, S5, S7, S8, S10 and S12 as charted in Table 2. And by consolidating these changes, we can formulate the mux utilisation for our approach.

Here in this study, we have chosen 8 × 8 RCMB multiplier to explain our idea. From the consolidated changes, it is revealed that we entail (*N*/2) + 3 multiplexers and verified for various bit width multiples of eight as outlined in Table 2. The earlier work [2] utilises *N* + 3 multiplexers for the selection of appropriate signals. Since multiplier structure is the interconnection of full adders and half adders, the adders involved in the critical path have to wait for the previous carry signal to produce output sum bit S. Implementation of our proposed algorithm for obtaining TP in RCMB results in less multiplexers utilisation which has its impacts on area, delay and power that are analysed in the Results and discussion section. Because of this drastic reduction in multiplexers, the area, delay and power of TP multiplier are reduced compared to prior work of implementing TP in modified booth algorithm [2].

4 SWP

A subword is a lower precision unit of data contained within a word. When SWP concept is adopted in multiplier, then multiple lower precision multiplications are possible. By applying SWP to signed multiplication like modified booth, the MB algorithm has to be performed for all the lower precision multiplication. While doing so, the number of multiplexers increases for selecting appropriate partial products. And this increases the hardware complexity of the design than TP multiplier. In this section, we have explained multiple SWP in RCMB multiplier for *N* = 16, i.e., four 4-bit multiplications or two 8-bit multiplications (*N*/2 or *N*/4) are performed which is shown in Figure 11. When two 8-bit multiplications are performed, i.e., only *N*/2 operation is performed and it is called as TP multiplication. This is the reason we say TP is a subset of SWP. The 'Algorithm for obtaining twin precision in RCMB' section describes the operation of TP in RCMB, and we need (*N*/2 + 3) multiplexers for selection of partial products. Apart from this, we need to calculate mux utilisation for four 4-bit operations in multiple SWP.

The bounded regions in Figure 11 that are green in colour illustrate four 4-bit (*N*/4) multiplications and red bounded region shows TP, i.e., *N*/2 multiplications. Multiple SWP multiplication means either *N*/2 or *N*/4 bit multiplication is possible. While doing four 4-bit multiplications, the first lower precision multiplication will perform for inputs *x*0 to *x*3 and *y*0 to *y*3, second precision multiplication inputs are from *x*4 to *x*7 and *y*4 to *y*7, third is from *x*8 to *x*11 and *y*8 to *y*11 and fourth is from *x*12 to *x*15 and *y*12 to *y*15. For all these four lower precision multiplication, RCMB algorithm is applied. In this algorithm, all the MSB bit of partial product row has to be inverted for achieving twos complement, 1 s are added for sign extension and negk bits are added in the LSB of each partial product row. For first lower precision multiplication, the first two rows (only green bounded region) of partial product in Figure 11 for *N* = 16 are taken into consideration. Since the inputs are *x*0 to *x*3 and *y*0 to *y*3, the MSB bit for this multiplication is P30 and P31 has to be inverted to achieve twos


X15 X14 X13 X12 X11 X10 X9 X8 X7 X6 X5 X4 X3 X2 X1 X0
Y15 Y14 Y13 Y12 Y11 Y10 Y9 Y8 Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
Q170 Q170 Q160 Q150 Q140 P130 P120 P110 P100 P90 P80 P70 P60 P50 P40 P30 P20 P10 P00
1 P161 P151 P141 P131 P121 P111 P101 P91 P81/Neg2 P71 P61 P51 P41 P31 P21 P11 P01 neg 0
1 P162 P152 P142 P132 P122 P112 P102 P92 P82 P72 P62 P52 P42 P32 P22/1 P12 P02/1 neg 1
1 P163 P153 P143 P133 P123 P113 P103 P93 P83 P73 P63 P53 P43 P33 P23/neg2 P13 P03 neg 2
1 P164 P154 P144 P134 P124 P114 P104 P94 P84 P74 P64 P54 P44 P34 P24 P14 P04 neg 3
1 P165/Neg7 P155 P145 P135 P125 P115 P105 P95 P85 P75 P65/neg4 P55 P45 P35 P25 P15 P05 neg 4
1 P166 P156 P146 P136 P126 P116 P106 P96 P86 P76 P66/Neg5 P56 P46 P36 P26 P16 P06 neg 5
1 P167 P157 P147 P137 P127 P117 P107/Neg6 P97 P87 P77 P67 P57 P47 P37 P27 P17 P07 neg 6
neg 7
S31 S30 S29 S28 S27 S26 S25 S24 S23 S22 S21 S20 S19 S18 S17 S16 S15 S14 S13 S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1 S0
Figure 11 Multiple SWP in RCMB multiplier for *N* = 16.

complement. Multiplexer will select either $P30$ or $P\bar{3}0$ and $P31$ or $P\bar{3}1$. Sign bit ‘1’ has to be added in fourth, fifth and seventh columns which is for sign extension in RCMB algorithm. In the fourth column, none of the partial product is available to merge sign bit ‘1’, so this leads to extra half adder during the addition of partial products to produce sum bit (S) and this sign bit will be 0 for $N/2$ and N bit operation. Whereas in fifth and seventh column sign bit 1 is merged with P02/1, P22/1, i.e., mux is employed in the places of P02 and P22 which selects either P02 or 1 depends on the operation. neg0 and neg1 are added in the LSB. This leads to utilisation of four multiplexers in the first lower precision region bounded in the green colour to select the partial products for N/4 operation.

Likewise for second lower precision multiplication, the algorithm has to be applied in the third and fourth rows. The MSB bit such as $P72$, $P73$ are to be inverted and sign bit 1 has to be added in the 12th, 13th and 15th columns. In the 12th column, sign bit 1 has been inserted for $N/2$ operation so this can be used for $N/4$ bit operation and we can avoid mux for this operation. In the 13th and 15th columns, sign bit 1 has been added for second lower precision multiplication and it is merged with partial products which are not involved in $N/4$ bit operation. neg2 and neg3 bits have to be added in the LSB position, i.e., in columns 9 and 11. $P23$ and $P81$ will be replaced as neg2 and neg3 by mux which is represented as $P23$/neg2 and $P81$/neg3 in Figure 11. Since we have inserted mux in these places for $N/2$ bit operation, it is not necessary to add additional mux. Totally, four muxes are added in the second lower precision region. Likewise in the third and forth lower precisions, the algorithm is applied which utilises four muxes for each multiplication. Apart from mux utilisation of $N/2$ bit operation (twin precision) by making the above changes, we need 16 muxes in $N/4$ operation for $N = 16$. So to perform multiple SWP with either $N/2$ or $N/4$ bit RCMB multiplication, we need 27 muxes in the multiplier architecture. This mux calculation for multiple SWP is made by addition of mux count in the proposed TP multiplier as tabulated in Table 3 and mux required for four 4-bit multiplications. This results in an increase in area than our proposed twin multiplier. Table 4 shows mux count for proposed twin and multiple SWP. Throughput in the signed multiplier like MB, RCMB can be increased beyond two in multiple SWP. But if the throughput is increased beyond two, then mux count will also increase as shown in Table 4. The unwanted partial products (Pij) that are not required for $N/4$ bit operation are made zero. This is done by passing unwanted Pij to two input AND gate, and other input will act as control signal and for $N/4$ bit operation; more partial products are made zero than $N/2$ operation. And due to these reasons, area overhead will be higher than proposed twin. Hereby we conclude hardware complexity is more for multiple SWP while performing signed multiplication based on algorithms like MB and RCMB. We have calculated mux count for multiple SWP for $N = 16$, 32 in Table 4. Likewise for higher bit width mux utilisation will be higher than proposed twin.

Table 3 Mux utilisation

S number	Bit width	Twin 2009 MUX ($N + 3$)	Proposed twin mux ($N/2$) + 3
1	8	11	7
2	16	19	11
3	32	35	19
4	64	67	35
5	128	131	67

5 Results and discussion

In this paper, we have implemented TP technique in RCMB and a suitable algorithm is proposed for obtaining double throughput, which is applicable for all bit width that are multiples of eight. Our approach of implementing TP in RCMB gives better performance compared to [2]. Our implementation of the proposed algorithm yields less mux utilisation, and its impacts on area, delay and power compared to prior work of implementing twin precision in MB algorithm are analysed. In the twin precision implementation, changes to be made for N and $N/2$ b multiplication are selected using multiplexers which increase the design complexity of the multiplier. Our proposed method decreases this complexity by utilising less multiplexers. Less multiplexers are utilised because when TP implementation is made in RCMB according to our proposed algorithm, lesser changes are to be made in partial product array for $N/2$ bit multiplication. We have achieved nearly 40% to 48% (for $N = 8$ to 128) of less multiplexers compare to previous work [2].

To our knowledge, TP implementation to signed multipliers is made in [2] and further optimization in TP multiplier is not done. Compared to prior work [2], our proposed algorithm implementation requires less change for performing $N/2$ multiplication. These

Table 4 Mux utilisation for SWP and TP

S number	Bit width	Proposed twin mux	Mux count for multiple SWP-($N/2 + N/4$)
1	16	11	27
2	32	19	52

changes or selection of appropriate partial products are typically selected using multiplexers. Table 1 illustrates the signal selection that has to be made for $N/2$ multiplication in MB. TP implementation in MB contains $(N/2) + 1$ partial product rows and adopts sign extension scheme presented in [16]. TP implementation in MB which adopts the sign extension scheme in [16] requires that more changes are to be made for $N/2$ bit multiplication. Multiplexer utilisation is directly related to the changes to be made in partial product row for $N/2$ b multiplication, and for every change, the multiplexers are to be deployed to select required partial product for N and $N/2$ b multiplication.

For twin precision implementation in modified booth algorithm [2], the changes (steps) that have to be made for $N/2$ b multiplication apart from booth encoding and decoding are PLSB, and its potential carry (most significant part (aMSP)) has to be performed separately for $N/2$ LSB and MSB multiplication. And also sign extension has to be made separately for $N/2$ b multiplication. So these three steps have to be performed in $N/2$ LSB and MSB multiplication. In RCMB [11] apart from booth encoding and decoding, the two steps to be performed for $N/2$ b multiplication are neg bit added in the LSB of each partial product row to generate twos complement and 1 s added for sign extension. RCMB does not require PLSB and potential carry (less complexity) so this makes RCMB more suitable for TP implementation and utilises less multiplexers. For an 8×8 TP multiplier, twin 2009 [2] requires 11 multiplexers, and our proposed method requires only 7 multiplexer as consolidated in Table 3. Reduction in multiplexer occurs because only two steps have to be performed in RCMB $N/2$ b multiplication compared to [2] which performs three steps in $N/2$ b multiplication.

Our proposed method has been tested for various bit widths that are multiples of eight and results are compared with prior work [2] and tabulated in Table 4. From the analysis made for various bit widths, it is inferred that our method needs $(N/2) + 3$ multiplexers. Due to reduction in multiplexers, overall area, delay and power are reduced for TP multiplier. For the previous implementation of TP technique [2], the multiplexer utilisation has been formulated as $N + 3$. This reduction in multiplexer utilisation in our proposed work reduces the design complexity of TP multiplier.

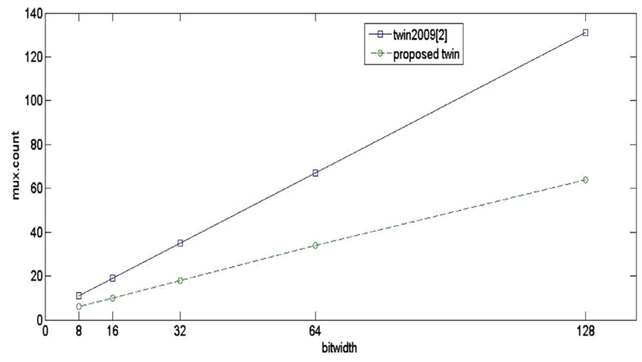
Figure 12 shows a graphical representation of Table 5. This table shows the comparison chart of implementation results of TP in MB (twin 2009) and RCMB (proposed twin) with non-TP multiplier. Figure 12a illustrates the utilisation of multiplexer (mux) in the implementation of TP for the proposed and existing method. A drastic decrease in the mux utilisation is much noticeable for higher bit width, i.e., from bit width of 16. Figure 12b,c,d portrays the changes in area, delay and power.

Cadence RTL compiler with TSMC library 180 nm is used to synthesized and analyse the cell area, delay and power. All the evaluation including power consumption is taken after post place and route synthesis of cadence digital flow. For higher bit width of multiplier, the reduction in area, delay and power are more noticeable for proposed twin compared with twin 2009 [2]. TP multiplier usually requires muxes for selection of partial products, so in Table 5, non-TP multiplier (TPM) results in less area compared to twin 2009 [2] and proposed twin. Usually, non-TP multiplier requires less area because of the absence of multiplexers, but the TP multiplier produces double output which is not possible in non-TP multiplier. Though non-TP multiplier produces less delay compared with TP multiplier, it cannot produce N or two $N/2$ b output at a time. And also increase in delay for TP is only 15% when compared to non-TP multiplier. While performing $N/2$ b multiplication in TP multiplier, the unwanted partial products are made to zero and this is the reason for achieving lesser power in TP multiplier compared to non-TP.

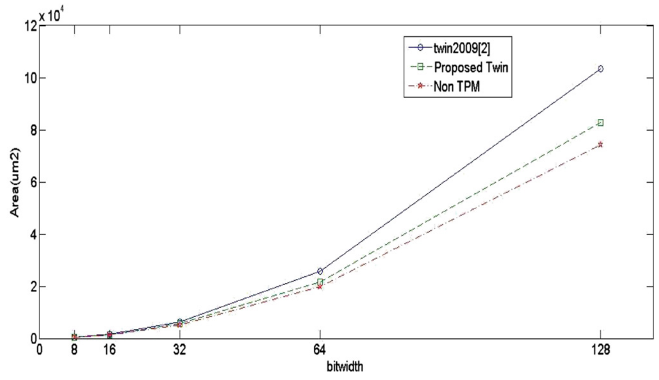
From Table 5, it is inferred that our TP implementation in RCMB achieves reduction in delay of about 5% to 18%, area of about 5% to 20% and power reduction of 8% to 32% due to reduction in mux utilisation. And the mux utilisation is drastically reduced up to 40% than the previous method [2]. When $N/2$ multiplication is performed, i.e., when multiplier performs narrow width operation, a significant reduction in power is achieved. Most of the partial products are made zero in $N/2$ b multiplication, so it leads to overall power reduction.

From the results in Figure 11, it is inferred that our implementation gives better performance compared with the twin 2009 [2] methodology.

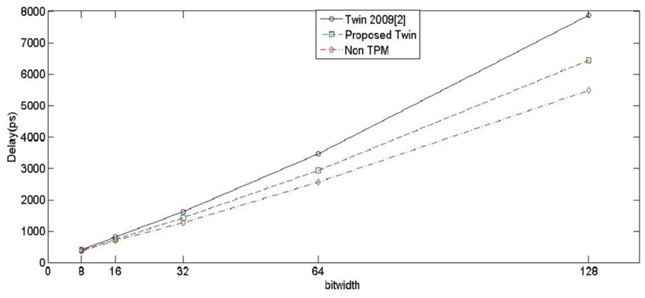
In hardware architectures, obtaining reconfigurable architecture in ASIC is more challenging. Through TP technique, double throughput is achieved in signed multipliers. In this study, we have proposed an optimised twin precision multiplier. To prove the effectiveness of the proposed multiplier, we have implemented the twin 2009 [2] multiplier and the proposed optimised TP multiplier in the FFT complex multiplication and analysis were made. Apart from the effective hardware utilisation, our proposed TP multiplier yields reduced area, power and delay compared to the twin 2009 [2] TP multiplier.



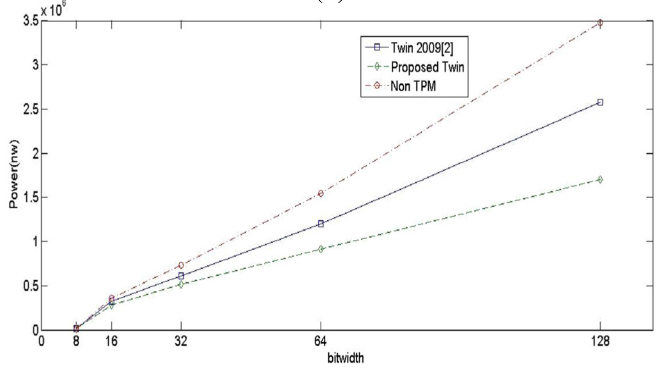
(a)



(b)



(c)



(d)

Figure 12 Graphical representation of Table 5. Comparison of existing and proposed TP; **(a)** bit width vs mux, **(b)** bit width vs area, **(c)** bit width vs delay and **(d)** bit width vs power.

Table 5 Comparison chart of existing TP multiplier, proposed TP multiplier and non-TP

	Non-TPM					Twin 2009 [2]					Proposed twin				
Bit width	8	16	32	64	128	8	16	32	64	128	8	16	32	64	128
Mux	-	-	-	-	-	11	19	35	67	131	7	11	19	35	67
Area (μm^2) (cells)	398	1,385	5,247	19,706	74,390	444	1,603	6,412	25,780	103,320	412	1,458		216,552	90,836
Delay (ps)	359	692.27	1,273.8	2,553.5	5,481.65	394.6	818.5	1,627.4	3,453.8	7,865.3	378.2	7438	1,415.4	2,935.05	6,449
Power (nW)	13,051.8	362,256.4	731,984.4	1,548,441.2	3,478,721.9	12,198	320,581	609,987	1,200,342	2,576,831	11,222	280,212	5,123,891	912,259	1,700,708

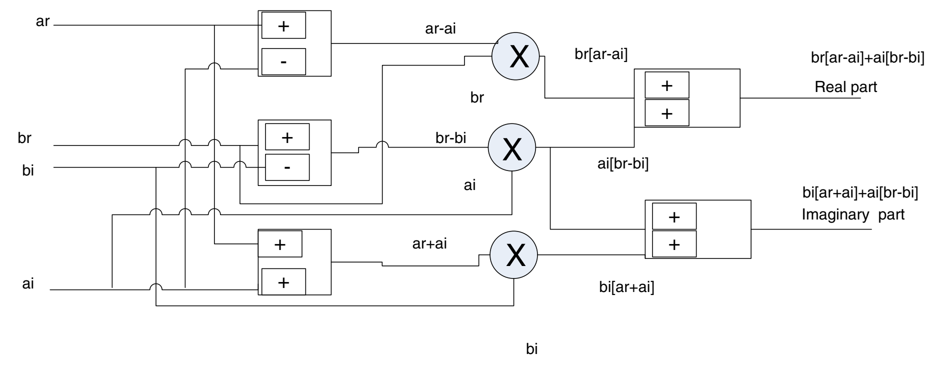


Figure 13 Complex multiplication in FFT.

The complex multiplier in FFT [20] is realised using four real multipliers, one adder and one subtractor as per Equation 8 as shown in Figure 13. In VLSI implementation, the complex multiplier occupies majority chip area. So according to Equation 9, complex multiplication can be realised by only three multipliers.

$$(a_r + ja_i)(b_r + jb_i) = (a_r b_r - a_i b_i) + j(a_i b_r + a_r b_i) \tag{8}$$

$$(a_r + ja_i)(b_r + jb_i) = \{b_r(a_r - a_i) + a_i(b_r - b_i)\} + j\{b_i(a_r + a_i) + a_i(b_r - b_i)\} \tag{9}$$

First, our analysis is made by replacing three multipliers in the complex multiplication in Figure 13 as TP multiplier. Second, the experiment was conducted with a 32-bit TP multiplier. So in a single 32-bit multiplier, two 16-bit multiplications ($N/2$ multiplication) or one 32-bit multiplication (N bit multiplication) can be performed. During N bit multiplication, the data width is 32 bit and twiddle factor is also 32 bit whereas in $N/2$ multiplication, data width is 16 bit and twiddle factor is 16 bit. When the three 32-bit multiplier in Figure 13 is replaced by TP multiplier as per TP logic, it can perform one 32-bit multiplication or two 16-bit multiplications.

While performing $N/2$ operation since the data and twiddle factor bit width are 16, the $b_r(a_r - a_i)$ and $a_i(b_r - b_i)$ can be performed in one single TP multiplier and in another multiplier $a_i(b_r - b_i)$, and $b_i(a_r + a_i)$ is performed and one multiplier can be left free. So this gives a way to overall power reduction when $N/2$ multiplication is performed or multiplier is operated in narrow width. The results are tabulated in Table 6. All the results are synthesized in cadence RTL compiler with TSMC 180 nm library. The inferences from the results in Table 6 are the percentage of improvement in area is 16%, delay of 17% and power improvement of 22%. Our proposed method gives better performance because it utilises less multiplexers compared to the existing TP multiplier.

6 Conclusions

Double throughput in ASIC environment is achieved effectively by implementing TP technique in RCMB, and a suitable algorithm is proposed. Our implementation requires less changes in the partial product array to acquire TP. Depending upon the multiplication either N or $N/2$ bit, selection of appropriate partial products in TP multiplier is done using multiplexers. Since our implementation utilised less multiplexers of about $(N/2) + 3$, the overall delay, area and power are reduced compared to prior implementation of TP technique in MB algorithm. Our proposed TP implementations consume 40% to 50% (for $N = 8$ to 128%) of less multiplexers. And our implementation gives better performance (area, delay and power) compared to prior implementation of TP in MB algorithm. To test the efficiency of the system, our proposed TP multiplier is implemented in FFT complex multiplication and its results gives better performance for our approach.

Table 6 TP in complex multiplication

Parameter	Twin 2009 [2] multiplier	Proposed TP multiplier (proposed twin)
Data width	32	32
Area (μm^2)	20,186.47	16,956.99
Delay (ps)	5,123	4,252
Power (nW)	1,867,221.23	1,456,432.67

Competing interests

The authors declare that they have no competing interests.

Authors' information

A. Rosi received her Bachelor of Engineering in Electronics and Communication and Master of Engineering degree in Applied Electonics from Anna University, India, in the year 2007 and 2009, respectively. Since 2010, she has been a senior project assistant in the project funded by the Ministry of Earth Science - India. She is currently pursuing Ph.D in VLSI Architecture at Anna University, India.

Dr. R. Seshasayanan was born in the year 1958 in India and received his B.E. degree from the College of Engineering, M.E. degree from Anna University in the year 1980 and 1983, respectively. He received his Ph.D from Anna University. He is presently working as an Associate Professor in the Department of Electronics and Communication, Anna University, and his area of interests are Low Power VLSI Design and Reconfigurable Architectures for Image processing. He is actively involved in various projects funded by the Ministry of Earth Science and Ministry of Defence - India.

Received: 1 March 2014 Accepted: 13 January 2015
Published online: 04 March 2015

References

1. M Själander, H Eriksson, P Larsson-Edefors, An efficient twin- precision multiplier, in *Proc. 22nd IEEE Int. Conf. Comput. Des*, 2004, pp. 30–33
2. M Själander, P Larsson-Edefors, Multiplication Acceleration Through Twin Precision, in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 17*, 9th edn., 2009
3. Differences between FPGA and ASIC. http://www.xilinx.com/fpga/asic.htm.
4. About ASIC and FPGA. http://www.altera.com
5. GH Loh, Exploiting data-width locality to increase superscalar execution bandwidth, in *Proc. 35th Int. Symp. Microarchitecture*, 2002, pp. 395–405
6. D Brooks, M Martonosi, Dynamically exploiting narrow width operands to improve processor power and performance, in *Proc. 5th Int. Symp. High Perform. Comput. Arch*, 1999, pp. 13–22
7. S Krithivasan, MJ Schulte, Multiplier Architectures for Media Processing, in *IEEE Conference*, 2003
8. S Khan, E Casseau, D Menard, Reconfigurable SWP operator for multimedia processing, in *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, 2009
9. A Danysh, D Tan, Architecture and Implementation of a vector/SIMD multiply-accumulate unit, in *IEEE Transaction on Computers, vol. 54*, 2005, pp. 284–293
10. AD Booth, A signed binary multiplication technique, in *Quarterly J. Mechanical and Applied Math, vol. 4*, 1951, pp. 236–240
11. F Lamberti, N Andrikos, E Antelo, P Montuschi, Reducing the Computation Time in (Short Bit-Width) Two's Complement Multipliers, in *IEEE Transactions On Computers, vol. 60*, 2nd edn., 2011
12. OL MacSorley, High speed arithmetic in binary computers. Proc. Inst. Radio Eng. **49**(1), 67–97 (1961)
13. Z Huang, MD Ercegovac, High-Performance Low-Power Left-to-Right Array Multiplier Design. IEEE Trans. Comput. **54**(3), 272–283 (2005)
14. R Zimmermann, DQ Tran, Optimized Synthesis of Sum-of- Products, in *Proc. Conf. Record of the 37th Asilomar Conf. Signals, Systems and Computers, vol. 1*, 2003, pp. 867–872
15. W-C Yeh, C-W Jen, High-speed Booth encoded parallel multiplier design. IEEE Trans. Comput. **49**(7), 692–701 (2000)
16. J Fadavi-Ardekani, M N Booth encoded multiplier generator using optimized Wallace trees, in *IEEE Trans. Very Large Scale Integr. (VLSI) Syst, vol. 1*, 2nd edn., 1993, pp. 120–125
17. H Eriksson, P Larsson-Edefors, M Sheeran, M Själander, D Jo-hansson, M Schölin, Multiplier reduction tree with logarithmic logic depth and regular connectivity, in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2006, pp. 4–8
18. MD Ercegovac, T Lang, *Digital Arithmetic*. Morgan Kaufmann Publishers - An Imprint of Elsevier, 2004
19. JY Kang, JL Gaudiot, A Fast and Well-Structured Multiplier, in *Proc. Euromicro Symp. Digital System Design*, 2004, pp. 508–515
20. YS Algnabi, R Teymourzadeh, M Othman and S Islam. FPGA Implementation of Pipeline Digit-Slicing Multiplier-Less Radix 22 DIF SDF Butterfly for Fast Fourier Transform Structure, Institute of MicroEngineering and Nanoelectronics IMEN, VLSI Design Department, Malaysia, in *The 5th European conference on antennas andpropagation (EUCAP2011)*, pp 4168–4172.