International Journal of Electrical, Electronic and Communication Sciences
ISSN: 2517-9438
Vol:4, No:3, 2010

# A High-Speed Multiplication Algorithm Using Modified Partial Product Reduction Tree

P. Asadee

***Abstract*—**Multiplication algorithms have considerable effect on processors performance. A new high-speed, low-power multiplication algorithm has been presented using modified Dadda tree structure. Three important modifications have been implemented in inner product generation step, inner product reduction step and final addition step. Optimized algorithms have to be used into basic computation components, such as multiplication algorithms. In this paper, we proposed a new algorithm to reduce power, delay, and transistor count of a multiplication algorithm implemented using low power modified counter. This work presents a novel design for Dadda multiplication algorithms. The proposed multiplication algorithm includes structured parts, which have important effect on inner product reduction tree. In this paper, a 1.3V, 64-bit carry hybrid adder is presented for fast, low voltage applications. The new 64-bit adder uses a new circuit to implement the proposed carry hybrid adder. The new adder using 80 nm CMOS technology has been implemented on 700 MHz clock frequency. The proposed multiplication algorithm has achieved 14 percent improvement in transistor count, 13 percent reduction in delay and 12 percent modification in power consumption in compared with conventional designs.

***Keywords*—**adder, CMOS, counter, Dadda tree, encoder.

## I. INTRODUCTION

THERE are two kinds of multiplication algorithms, serial multiplication algorithms and parallel multiplication algorithms [1,2]. Serial multiplication algorithms use sequential circuits with feedbacks. In serial multiplication algorithms, inner products are sequentially produced and computed. Parallel multiplication algorithms often use combinational circuits, and do not contain feedback structures [3,4]. There are two well-known kinds of parallel multiplication algorithms, array multiplication algorithms and Dadda multiplication algorithms. In array multiplication algorithms, cells, which consist of an AND-gate computing inner products and a counter, are put in a network pattern like an array. Dadda multiplication algorithms use AND-gates, carry save counters and a carry propagate adder (CPA) [5,6]. Dadda multiplication algorithms have a tree structure. While Dadda multiplication algorithms work at higher speed than array multiplication algorithms, Dadda multiplication algorithms have more complicated structure than array multiplication algorithms [4]. Multiplication is the fundamental arithmetic operation important in several processors and digital signal processing systems. Digital signal processing systems need multiplication algorithms to implement DSP algorithms such as filtering where the multiplication algorithm is directly within the critical path [4]. Therefore, the demand for high-speed multiplication algorithms has become more important. The higher speed results to enlarged power consumption, thus, low power architectures will be the choice of the future. This has given way to the growth of new circuit algorithms, with the plan of reducing the power consumption of multiplication algorithms with having high-speed structures and appropriate performance [7,8,9].

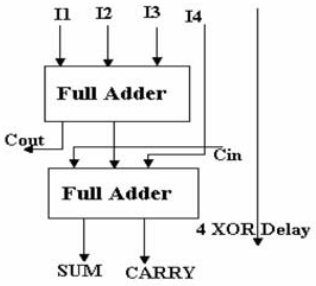


Fig. 1 Four to two counter using full adders

The proposed structured Smith multiplication algorithm uses a recognition component, a radix-4 Smith encoder, and a structured inner product producer component, 16-bit high-speed Dadda-trees for inner product reduction, a 4-bit counter tree and a final carry-hybrid adder. The recognition component generate the efficient signals of the input data, appropriately selects the multiplication algorithm and multiplicand operands, and then produces control signals to disable the units of the Dadda-trees and the counter-tree to match the desired data precision. Based on the number of multiplication algorithm bits, the Smith encoder and inner product generator calculate the number of inner products produced, while maintaining the unused inner product generator segments in the static state. The control and enable signals produced by the recognition component choose either the Dadda-tree or the 4-bit counter-tree for a given single multiplication operation. Therefore, the units of the unused circuits are able to continue their static condition. In the static condition, the previous values are held, to avoid any switching from happening in the unused part of the structure. To take advantage of short accuracy, signal gating can selectively turn

P. Asadee is with the Islamic Azad University Varamin branch (e-mail: asadee2@gmail.com).

off those parts of the Dadda-trees and 4-bit counter-tree, which are not currently in use, and make the proposed Smith multiplication algorithm perform like a changeable size multiplication algorithm. In the final step, the product is generated from the outputs of the active Dadda-tree and carry-hybrid adder. Thus, the Smith encoder, inner product generator and carry-hybrid adder can be shared and use again to calculate 16 or 8-bit multiplications. For 4-bit multiplications, the Smith encoder and inner product generator components are unused and turned off. The proposed Smith multiplication algorithm is implemented with five pipeline steps, with an input enable signal being used as a power-reduction part for each step.

## II. Four to Two Counter and Full-Adder

The four to two counter organization in fact adds five inner products bits into three. The structure is connected in such a way that four of the inputs are coming from the same bit place of the weight j while one bit is come from the adjacent place j-1 (recognized as carry-in). The outputs of four to two counter uses one bit in the place j and two bits in the place j+1. This architecture is named counter since it adds four inner products into two (while using one bit exactly connected between adjacent four to two counters). The block diagram of four to two counter can also be built using 3-2 counters. It consists of two 3-2 counters (full adders) in series and involves a critical path of 4 XOR delays as shown in Fig. 1. Counters are the fundamental building blocks in all the multiplication algorithm components. Hence using fast and efficient full counters plays an important role in the performance of the total system. In the following section, we describe the counter units used in our design.

The 28 transistor counter is the CMOS traditional adder circuit. This counter unit is built using equal number of N-MOS and P-MOS transistors. The logic for the complimentary MOS logic was realized using

$$C_{out} = AB + BC_{in} + AC_{in} \quad (1)$$

$$Sum = ABC_{in} + (A + B + C_{in})\overline{C_{out}} \quad (2)$$

The first 12 transistors of the circuit generate $C_{out}$ and the remaining transistors generate the Sum outputs. Thus, the delay for computing $\overline{C_{out}}$ is added to the total propagation delay of the Sum output. The structure of this counter circuit is very large and therefore uses large on-chip area.

The stationary counter circuit was implemented using new logic and reduced number of transistors. The basic idea in the stationary counter is the use again of charge stored in the load capacitance during the high output to drive the control logic. In regular counter designs, the input charge applied at logic high will be drained off during logic low form. This is achieved by using only one voltage source ($V_{DD}$) in the circuit. As an added advantage, there will be no path from one voltage level ($V_{DD}$) to the other (GND). The removal of the direct path to the ground eliminates the short circuit power unit for the counter structure. This reduces the total power

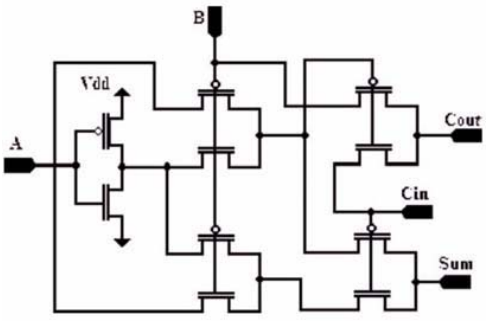

Fig. 2 A novel full adder

consumed in the circuit and making it an energy efficient implementation. The stationary counter is not only power optimized but also area efficient due to its transistor count. The main disadvantage of the stationary counter is the threshold voltage drop at the output voltage for certain input structures. A detailed relative study of this counter with other low power counters can be found in [9,10,11].

In the counter unit, the design of XOR and XNOR of A and B is done using pass transistor logic and an inverter is to invert the input signal. This design results in faster XOR and XNOR outputs and ensures that there is a stability of delays at the output of these gates. This results to less false SUM and $C_{out}$ signals. The capacitance at the outputs of XOR and XNOR gates is also decreased, as they are not loaded with inverter. Since, the signal reduction at the SUM and $C_{out}$ is important for sub-micron circuits, drivers can be used to decrease the degradation. The driver will help in producing outputs with equal rise and fall times. This consequences in better performance concerning speed, low power consumption and driving capabilities. The output voltage swing will be equal to the $V_{DD}$, if a driver is used at the output. Fig. 2 gives the circuit level diagram of counter. A detailed comparative study of presented counter with other low power counters can be found in [12].

## III. Smith Encoder

The recognition component uses the effective input data, and then produces the control signals that are used to turn off the appropriate units of the Dadda-tree and counter-tree, to match the data precision. In the presented multiplication algorithm, the control signals not only select the data flows, but also calculate the pipeline register components, in order to maintaining the non-effective bits in their previous states and, thus, ensure that the functional components addressed by these data do not consume switching power. In addition, these control signals are used to control the Dadda-trees and counter-tree. Fig. 3 shows the functional blocks of the active-range recognition unit that includes one-detect circuit (OR gate), comparator, multiplexers and logic gates. The OR gate is used, which determines the output to be high if any of the inputs are high. Both the 16-bit wide multiplication algorithm and the multiplicand input operands are partitioned into three parts, where the detection is done for the 15-bit, 8-bit and 4-

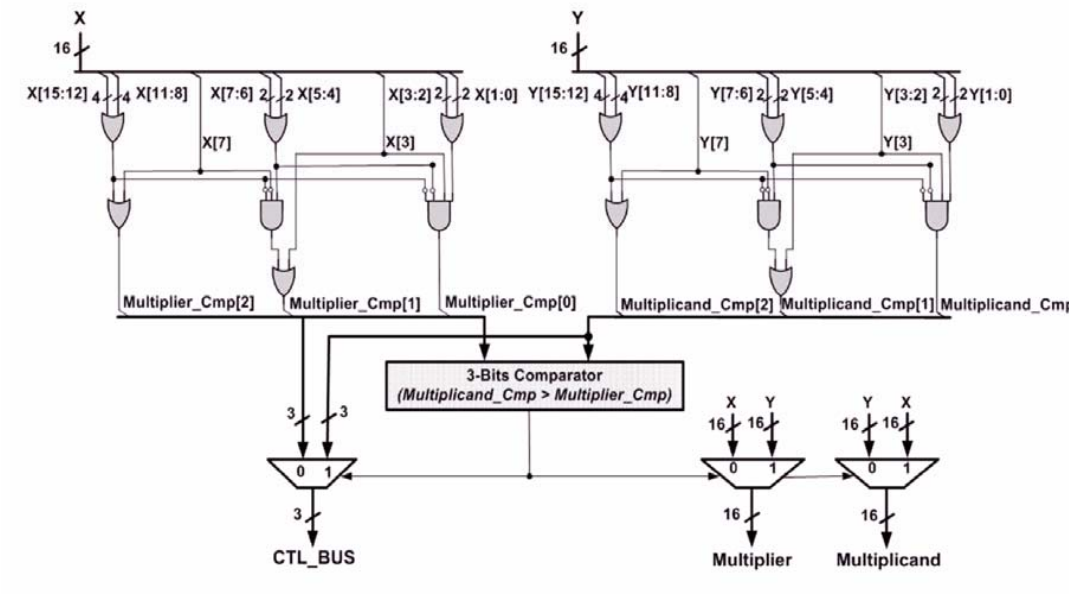


Fig. 3 Recognition component

bit ranges. The presented multiplication algorithm supports three multiplication forms that are 15-bit, 8-bit and 4-bit multiplication forms. Both the 16-bit and 8-bit multiplication forms are implemented using Smith multiplication algorithm values greater than 8F h-decimal utilize the 16-bit multiplication form. This problem is solved in the 4-bit multiplication form by using a 4-bit unsigned counter-tree. The output of the three circuits of both the multiplication algorithm and multiplicand are grouped into a 3-bit line, where the most significant bit shows the 16-bit multiplication form, the second bit shows the 8-bit multiplication form and the least significant bit shows the 4-bit multiplication form. The two 3-bit buses are compared using a 3-bit comparator. This comparison detects if the input multiplicand operand is greater than the input multiplication algorithm operand. The output of the comparator is used to switch the input multiplicand and multiplication algorithm operands.

The radix-4 smith encoder can generate five possible values of -2, -1, 0, 1 and 2 times the input [5]. Three control signals $COMP_j$, $SHIFT_j$ and $ZERO_j$ (j=0,…,7) are produced depending on the 3-bit coding method shown in the Smith-coding table [4]. The Smith encoder is used to produce these control signals, which are used in the inner product generation component to direct appropriate operation ($OP_i$, i=0,…,7) on the input multiplicand operand. Fig. 4 shows the inner product generator component, which is designed to be shared between the 16-bit and 8-bit multiplication forms. The ZERO signal is used to output zeros as output of that inner product step, the COMP complements the input multiplicand operand, and finally the SHIFT signal shifts the input multiplicand operand left by one. The total numbers of inner products (PP) produced are N/2 (N=max. number of multiplication algorithm bits), where $PP_i=OP_i.M_i$ ($M_i$=ith-bit of the multiplicand, i=0,…,15). The 8-bit multiplication form only needs first four of the inner products. Thus, in the pipeline step between the Smith encoder and the inner product generator component, all the necessary control signals are produced to turn on (turn off) the circuit in use (not in use). There are three types of arrangement forms for the inner product, which are steady with the operation forms for the inner product, which are consistent with the operation forms of the low-power multiplication algorithm. In the 16-bit multiplication form all eight of the inner product ($PP_j$, where j=0,…,15) producer rows are active and in the 8-bit multiplication form only the first four of the inner product ($PP_j$, where j=0,…,7) producer rows are active, and the rest of the circuit is in static state. Finally, in the 4-bit multiplication form all eight of the inner product producer rows are put in the steady state.

## IV. Modified Dadda tree architecture

In this section, we present the proposed design for Dadda multiplication algorithms. The proposed multiplication algorithm contains multiple parts. If no or one part is faulty, the proposed multiplication algorithm works correctly. We divide an n×n Dadda multiplication algorithm into 2n parts. We show the presented design for Dadda multiplication algorithms. Dadda multiplication algorithm uses a carry chain counter. As is well known, there are several kinds of carry chain counters. We use the simplest kind of carry chain counters, ripple-carry counters (RCCs). Dadda multiplication algorithms using other kinds of carry chain counters are discussed. Some improvements of the proposed design are also presented [5].

Here, we divide an n×n Dadda multiplication algorithm into

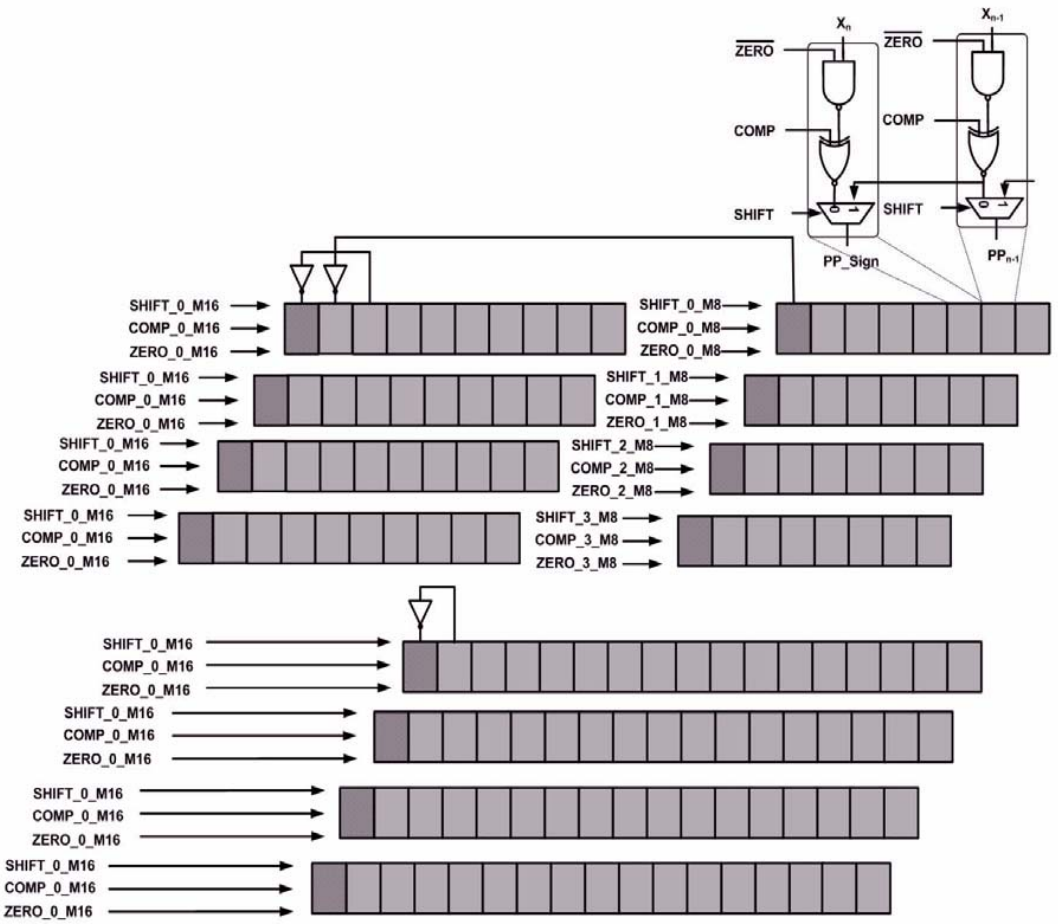


Fig. 4 Inner product generator

2n parts $SL_i$ ( $0 \le i<2n$ ). What gates and counters belong to which parts are recursively decided as described below: 1) An counter whose sum output is the i-th part belongs to $SL_i$. 2) An counter whose sum output is connected to an counter which belongs to $SL_i$ also belongs to $SL_i$. 3) An AND gate whose output is connected to an counter which belongs to $SL_i$ also belongs to $SL_i$. Note that the carry output of an counter which belongs to $SL_i$ is always connected to $SL_{i+1}$. Fig. 5 shows eight 4×4 Dadda multiplication algorithm. In this figure, the dotted arrows mean carry outputs of carry save counters. Fig. 5 shows the 4×4 Dadda multiplication algorithm divided into eight parts. Each part consists of AND-gates and counters.

The counters and wires in every part make the same tree structure as the Dadda multiplication algorithm shown in Fig. 5 except two difference discussed in the next paragraphs.

For example, in $SL_3$, the counters $FA_{1,3}$, $FA_{2,3}$ and counter $HA_{3,3}$ correlate to the carry save counters $CSA_1$, $CSA_2$ and the carry propagate counter $CPA_3$, respectively. The inner product A[0]B[3], A[1]B[2], A[2]B[1] and A[3]B[0] correlate to AB[3], AB[2], AB[1] and AB[0], respectively. The third bit of product Y[3] correlates to the product Y. One of the differences is the following. For most parts $SL_i$, the Dadda multiplication algorithm uses some counters and wires which do not correlate to any counters and wires in the parts $SL_i$. For example, the fifth part $SL_5$ does not use wires for inner products correlating to AB[1] and AB[0] as well as counters correlating to $CSA_1$.

The other difference is that carry output of the counters in a part $SL_i$ are connected to not a part of the part $SL_i$ but a part of its neighbor part $SL_{i+1}$. For example, the carry outputs of $FA_{1,3}$ and $FA_{2,3}$ in $SL_3$ are connected to $FA_{2,4}$ and $FA_{3,4}$ of $SL_4$ and not counters in $SL_3$. The proposed n×n Dadda multiplication algorithm consists of 2n parts $SL'_i$ ( $0 \le i \le 2n-1$ ) and a part $SL'_R$. The i-th part $SL'_i$ ( $0 \le i \le 2n-2$ ) can play the role of both $SL_i$ and $SL_{i+1}$. The (2n-1) part $SL'_{2n-1}$ plays the role of only $SL_{2n-1}$, and the part $SL'_R$ plays

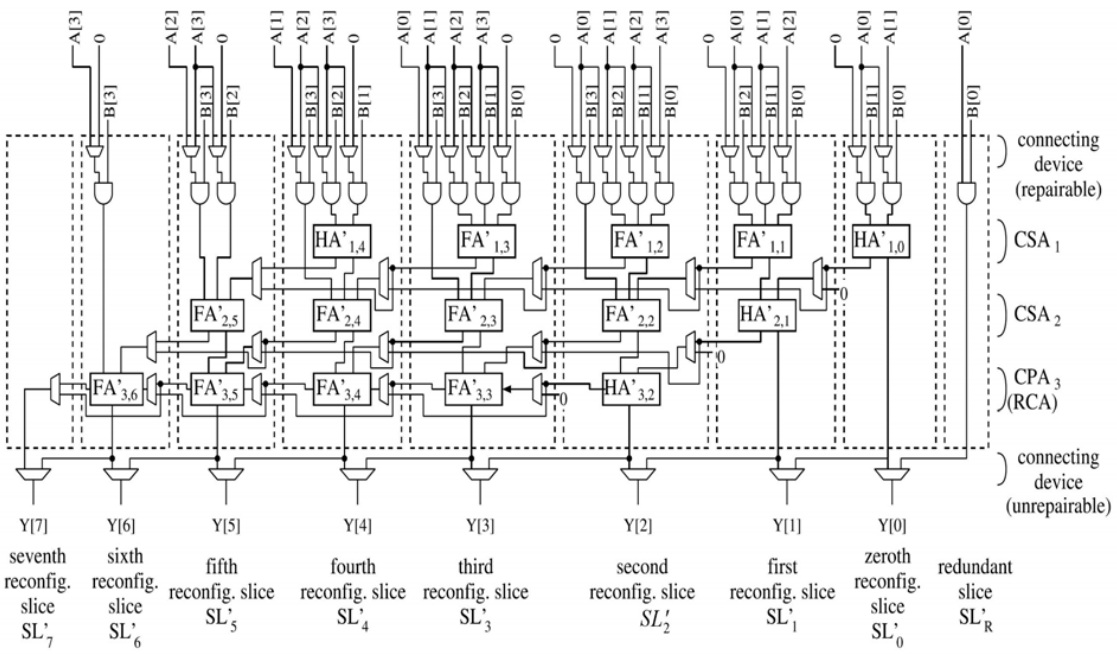


Fig. 5 Dadda multiplier divided into eight parts.

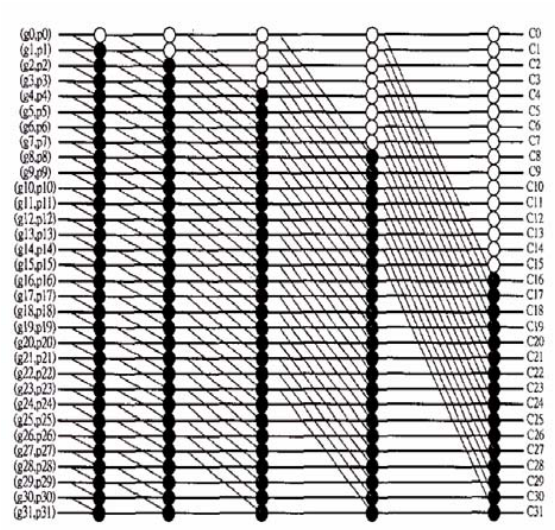

Fig. 6 CHA array architecture

the role of only $SL_0$. Fig. 5 shows an example of the presented 4×4 Dadda multiplication algorithm. The multiplication algorithm contains eight arrangement parts and a redundant part. The arrangement parts use AND gates, counters and switches. The control signals for switches are implemented by high-speed circuits. In the proposed multiplication algorithm, arrangement is made in the chain method, that is, if $SL'_k$ is faulty, $SL'_R$ plays the role of $SL_0$, and $SL'_i$ ($\forall i < k$) arrangement part plays the role of $SL_{i+1}$. The arrangement part $SL'_j$ ($\forall j > k$) plays the role of $SL_j$.

The AND gates in $SL'_i$ (0<=i<2n-1) must be capable of calculating not only the inner products A[x]B[y] (x+y=i, $0 \le$ x,y<n) but also the inner products A[x+1]B[y] (x+y+1=i, $0 \le$ (x+1),y<n). For those calculations, switches are inserted between the input A and the AND gates.

Similarly to the AND gates, counters and wires in the parts of the Dadda multiplication algorithms, also make the same tree structure as the non-Dadda multiplication algorithms but presented circuit makes high-speed structure. As shown in Fig. 5, some counters are full adders and others are half adders in different parts. The kind of the counter $A_{j,i}$ correlating to a carry save/propagate counter $CA_j$. No counter in $SL'_i$ and $SL_{i+1}$ do not correlate to $CA_j$. In the example shown in Fig. 5, $FA'_{1,1}$ correlating to $CSA_1$ in $SL'_1$ would be a counter because the correlating counter $FA_{1,2}$ in $SL_2$ is a counter. An counter $HA'_{2,1}$ is a half adder because the correlating counter $HA_{2,2}$ in $SL_2$ is a counter and any counters in $SL_1$ and $SL_2$ do not correlate to $CSA_2$. No counter in $SL'_1$ correlates to $CPA_3$ because no counter in $SL_1$ and $SL_2$ correlates to $CPA_3$. No wires for inner products and sum outputs in $SL'_i$ correlates to a data path in the Dadda multiplication algorithm if and only if any wires in $SL_i$ and $SL_{i+1}$ do not relate to the data path.

The carry inputs of $SL'_i$ (0<=i<=2n-1) are connected back to not only $SL'_{i-1}$ but also $SL'_{i-2}$ through switches selecting those carry signals. It is because $SL'_i$ uses carry signals output from $SL'_{i-2}$ if $SL'_{i-1}$ is faulty while it usually uses carry signals output from $SL'_{i-1}$. In the example shown in Fig. 5, the

counter $FA'_{2,3}$ in $SL'_3$ is linked back to the carry output of both $FA_{1,1}$ and $FA_{1,2}$ throughout a switch. The i-th bit of the product Y[i] is calculated in either $SL'_i$ or $SL'_{i-1}$. So, switches using the output signals are linked to those outputs.

As $SL'_R$ plays the role of only $SL_0$, the building of $SL'_R$ is the same as that of $SL_0$. In the similar fashion, the construction of $SL'_{2n-1}$ has switches selecting the carry inputs. Note that switches selecting multiplicand A and carry signals are repairable because the false switches in $SL'_i$ can repaired by deleting $SL'_i$. Switches using output signals are not fixable.

## V. Carry Hybrid Structure

Minimizing the critical path is the most usual way to decrease the propagation time [5]. Therefore, we use a method built with an extremely compact array of cells, each implementing the "*" operator. Its critical path down to $\log_2 n$ logical levels while keeping the fan-out to two for each unit. A 64-bit CHA is presented for high-speed and low-power multiplication algorithms. Since the speed of a CHA adder mainly relates on the speed of carry propagation chain. In order to speed up the generation of the carry chain and obtaining low power, low voltage logic, was used to implement the high-speed, low-power CHA adder. As shown in Fig. 6 the 64-bit carry hybrid adder has parallel structure. The key element of the 64-bit CHA was "*" operator. The "*" operator algorithm was shown as follows:

$$(g_i, p_i) * (g_j, p_j) = (g_i + p_i g_j, p_i p_j) \quad (3)$$

g=a*b and p=a$\oplus$b if a, b were input signal. The "o" cells were similar to the "*" ones as shown in Fig. 6, but are only used as buffers in order to make the signal propagation consistent across the adder. Each black unit is the "*" operator and the various circuits of the black unit were designed. By using the N-P logic components, the 64-bit adder can be designed as a pipelining structure.

As mentioned earlier, the CHA units were used to implement the "*" operator in the 64-bit CHA. In order to make the carry propagation chain have the critical delay path, we input the pipeline signal $(A_{31}A_{30}...A_1A_0) + (B_{31}B_{30}...B_1B_0)$ as follows: (000…00)+(111…11) and (111…11)+(000…01). The 64-bit CHA adder using 80 nm CMOS technology with 1.3V power supply. Due the CHA units, operation speed comparison results show the new circuit has the speed advantage over the conventional circuit. The new 64-bit CHA adder could be operated on 700 MHZ clock frequency with 1.3V power supply and the conventional adder could not operate on 700 MHz. It was just about 500 MHz. The maximum operation frequency and power consumption comparison results are calculated, and the power consumption is calculated under maximum operation frequency. Note that the normalized Power freq. is also given. It shows that the new circuit has less power consumption under same operation frequency.

TABLE I
COMPARISON BETWEEN 64×64 BIT TREE MULTIPLIERS

| Multipliers | Present study | [6,7,8] | [9,10,11,12] |
|---|---|---|---|
| Technology ( $nm$ ) | 80 | 80 | 80 |
| Transistor counts | 31169 | 36245 | 38654 |
| Multiplication time (ns) | 4.5 | 5.3 | 6.8 |
| Chip Area ( $mm^2$ ) | 0.69 | 0.94 | 0.85 |
| Power Diss. (mW/MHz) | 0.67 | 0.85 | 1.19 |

nm: Nanometer; ns: Nano second; MHZ: Mega hertz

## VI. Conclusion

In this paper, we presented a novel high-speed, low-power multiplication algorithm. Three important modifications have been implemented in inner product generation step, inner product reduction step and final addition step. The new functional units, together of optimized Dadda-tree were used to design the proposed Smith multiplication algorithm. The presented multiplication algorithm has better power-consumption characteristics, and thus be more power efficient than other multiplication algorithms. In this work, the low-voltage and high-speed 64-bit CHA was designed and implemented. The internal delay and power considerations reduce voltage swing scheme. Based upon the SPICE simulation results, the 64-bit CHA has the speed and power dissipation advantages over the conventional adder. This paper has proposed a novel design for Dadda multiplication algorithms. A high-performance algorithm for the proposed design has also been shown. The presented design has been evaluated from the aspects of the area, power and delay time. The new multiplication algorithm has achieved 14 percent improvement in transistor count, 13 percent reduction in delay and 12 percent modification in power consumption in compared with conventional designs. Table I shows comparison between present study and other designs.

## References

[1] R.P. Brent and H.T. Kung, "A regular layout for parallel adders", IEEE Trans. Comput., vol. C-31, pp. 260-264, Mar. 1982.
[2] A. Goldovsky, B. Patel and B. Schulte, "Design and implementation of a 16 by 16 low-power two's complement multiplier", ISCAS 2000 Geneva, vol. 5, pp. 345-348.
[3] G. Goto., et. al., "A 4.1-ns compact 54*54-b multiplier utilizing sign-select Booth encoders", IEEE J. Solid-State Circuits, vol. 32, pp. 1676-1682, Nov. 1997.
[4] X. Huang, et. al., "High-performance VLSI multiplier with a new redundant binary coding", Journal of VLSI Signal processing, vol. 3, pp. 283-291, Oct. 1991.
[5] P. Giopeen, "Novel and Efficient four to two counters", ISCAS 2000 Geneva, vol. 5, pp. 312-319.
[6] D. Kudeeth., "Implementation of low-power multipliers", Journal of low-power electronics, vol. 2, 5-11, 2006.
[7] K.H. Ching, "A novel carry-lookahead adder", Proc. International symposium low-power electronics and design, 1998.
[8] H. Lee, "A High-Speed Booth Multiplier", ICCS 2002.
[9] T.Y. Tang, and C.S. Choy, "Design of self-timed asynchronous Booth's multiplier", in Proc. Asia South Pacific Design Automation Conf., Jan 2000, pp. 15-19.

[10] K. Numba and H. Itu, "High-speed design for Wallace multiplier", ISSCC 2003.
[11] Y.N. Ching, "Low-power high-speed multipliers", IEEE Transactions on Computers, vol. 54, no. 3, pp. 355-361, 2005.
[12] M. Sheplie, "High performance array multiplier", IEEE transactions on very large scale integration systems, vol. 12, no. 3, pp. 320-325, 2004.