A Reconfigurable Parallel Prefix Ling Adder with modified Enhanced Flagged Binary Logic

Soumya Ganguly, Abhishek Mittal, Syed Ershad Ahmed

Department of Electrical Engineering
Birla Institute of Technology and Science, Hyderabad Campus, Hyderabad-500078, India

Abstract— **This paper presents a Reconfigurable Parallel Prefix Ling Adder. The proposed design can be partitioned to perform as one 16 bit, two 8 bit and four 4 bit adders. We also propose a new architecture for Enhanced Flagged Binary Adder (EFBA) designs which reduces the delay of operation considerably. The new adders are, therefore, modifications of conventional Reconfigurable Carry Lookahead Adder (CLA) - EFBA arrangements. We estimate a reduction of 6.8% in the delay in the critical path with respect to traditional implementation of CLA-EFBA logic. The proposed architectures are compared with the previous architectures in literature and are shown to perform better.**

Keywords—Ling adder, reconfigurability, flagged addition, parallel prefix, carry lookahead, reconfigurable adder.

I. INTRODUCTION

The most fundamental operation carried out in Microprocessors and Digital Signal Processors is Addition. Thus, it is imperative that binary adders achieve high speed of operation, take up less area and consume less power dissipation.

The speed of binary addition in digital circuits is restricted by the time required to propagate a carry through the adder. The Ripple Carry Adders (RCA) are small but slow, the parallel Carry Look-ahead Adders (CLA)/ Carry Select Adders are often fast but large. A large variety of algorithms and implementations have been proposed for binary addition [1], [2]. Parallel-prefix adders are used when high operation speed is required [3]. Parallel Prefix adders provide an excellent basis for making an extensive range of design tradeoffs in terms of delay, area and power. Parallel-prefix adders are appropriate for VLSI implementation as they can be realized using simple cells and maintain regular connections between them.

Several variants of the Carry-Lookahead equations, like Ling carries [4], have been presented. These variants simplify carry computation and can lead to faster structures. The simplified form of Ling equations has been exploited for the design of multilevel block Carry Lookahead adders.

Different research works [5-6] proposed the formulation of the Ling recurrence as a prefix computation to obtain high performance parallel adder implementations. The use of the Ling recurrence as a prefix computation leads to slightly faster adders than existing Carry Lookahead Parallel Prefix implementations at the cost of increase in hardware than the corresponding standard Carry Lookahead Adders as was proposed in [2]. A simple methodology to implement the Ling scheme in a flagged prefix adder has also been shown in literature.

Reconfigurable adders have gained significance for their applications in real time processing of audio/video signals. There is a growing interest in datapath components that are capable of performing computations with variable operand size, e.g. adders capable of doing both N and N/2-b additions [1]. A reconfigurable structure allows us to provide a large number of resources that can be used in different ways by different applications. In other words, the high reconfigurability of the developed structure is achieved with negligible hardware overhead. A design for reconfigurable Carry Lookahead adder has been proposed in [7].

Flagged binary adders are widely used for floating point arithmetic, BCD addition and certain applications of image processing. These require either a unity increment/decrement to be performed on the sum, or addition of a constant to it. This is achieved by generating flag bits that are used to compute the final sum, and eliminates the need of an extra adder structure.

In the next section, we present preliminary information about the previous work on various adder architectures and discuss the flagged binary adders and Ling adders in detail. The proposed design of Reconfigurable Ling Adder with modified Enhanced Flagged Binary logic is presented in section III. Experimental results on the performance of our proposed designs when compared with existing architectures are given in section IV and conclusions are drawn in section V.

II. RELATED WORK

Much of the work on VLSI adders has focused on speeding up the critical path of the addition operation, that is, finding out the carry bits. Most of the fast adders are based on being able to calculate the carry propagation much faster without having to wait for it to ripple through each bit of the adders. The Carry Lookahead (CLA) technique is one such popular technique which calculates the carry-out for a block of bits in parallel to, and separately from, calculating the sum outputs of the block [7]. It uses bit generate and bit propagate signals and a set of recursive equations to calculate carry bits for each bit position.

The CLA technique is speeded up by implementing it in a prefix tree [3]. There are various families of prefix trees like Sklansky, Kogge-Stone, Brent-Kung, Han-Carlson, Ladner-Fischer and Knowles, each offering different tradeoffs in area, delay and complexity. The 8-bit Kogge-Stone [9] and Ladner-Fischer [10] adder structures that are used later in the paper to implement proposed designs are as shown in figure 1. Further, reconfigurability can be realized in a normal CLA [7] as well as in its prefix implementation. The design, however, modifies the prefix tree structure to achieve reconfigurability.

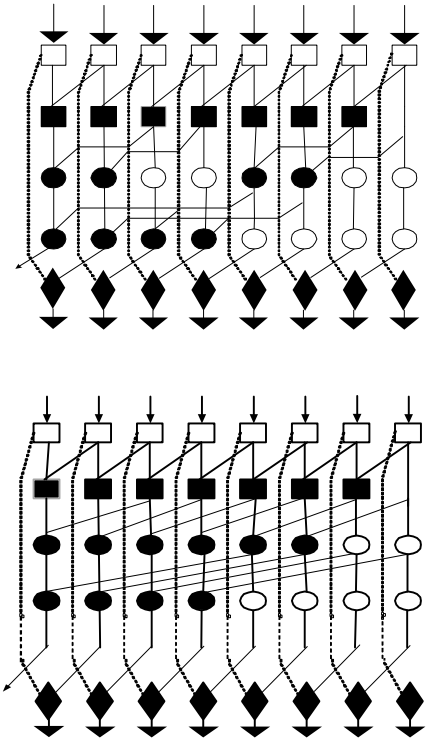

Figure 1: Ladner-Fischer and Kogge-Stone prefix structures

Ling [4] proposed a simplified form of carry Lookahead equations that rely on adjacent bit pairs (a_i ,b_i) and (a_{i-1} ,b_{i-1}). The i^{th} Ling carry H_i was defined in [4] as $H_i = c_i + c_{i-1}$. Using this equation, each H_i can be expressed as:

$$H_i = g_i + g_{i-1} + p_{i-1} . g_{i-2} + \ldots\ldots\ldots + p_{i-1} . p_{i-2} \ldots\ldots p_1 . g_0$$

The Ling carries H_i can be computed faster than the corresponding carries c_i since they rely on a simpler Boolean function. Consider, for example, the case of c_3 and H_3,

$$c_3 = g_3 + p_3 . g_2 + p_3 . p_2 . g_1 + p_3 . p_2 . p_1 . g_0$$
$$H_3 = g_3 + g_2 + p_2 . g_1 + p_2 . p_1 . g_0$$

Assuming the use of two-input logic gates, the calculation of c_3 requires four logic levels for the fastest implementation, while, for H_3, only three logic levels suffice.

Sum in a Ling adder is defined as:

$$S_i = H'_{i-1} \bullet d_i + H_{i-1} \bullet (d_i \text{ xor } p_{i-1}) \quad (1)$$

where H'_{i-1} is the complement of H_{i-1} , $d_i = a_i$ xor b_i and $p_{i-1} = a_{i-1} + b_{i-1}$. Though computation of the bits H_i is simpler, the derivation of the final sum bits S_i using the Ling carries is complicated compared to the case when carry lookahead carries are used.

A systematic methodology that allows the parallel-prefix computation of Ling carries was proposed in [5]. It defines the group generate and group propagate signals for a Parallel Prefix Ling adder as:

$$G_i^* = g_i + g_{i-1} \quad (2)$$
$$P_i^* = p_i . p_{i-1}$$

with $0 \le i \le n - 1$, $g_{-1} = p_{-1} = 0$, $G^*_k = P^*_k = 0$, for $k < 0$, $g_i = a_i.b_i$, $p_i = a_i + b_i$. Using them, we have written the Ling carries for an 8-bit adder as follows:

$$H_0 = G_0^* \quad (3)$$
$$H_1 = G_1^* \quad (4)$$
$$H_2 = G_2^* + P_1^* . G_0 \quad (5)$$
$$H_3 = G_3^* + P_2^* . G_1^* \quad (6)$$
$$H_4 = G_4^* + P_3^* . G_2^* + P_3^* . P_1^* . G_0^* \quad (7)$$
$$H_5 = G_5^* + P_4^* . G_3^* + P_4^* . P_2^* . G_1^* \quad (8)$$
$$H_6 = G_6^* + P_5^* . G_4^* + P_5^* . P_3^* . G_2^* + P_5^* . P_3^* . P_1^* . G_0^* \quad (9)$$
$$H_7 = G_7^* + P_6^* . G_5^* + P_6^* . P_4^* . G_3^* + P_6^* . P_4^* . P_2^* . G_1^* \quad (10)$$

Basic increment and decrement of the sum can be achieved by employing the concept of the flagged prefix adder. It was first proposed in [8] where a method was shown to generate flag bits that determine the inversion of sum bits (the result from the addition of two numbers) to generate appropriate results. In other words, flag bits are XORed with sum bits to get the incremented/decremented sum. Then [6] proposed enhanced flagged prefix adder (EFBA) wherein a method was shown to add two operands A, B and a constant M. It derived equations for flag bits, and subsequently for the final sum bits, from the equations of a full adder as follows:

The k^{th} sum bit in a full adder is calculated as:

$$R_k = S_k \text{ xor } M_k \text{ xor } d_k \quad (11)$$

Here R represents the final sum of A+B+M, S_k is the sum bit, M_k is constant bit and d_k is the carry in of k^{th} bit position.

The carry-out from the k^{th} bit position, d_{k+1} is calculated as:

$$d_{k+1} = S_k M_k + S_k d_k + M_k d_k, \quad (12)$$

Since M is known, we can substitute for each M_k in the expressions (11) and (12). Equation (12) gives

$$d_{k+1} = \begin{cases} S_k . d_k, & \text{if } M_k = 0 \\ S_k + d_k & \text{if } M_k = 1 \end{cases}$$

Similarly in (11), if M_k=0 then flag bit is same as carry-in. When M_k=1, then flag bit is taken as the invert of the carry-in.

$$F_k = \begin{cases} d_k, & \text{if } M_k = 0 \\ (d_k)', & \text{if } M_k = 1 \end{cases}$$

The flag bits F_k are finally XORed with the sum bit S_k to get the final sum R_k.

The computation of A+B+M can be illustrated assuming A=9, B=78, M =57:

A = 0000_1001
B = 0100_1110
M = 0011_1001
S = A+B = 0101_0111
F = 1100_0111
R = S XOR F = 1001_0000 = A+B+M

The EFBA architecture presented in [6] is accomplished via a modified ripple carry adder (RCA) to add the sum and a known constant as shown in figure 2. This leads to a linearly increasing delay with increase in operand size.

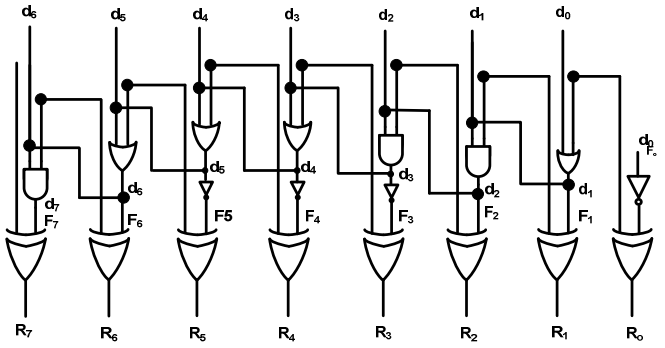

Figure 2: Enhanced Flagged Binary Adder block

III. PROPOSED WORK

A. Reconfigurability in Ling Adder

This section presents reconfigurability technique applied to parallel prefix Ling adders. We have chosen Ladner-Fischer structure for the even and odd trees of a Ling prefix tree [5] to explain reconfigurability, since it has less number of interconnections and black cells which makes it easier to illustrate the design. However, reconfigurability can be achieved using Kogge-Stone Ling prefix structure in a similar fashion. The figure 1 shows an 8 bit Ladner-Fischer and Kogge-Stone prefix structure tree. We will take an 8-bit adder first and then extend the argument to 16-bit and higher adders.

The Ling carry equations for an 8-bit adder are depicted in (3)-(10). The ling carries H_4 to H_7 (7-10) belong to the MSB half of the adder. According to sum equation (1), these carries will determine the sum bits of the MSB adder. When the 8-bit adder is reconfigured as two 4-bit adders, we need to modify generate and propagate signals such that the signals from the LSB half do not affect the sum bits of the LSB half. To isolate the sum bits of the MSB 4-bit adder from the sum bits of the LSB 4-bit adder, group generate signal G_4^* and group propagate signals P_3^* and P_4^* are modified. Let α be the partition control signal.

When α=1, the Ling prefix structure should behave as an 8-bit adder and when α=0, it should reconfigure into two 4-bit adders. Let us define two new signals, αg_3 and αp_3, such that

$$\alpha\, g_3 = \alpha \,.\, g_3$$
$$\alpha\, p_3 = \alpha \,.\, p_3$$

where g_3 and p_3 are previous generate and propagate signals. Using them, we redefine the group generate signal G_4^* and group propagate signals P_3^* and P_4^* as:

$$mG_4^* = g_4 + \alpha\, g_3$$
$$mP_3^* = \alpha\, p_3 \,.\, p_2$$
$$mP_4^* = p_4 \,.\, \alpha\, p_3$$

where m indicates a modified signal. Hence, the Ling carry equations (7-10) will transform to:

$$H_4 = mG_4^* + mP_3^* \,.\, G_2^* + mP_3^* \,.\, P_1^* \,.\, G_0^*$$
$$H_5 = G_5^* + mP_4^* \,.\, G_3^* + mP_4^* \,.\, P_2^* \,.\, G_1^*$$
$$H_6 = G_6^* + P_5^* \,.\, mG_4^* + P_5^* \,.\, mP_3^* \,.\, G_2^* + P_5^* \,.\, mP_3^* \,.\, P_1^* \,.\, G_0^*$$
$$H_7 = G_7^* + P_6^* \,.\, G_5^* + P_6^* \,.\, mP_4^* \,.\, G_3^* + P_6^* \,.\, mP_4^* \,.\, P_2^* \,.\, G_1^*$$

When α=1, the above equations become the same as (7)-(10). When α=0, the above equations become:

$$H_4 = mG_4^* = g_4$$
$$H_5 = G_5^* = g_5 + g_4$$
$$H_6 = G_6^* + P_5^* \,.\, mG_4 = g_6 + g_5 + p_5 \,.\, g_4$$
$$H_7 = G_7^* + P_6^* \,.\, G_5^* = g_7 + g_6 + p_6 \,.\, g_5 + p_6 \,.\, p_5 \,.\, g_4 \quad (10)$$

These equations resemble (3)-(6) and actually make the MSB 4-bit adder work as an independent block. To see how the sum bits will be calculated, we observe that bits S_0-S_3 are calculated normally. For bit S_4, we need d_4, αp_3 and H_3 to be given to the multiplexer. From bits S_5-S_7, we need to use the modified Ling carries H_4 to H_6, and for carry-out, the modified H_7 needs to be used.

The gate level structure of the reconfigurable Ladner-Fischer Ling prefix is illustrated in figure 3, where the two AND gates shown in grey are needed to produce the modified group generate and propagate signals.

This idea of reconfigurable Ling adders can be extended to a 16-bit adder. The 16-bit parallel prefix Ling adder is shown in figure 4, also implemented using the Ladner-Fischer tree. As we can see, we need to modify two group generate-

propagate blocks (shown in grey) in a similar manner as done for an 8-bit adder. More specifically,

$mG_8^* = g_8 + \alpha\, g_7$
$mP_7^* = \alpha\, p_7 . p_6$
$mP_8^* = p_8 . \alpha\, p_7$

where α is now taken as the partition control bit for reconfiguring a 16-bit adder into two 8-bit adders. This architecture can be further partitioned to work as one 16-bit adder, two 8-bit adders and four 4-bit adders. This reconfigurability is achieved using two partition control bits: α and β. Table 1 states the three cases and how the adder behaves when those cases are realized.

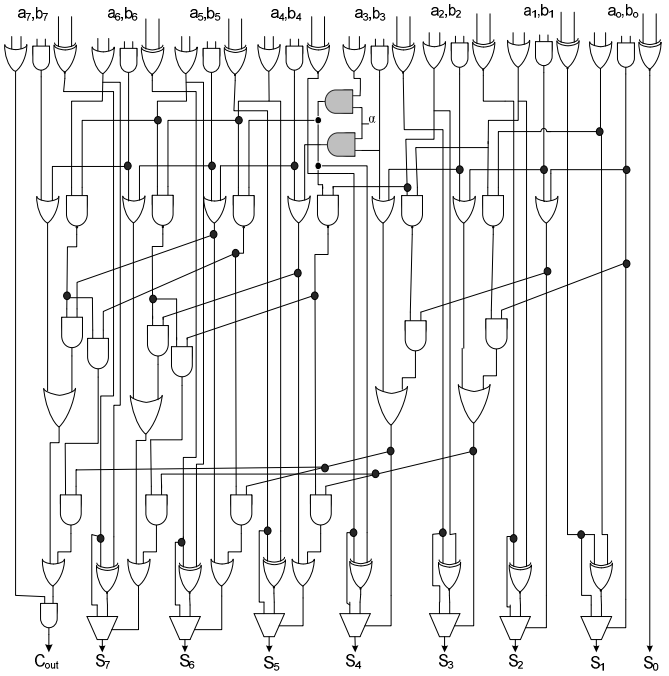

Figure 3: Reconfigurable parallel prefix Ling adder (Ladner-Fischer)

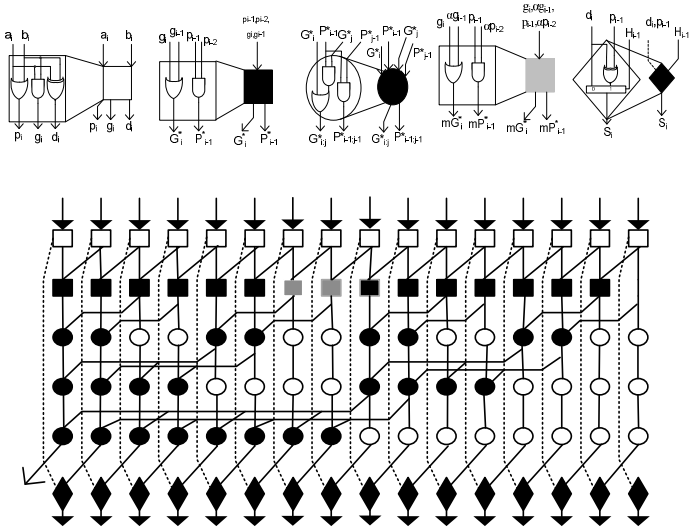

Figure 4: 16-bit reconfigurable parallel prefix Ling adder (Ladner-Fischer)

TABLE 1 : ADDER PARTITION FOR CONTROL SIGNAL VALUES

α	**β**	**Adder**
1	1	16-bit adder
1	0	Two 8-bit adders
0	0	Four 4-bit adders

The signals to be modified will be:
$mG_4^* = g_4 + \alpha g_3$, $mP_3^* = \alpha p_3 . p_2$, $mP_4^* = p_4 . \alpha p_3$;
$mG_8^* = g_8 + \beta g_7$, $mP_7^* = \beta p_7 . p_6$, $mP_8^* = p_8 . \beta p_7$;
$mG_{12}^* = g_{12} + \alpha g_{11}$, $mP_{11}^* = \alpha p_{11} . p_{10}$, $mP_{12}^* = p_{12} . \alpha p_{11}$

Now that we have shown how to achieve reconfigurability in a parallel prefix Ling adder implemented using the Ladner-Fischer tree, the same algorithm can be proposed for a Ling adder implemented using the Kogge-Stone tree. The figure 5 shows a 16-bit Kogge-Stone Ling adder with the modified blocks that will modify the incoming generate and propagate signals in a similar manner as has been shown for a Ladner-Fischer structure. As the Kogge-Stone Adder is the fastest tree prefix adder with the advantage of having bounded fan-out (maximum 2) [3], we have used the Kogge-Stone Algorithm in all the hardware implementations that we have performed in this paper.

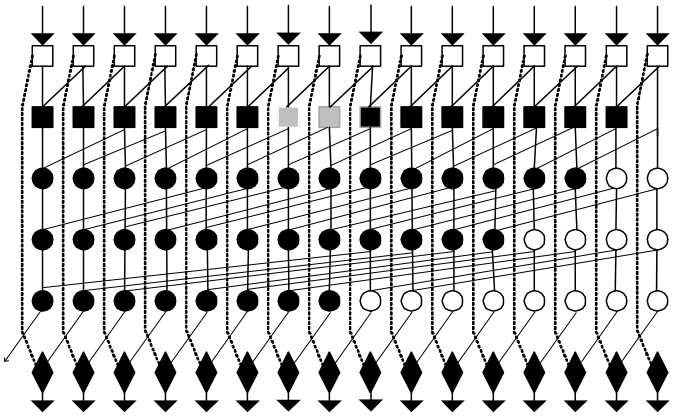

Figure 5: 16-bit reconfigurable parallel prefix Ling adder (Kogge-Stone)

B. Modified EFBA

There have been research works that integrate parallel prefix Ling adder with flagged addition, but they do not achieve flagged constant addition. This can be done by taking the sum bits S_i from the output of a Ling adder and giving them as inputs to the flagged constant addition block that was proposed in [6]. The resulting parallel prefix Ling adder with flagged constant addition will be faster than the traditional parallel prefix carry lookahead adder with the same flagged constant addition block.

The enhanced flagged binary adder (EFBA) architecture presented in [6] is accomplished via a modified ripple carry adder (RCA) to add the sum and a known constant as shown in figure 6.

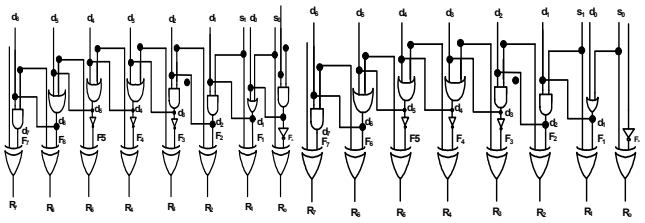

Figure 6: 16-bit enhanced flagged binary adder (EFBA)

However, the existing design proposed in [6] computes the flag bits, and correspondingly, the sum bits in a serial fashion (figure 6). In case the constant to be added is known, the implementation of the EFBA logic can be done in parallel by means of a prefix tree as shown in figure 8. In our proposed design, we have used a modified Ling prefix tree scheme to reduce the delay (which occurs in the existing scheme [6] due to the rippling effect) in calculation of flag bits. The known constant to be added to the Sum in our case is 00111001 00111001 which is chosen because the 8 MSB bits and the 8 LSB bits are same as 00111001 which was the constant used in [6]. Therefore, the two designs (figure 6 and 8) can be compared on similar grounds. Note that figure 7 shows the block diagram of our proposed design with all the 16 bits shown together. The circuit of each of these blocks is depicted in figure 8.

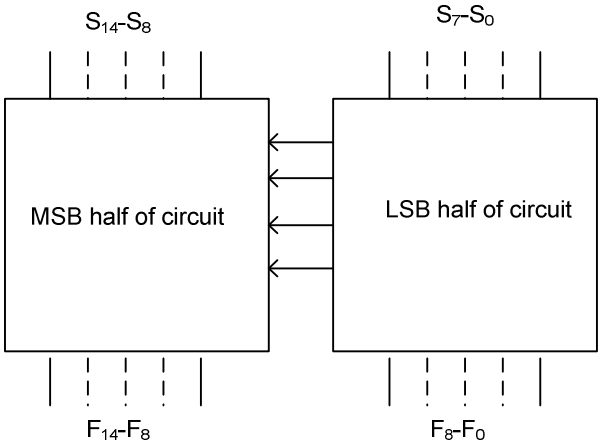

Figure 7: Block diagram view of the proposed flagged binary adder.

C. Comparison of proposed and existing designs :

Prefix Tree Block :

Delay: The G*P* block in Ling adder uses one logic level less compared to the first level of CLA. As can be seen clearly from the figure, the black dot in CLA represents a complex and-or gate compared to the single gate level (OR gate) in Ling adders [5] represented by black square.

Area: Kogge stone prefix tree is used to implement a 16-bit CLA and a 16-bit Ling adder (figure 5).

The first stage of the ling adder (black square) takes less hardware than the first stage of the CLA (black dot). However, ling adder requires an additional XOR gate per bit (assuming that a XOR gate and a 2:1 multiplexer have similar complexity) for calculation of the sum bits using the Ling carries H_i. Further, the g-p-d block (white squares) as shown in figure 5 of the Ling adder (where g_i, d_i and p_i are calculated) carries one gate extra compared to the g-p block of a CLA (where only g_i and p_i needs to be calculated). Therefore, the hardware complexity and correspondingly the area increases in case of a Ling adder.

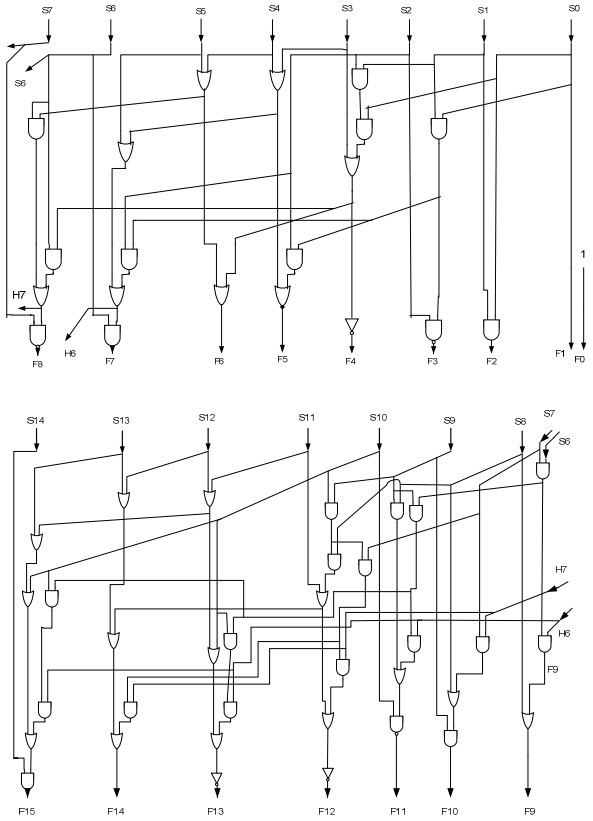

Figure 8: 16 bit Modified Enhanced Flagged Binary Adder(MEFBA)

Flag Block:

Delay: Our proposed MEFBA design gives a significant reduction in delay as compared to the conventional EFBA. This reduction in delay is achieved due to parallel operation in MEFBA as compared to serial operation (ripple carry effect) in EFBA.

Area: There is a slight increase in area for the proposed MEFBA design over the existing EFBA design. This is due to the fact that parallel operations typically require more hardware than serial operations.

Proposed Design:

Delay: Our proposed reconfigurable Ling Adder (RLing)– modified enhanced flagged binary adder (MEFBA) arrangement (figure 8) is compared with the reconfigurable CLA Adder (RCLA) - EFBA arrangement (figure 6). Our scheme reduces the delay significantly when compared to RCLA - EFBA arrangement.

Area: As the proposed scheme was implemented using parallel prefix ling scheme, the total area required in this design is slightly more than the existing design.

IV. RESULTS

The existing and proposed adder designs discussed in this paper were structurally described using verilog HDL and simulated in R=0 mode using Cadence Incisive Unified Simulator (IUS) v6.1. The adder structures were mapped on TSMC 180nm Technology Slow-Normal library (operating conditions 0.9V, 125ºC) using Cadence RTL Compiler v7.1. All the inputs were set to have a toggle rate of 50%.

TABLE 2 (A): Comparison of existing EFBA with the proposed MEFBA

Metric	RCLA +EFBA	RCLA +MEFBA
Area(μm^2)	713.362 (100%)	718.301 (+0.692%)
Delay(ps)	1893 (100%)	1771 (- 6.445%)
Power(nW)	15627.767 (100%)	15802.489 (+1.12%)
Power Delay Product	29583362.93 (100%)	27986208.02 (- 5.398%)

TABLE 2(B): Comparison of Carry Lookahead Adder + EFBA with Ling Adder + EFBA

Metric	RCLA +EFBA	RLing +EFBA
Area(μm^2)	713.362 (100%)	800.856 (+12.26%)
Delay(ps)	1893 (100%)	1887 (-0 .317%)
Power(nW)	15627.767 (100%)	15410.588 (-1.39%)
Power Delay Product	29583362.93 (100%)	29079779.56 (-1.7%)

TABLE 2(C): Comparison of existing scheme with our proposed design

Metric	RCLA +EFBA	RLing +MEFBA
Area(μm^2)	713.362 (100%)	805.795 (+12.957%)
Delay(ps)	1893 (100%)	1764 (- 6.814%)
Power(nW)	15627.767 (100%)	15575.317 (-0.33%)
Power Delay Product	29583362.93 (100%)	27474859.19 (-7.12%)

As seen from the Table 2(A) to 2(C), with slight increase in area there is upto 6.8% reduction in delay. Further this is pessimistic estimation. With increase in the number of bits (size of the operands), the "reduction in delay" percentage will further increase.

V. CONCLUSION

This paper demonstrated how to make a parallel prefix Ling adder reconfigurable by introducing a partition control bit in the structure. The proposed architecture for a 16-bit Ling adder can be reconfigured into two 8-bit adders and four 4-bit adders. We also studied a modified logic for Enhanced Flagged Binary Adder and compared it with the existing EFBA. The proposed design, when implemented in a carry lookahead adder, is 6.5% faster along the critical path with a reduced power-delay product. The delay advantage increases to 6.8% along the critical path when the proposed design is implemented in a parallel prefix Ling adder.

VI. REFERENCES

[1] I. Koren, Computer Arithmetic Algorithms. A.K. Peters, Ltd., 2002.

[2] B. Parhami, Computer Arithmetic—Algorithms and Hardware Designs. Oxford Univ. Press, 2000.

[3] R. Zimmermann. *Binary Adder Architectures for Cell-Based VLSI and their synthesis.* PhD Thesis. Swiss Federal Institute of Technology, Zurich, 1997.

[4] H. Ling, "High-Speed Binary Adder," IBM J. R&D, vol. 25, pp. 156-166, May 1981.

[5] G. Dimitrakopoulos and D. Nikolos, "High-Speed Parallel-Prefix VLSI Ling Adders", *IEEE Trans. on Computers*, vol. 54, no. 2, pp. 225–231, Feb. 2005.

[6] V.Dave, E.Orukulu, J.Sainiie, "Constant addition with flagged binary adder architectures", Journal Integration, the VLSI Journal, vol 43, no 3, pp 258-267, June 2010.

[7] Jin-Fu Li, Jiunn-Der Yu, and Yu-Jen Huang, "A Design Methodology for Hybrid Carry- lookahead/Carry-Select Adders with Reconfigurability", in *IEEE International Symposium on Circuits and Systems*, Vol. 1, pp. 77 – 80, May-2005.

[8] N. Burgess, The flagged prefix adder and its applications in Integer arithmetic, Journal of VLSI Signal Processing Systems vol. 31 (2002) 263–271.

[9] P.M. Kogge and H.S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," IEEE Trans. Computers, vol. 22, no. 8, pp. 786-792, Aug. 1973.

[10] R.E. Ladner and M.J. Fisher, "Parallel Prefix Computation," J. ACM, vol. 27, no. 4, pp. 831-838, Oct. 1980.