Implementation of Fast Binary Counters Based on Symmetric Stacking

[1]I.Suneetha, [2] P. Neethachowdary, [3] K.Chethan, [4]V.Chandrika, [5] C.DileepKumar ,[6] P.Gunasekhar

[1] Professor & Head of the Department ECE, Annamacharya Institute of technology and sciences ,Tirupathi

[23456] Students: Department of ECE, Annamacharya Institute of technology and sciences ,Tirupathi

[23456]Email:-neethapantra88@gmail.com,kondampatichethan@gmail.com,vchandrika418@gmail.com,

dileeprandy2222@gmail.com, gunaroyal1429@gmail.com

Abstract-

In this short, another twofold counter structure is proposed. Wallace tree multipliers give a power-proficient methodology to rapid duplication. The utilization of rapid 7:3 counters in the Wallace tree decrease can additionally enhance the multiplier speed. And furthermore we actualize 128bit Vedic Wallace multiplier give fast and expends not so much power but rather more productively. Consequently in proposed strategy we create 8X8 Wallace tree multiplier stacker and Vedic Wallace 128X128 piece stacker. These proposed strategies have preferred execution enhancement over 6 TO 3 Bit stacker and 7 TO 3 bit stacker. In existing technique, It utilizes 3-bit stacking circuits, which bunch the majority of the "1" bits together, trailed by a novel symmetric strategy to consolidate sets of 3-bit stacks into 6-bit stacks. The bit stacks are then changed over to paired checks, delivering 6:3 counter circuits with no xor doors on the basic way. This shirking of xor entryways results in quicker plans with effective power and zone usage. In VLSI reenactments, the proposed counters are 30% quicker than existing parallel counters and furthermore expend less power than other higher request counters. Furthermore, utilizing the proposed counter-based Wallace tree multiplier models diminishes inertness and power utilization for 128-piece multipliers.

Index terms **– Counter, high speed, low power, multiplier, VLSI, Wallace tree.**

I. INTRODUCTION

Fast, effective expansion of different operands is a fundamental activity in any computational unit. The speed and power effectiveness of multiplier circuits is of basic significance in the general execution of chip. Multiplier circuits are a fundamental part of a math rationale unit, or an advanced flag processor framework for performing sifting and convolution. The parallel augmentation of whole numbers or settled point numbers results in halfway items that must be added to create the last item. The expansion of these fractional items overwhelms the dormancy and power utilization of the multiplier. So as to join the fractional items productively, section pressure is normally utilized. Numerous strategies have been introduced to upgrade the execution of the incomplete item summation, for example, the notable column pressure procedures in the Wallace tree [1] or Dadda tree [2], or the enhanced design in [3]. These strategies include utilizing full adders working as counters to lessen gatherings of 3 bits of a similar load to 2 bits of various load in parallel utilizing a convey spare viper tree. Through a few layers of decrease, the quantity of summands is diminished to two, which are then included utilizing a customary viper circuit. To accomplish higher proficiency, bigger quantities of bits of equivalent weight can be considered. The fundamental strategy when managing bigger quantities of bits is the equivalent: bits in a single segment are tallied, creating less bits of various loads. For instance, a 7:3 counter circuit acknowledges 7 bits of equivalent weight and checks the quantity of "1" bits. This tally is then yield utilizing 3 bits of expanding weight.

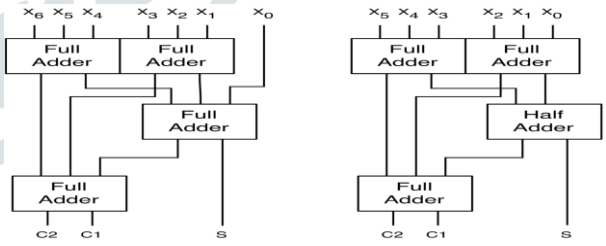


Fig. 1. A 7:3 counter and a 6:3 counter built from full and half adders.

The 7:3 and 6:3 counter circuits can be built utilizing full and half adders, as appeared in Fig. 1. A significant part of the deferral in these counter circuits is because of the chains of XOR entryways on the basic way. In this way, a lot quicker parallel counter design has been exhibited. A parallel 7:3 counter was exhibited in [4] and used to plan a fast counter-based Wallace tree multiplier in [5]. Also, counter structures as in [6] and [7] use multiplexers to lessen the quantity of XOR doors on the basic way. A portion of these muxes can be actualized with transmission door rationale to deliver considerably quicker plans.

In this concise, we present an including technique that utilizes bit stacking circuits pursued by a novel strategy for joining two little stacks to frame bigger stacks. A 6:3 counter constructed utilizing this technique utilizes no XOR doors or multiplexers on its basic way. VLSI reenactment results demonstrate that our 6:3 counter is in any event 30% quicker than existing counter structures while additionally utilizing less power. Recreations were likewise kept running on full multiplier circuits for different sizes. A similar counter-based Wallace multiplier configuration was utilized for every recreation, while the inward counter was fluctuated. Utilization of the proposed counter enhances multiplier effectiveness for bigger circuits, yielding 64-and 128-piece multipliers that are both quicker and expend less power than other counter based Wallace (CBW) plans.This paper is organized in six sections. After this introduction, in Section II, Symmetric bit stacking discussed, Section III existing method discussed of the paper. Finally, Sections IV about the proposed method explained, as well as the novel feature of the proposed method and V provides the simulation results and the conclusions, respectively.

II. SYMMETRIC BIT STACKING

The proposed 6:3 counter is acknowledged by first stacking the majority of the information bits with the end goal that the majority of the "1" bits are gathered together. Subsequent to stacking the info bits, this stack can be changed over into a twofold check to yield the 6-bit tally. Little 3-bit stacking circuits are first used to frame 3-bit stacks. These 3-bit stacks are then joined to make a 6-bit stack utilizing a symmetric procedure that includes one additional layer of rationale.

A. Three-Bit Stacking Circuit

Given data sources X0, X1, and X2, a 3-bit stacker circuit will have three yields Y0, Y1, and Y2 with the end goal that the quantity of "1" bits in the yields is equivalent to the quantity of "1" bits in the information sources, yet the "1" bits are gathered together to one side pursued by the "0" bits. It is

Clear that the yields are then framed by

$Y0 = X0 + X1 + X2 \ldots \ldots . (1)$

$Y1 = X0X1 + X0X2 + X1X2 \ldots . (2)$

$Y2 = X0X1X2 \ldots \ldots .. (3)$

In particular, the primary yield will be "1" if any of the sources of info is one, the second yield will be "1" if any two of the information sources are one, and the last yield will be one if each of the three of the data sources are "1." The Y1 yield is a dominant part work and can be actualized utilizing one complex CMOS door. The 3-bit stacking circuit is appeared in Fig. 2.

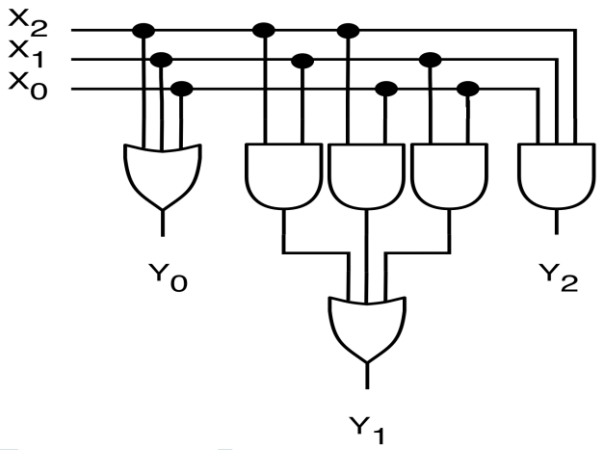


Fig. 2. Three-bit stacker circuit.

B. Merging Stacks

We wish to frame a 6-bit stacking circuit utilizing the 3-bit stacking circuits talked about. Given six information sources X0, . . . , X5, we first partition them into two gatherings of 3 bits which are stacked utilizing 3-bit stacking circuits. Let X0, X1, and X2 be stacked into signs named H0, H1, and H2 and X3, X4, and X5 be stacked into I0, I, and I2. In the first place, we turn around the yields of the primary stacker and consider the six bits H2, H1, H0, I0, I1, and I2. See the highest point of Fig. 3 for a case of this procedure. We see that inside these six bits, there is a train of "1" bits encompassed by "0" bits. To frame an appropriate stack, this train of "1" bits must begin from the furthest left piece.

In order to form the proper 6-bit stack, two more 3-bit vectors of bits are formed called *J*0, *J*1, *J*2 and *K*0, *K*1, *K*2. The idea is to fill the *J* vector with ones first, before filling the *K* vector. So we let

$$J_0 = H_2 + I_0 \ldots\ldots\ldots (4)$$

$$J_1 = H_1 + I_1 \ldots\ldots\ldots (5)$$

$$J_2 = H_0 + I_2 \ldots\ldots\ldots (6)$$

In this way, the first three “1” bits of the train are guaranteed to fill into the *J* bits although they may not be properly stacked. Now to ensure no bits are counted twice, the *K* bits are formed using the same inputs but with the AND gates instead

$$K_0 = H_2 I_0 \ldots\ldots\ldots (7)$$

$$K_1 = H_1 I_1 \ldots\ldots\ldots (8)$$

$$K_2 = H_0 I_2 \ldots\ldots\ldots (9)$$

On the off chance that the train of "1"s is close to three places in length, the majority of the K bits will be "0" as the AND entryway inputs are three positions separated.

In the event that the train is longer than three places in length, a portion of the AND doors will have the two contributions as "1"s as the AND entryway inputs are three positions separated. The quantity of AND doors that will have this property will be three not exactly the length of the train of "1"s. We see that now J0 J1 J2 K0K1K2 still contain a similar number of "1" bits as the contribution to add up to however at this point J bits will be loaded up with ones preceding any of the K bits. We should now stack J0 J1 J2 and K0K1K2 utilizing two more 3-bit stacking circuits. The yields of these two circuits would then be able to be connected to frame the stack yields Y5. Y0.

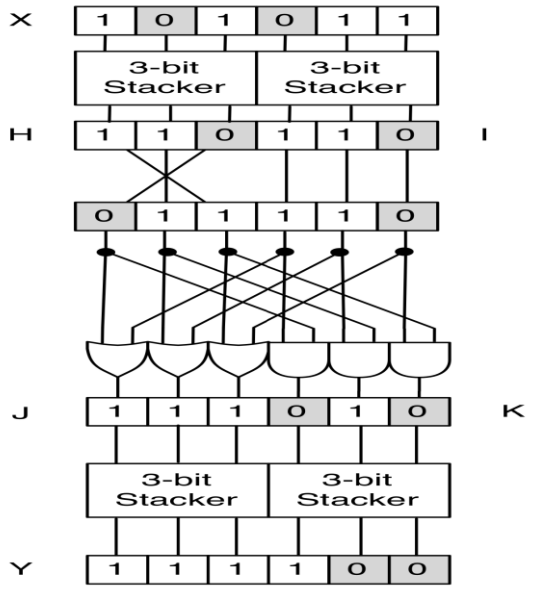


Fig. 3. Six-bit stacking example.

An example of this process is shown for an input vector containing four "1" bits in Fig. 3. In this example, first the *H* and *I* vectors are formed by stacking groups of three input bits. Then, the *H* vector is reversed, forming a continuous train of four "1" bits surrounded by zero bits. Corresponding bits are OR-ed to form the *J* vector which is full of "1" bits. Corresponding bits are AND-ed to form the *K* vector which finds exactly one overlap. Then, the *J* and *K* vectors are restacked to form the final 6-bit stack.

III.IMPLEMENTATION SYSTEM

A.6:3 Counter based design

For a faster, more efficient count, we can use intermediate values *H*, *I* and *K* to quickly compute each output bit without needing the bottom layer of stackers. Call the output bits *C*2, *C*1, and *S* in which *C*2, *C*1, *S* is the binary representation of the number of "1" input bits.

To compute *S*, we note that we can easily determine the parity of the outputs from the first layer of 3-bit stackers. Even parity occurs in the *H* if zero or two "1" bits appear in *X*0, *X*1, and *X*2.

Thus, *He* and *Ie*, which indicate even parity in the *H* and *I* bits, are given by

$$He = H0 + H1H2 \quad (10)$$

$$Ie = I0 + I1\ I2. \quad (11)$$

As *S* indicates odd parity over all of the input bits, and because the sum of two numbers with different parities is odd, we can compute *B*0 as

$$S = He \oplus Ie. \quad (12)$$

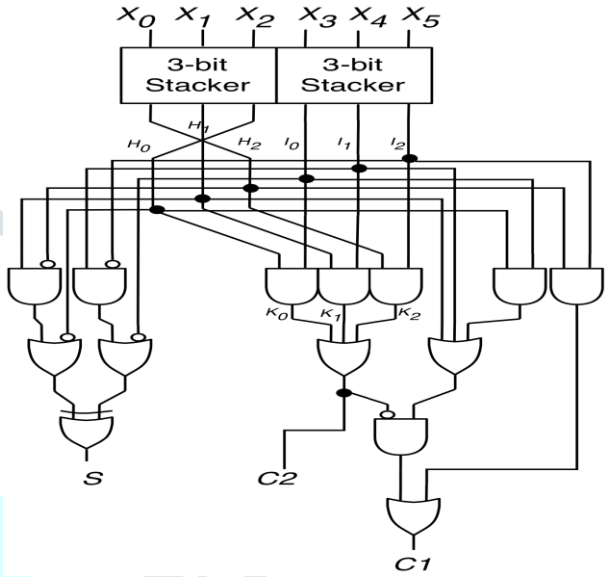


Fig. 4. A 6:3 counter based on symmetric stacking

Although this does incur one XOR gate delay, it is not on the critical path. To compute C1, we note C1 = 1 when the count is 2, 3, or 6. Therefore, there are two cases. First, we need to check if we have at least two but no more than three total inputs. We can use the intermediate H, I, and K vectors for this. To check for at least two inputs we need to see stacks of length two from either top level stacker, or two stacks of length one, which yields H1 + I1 + H0 I0. Second, we need to check if we have all six inputs as "1." We can check this by checking that all three of both the *H* and *I* bits are set. As these are bit stacks, we simply check the rightmost bit in the stack for this case, which yields H2 I2. Altogether, this yield

$$C1 = (H1 + I1 + H0\ I0)\ (K0 + \overline{K1} + K2) + H2\ I2. \quad (13)$$

We can easily calculate C2 as it should be set whenever we have at least 4-bit set

$$C2 = K0 + K1 + K2.$$

B.7:3 Counter based design

The symmetric stacking method can be used to create a 7:3 counter as well. The 7:3 counters are desirable as they provide a higher compression ratio. The design of the 7:3 counter involves computing outputs for *C*1 and *C*2 assuming both *X*6 = 0 (which matches the 6:3 counter) and assuming *X*6 = 1. We

compute the S output by adding one additional XOR gate.

If $X6 = 1$, then $C1 = 1$ if the count of $X0, \ldots, X5$ is at least 1 but less than 3 or 5, which can be computed as $C1 = (H0 + I0)J0\ J1\ J2 + H2\ I1 + H1\ I2$. (15) Also, $C2 = 1$ if the count of $X0, \ldots, X5$ is at least 3 $C2 = J0\ J1\ J2$. Both versions of $C1$ and $C2$ are computed and a mux is used to select the correct version based on $X6$. has muxes on the critical path. The 7:3 counter designs is shown in Fig. 5.

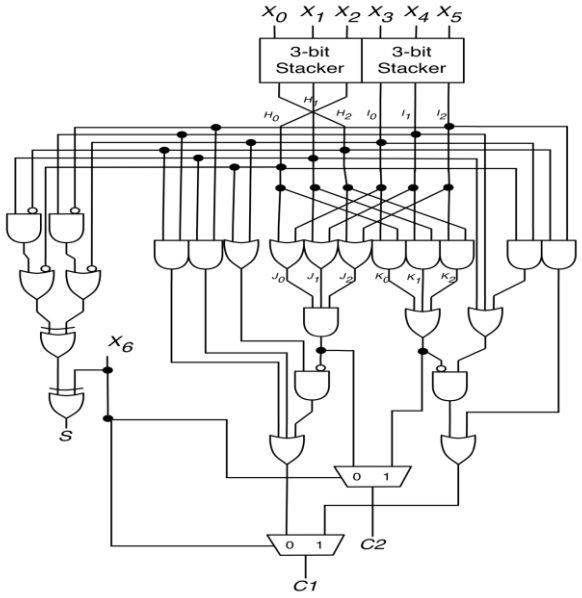


Fig. 5. A 7:3 counter based on symmetric stacking

A. Wallace tree multiplier

This section discusses the design of counter based Wallace (CBW) multiplier. The proposed algorithm uses a readjusted form of partial product tree which is rearranged as reverse pyramid [4]. Then the reduction is performed using the 7:3, 6:3, 5:3, and 4:3 counters along with the FAs and HAs. The CBW can perform the tree reduction in fewer stages as compared to the traditional Wallace multipliers [1]. Now, we will develop the equations to compute the maximum rows in each stage of CBW multiplier and total stages required for reduction process for N×N multiplier.

The first stage of an $N \times N$ multiplier has N rows. We need to find the maximum rows in subsequent stages until we are left with only two rows. Let us assume that the maximum rows in stage$i-1$ are 16 and the number of rows in each column are same. In order to perform the reduction at column c, two 7:3 compressors are used which will operate on 14 rows. The remaining two rows are reduced by using a 2:2 compressor.

This will reduce the rows in column c from sixteen to three. Similarly, the columns $c - 1$ and $c - 2$ are reduced by using two 7:3 and one 2:2 compressor. The three compressors used at column $c-1$ will produce three $Cout1$ bits which are added to the column c of the stage$i$. This will increase the rows in column c of stage$i$ from 3 to 6. The two 7:3 compressors at column $c - 2$ will produce two $Cout2$ bits which are also added to the column c of stage$i$. Hence, the rows in column c of stage$i$ will increase from 6 to 8.

It can be seen from the above example that the 2:2 compressor at column $c - 2$ does not produce a $Cout2$ bit so it has no effect on column c. The compression is performed mainly by using 7:3 compressors, the other compressors are used only if the rows in a column are not exact multiple of seven. There will be one unprocessed row if the rows in column c are equal to $(n \times 7) + 1$, where n is any positive integer.

Based on the observations of above example, we can obtain the rows in stage$i$ by adding

1) The number of traditional and proposed compressors at column c and $c - 1$ of stage$i - 1$.

2) The number of proposed compressors (7:3, 6:3, 5:3, and 4:3) at column $c - 2$ of stage$i - 1$.

3) The unprocessed row at column c of stage$i - 1$.

The dot notation [11] is used to represent the partial product tree of the CBW multiplier discussed in this section as shown in Fig. 5. The right most column is called column 0. The counters in each column are represented by the boxes around the dot products. The box enclosing seven, six, five, four, three, and two dots represents 7:3, 6:3, 5:3, 4:3, 3:2, and 2:2 counters, respectively. The stages are separated by a thick horizontal line. The architecture of CBW multiplier is based on the intelligent use of high speed counters. The Fig. 6 shows the dot diagram of a 16×16 CBW multiplier.

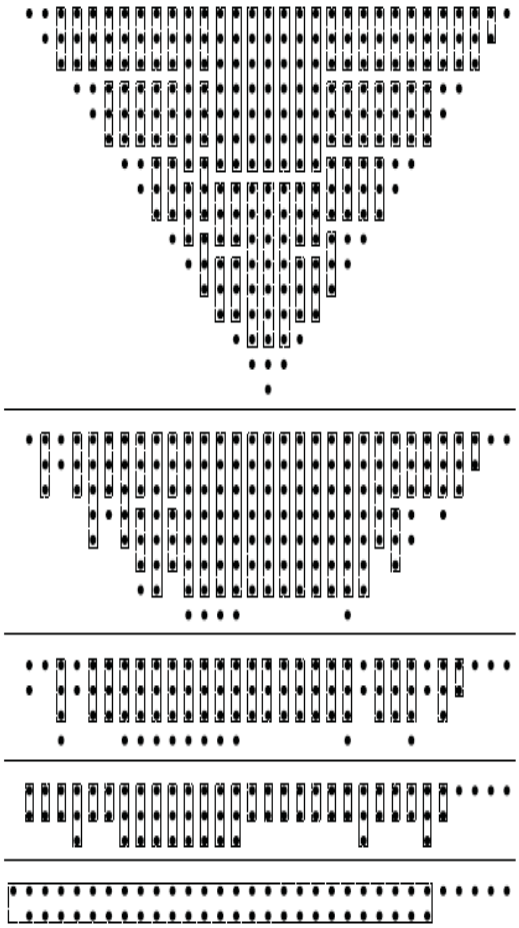


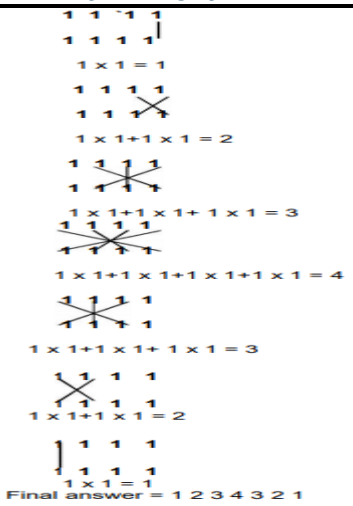
Fig. 6: Dot Diagram of CBW Multiplier

B. Vedic Wallace tree multiplier

Multiplication methods are extensively discussed in Vedic mathematics. Various tricks and short cuts are suggested by VM to optimize the process. These methods are based on concept of 1 Multiplication using deficits and excess 2 Changing the base to simplify the operation. Various methods of multiplication proposed in VM a) UrdhvaTiryagBhyam - vertically and crosswise b) Nikhilam navatashcharamam Dashatah: All from nine and last from ten

Urdhva – Triyakbhyam is the general formula applicable to all cases of multiplication and also in the division of a large number by another large number. It means vertically and crosswise. We discuss multiplication of two, 4 digit numbers with this method [8-9]. Ex.1. the product of 1111 and 1111 using Triyakbhyam (vertically and crosswise) is given below

Methodology of Parallel Calculation



VI. CONCLUSION

In this paper, we proposed a generic algorithm that can be used to construct the Counter Based Wallace (CBW) multiplier of any size. Here we have developed 8X8 Wallace tree multiplier stacker and also a highly efficient method of multiplication – "Urdhva Tiryakbhyam Sutra" based on Vedic mathematics. It is a method for hierarchical multiplier design which clearly indicates the computational advantages offered by Vedic methods.

Wallace tree multipliers provide a power-efficient strategy for high speed multiplication. The use of high speed 7:3 counters in the Wallace tree reduction can further improve the multiplier speed. And also we implement 128bit Vedic Wallace multiplier provide high speed and consumes less power and more efficiently. Hence in proposed method we have developed 8X8 Wallace tree multiplier stacker and Vedic Wallace 128X128 bit stacker.

REFERENCESS

[1] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Electronic Computers*, vol. EC-13, no. 1, pp. 14–17, February 1964.

[2] J. Fadavi-Ardekani, "M*n booth encoded multiplier generator using optimized wallace trees," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, no. 2, pp. 120–125, 1993.

[3] H. Eriksson, P. Larsson-Edefors, M. Sheeran, M. Sjalander, D. Johansson, and M. Scholin, "Multiplier reduction tree with logarithmic logic depth and regular connectivity," in *Proc.IEEE Int. Symp. Circuits and Systems, 2006. ISCAS*, Island of Kos, May 2006, pp. 5–8.

[4] R. S. Waters and E. E. Swartzlander, "A reduced complexity Wallace multiplier reduction," *IEEE Transactions on Computers*, vol. 59, no. 8, pp. 1134–1137, August 2010.