Design and Synthesis of a Three Input Flagged Prefix Adder

Vibhuti Dave, Erdal Oruklu, and Jafar Saniie
Department of Electrical and Computer Engineering
Illinois Institute of Technology
Chicago, Illinois, USA
{vdave,eoruklu,sansonic}@ece.iit.edu

Abstract— **For multi-operand addition, several techniques, such as carry-save adders, Wallace, and Dadda structures based on counters and compressors have been proposed. This paper proposes a technique to accomplish three-operand addition utilizing regular adder structures such as parallel-prefix adders. One of the advantages of this technique is the elimination of dedicated adder units to perform three-input addition. Conventional prefix adders are modified to generate intermediate outputs called flag bits to allow the addition of a third arbitrary number, thereby accomplishing multi-operand addition. This adder can find its use in applications such as multiplication or multi-media units. An evaluation has been performed for 16-bit three-input flagged prefix adder architectures (TIFPA) in terms of area, delay and power. The performance of this adder design has been compared to that of carry save adders to understand the performance gain of the proposed technique.**

I. INTRODUCTION

Minimizing the number of resources required within a processor would have a positive impact on its power performance. Furthermore, since an adder is one of the basic arithmetic units, any improvement in the performance of an adder would have a significant impact on the performance of a processor. This paper proposes a technique to accomplish these objectives by introducing certain flexibility to parallel prefix designs [1] [2] [3]. This flexibility allows prefix adders to be able to add three numbers at a time, thereby accomplishing multi-operand addition. This will eliminate the need to have a dedicated adder unit such as carry-save adders [4] to perform the same task along with a performance boost over carry-save adders (CSA).

Parallel prefix architectures are popular adder designs due to their regular layout and the fact that the carry signals are generated in parallel increasing the speed of the circuit. Prefix computation takes place in three stages, the first stage being the pre-processing stage which generates the *bit propagate* and the *bit generate* signals. This is followed by the prefix tree that generates all the *carry* signals in parallel and finally the post-processing stage that produces the final result which is a sum of the two input operands [5].

The prefix adders were first modified to produce a new design called flagged prefix adder in [6]. The flagged prefix adders utilize the *bit propagate* and *group propagate* outputs from the prefix tree to generate a new set of intermediate outputs called the *flag* bits [6]. The flag bits are further used to select the appropriate sum bits from the post-processing stage that need to be inverted to generate a completely new set of results. The new result could be the sum of the two input operands augmented or decremented by unity. The concept of generating flag bits was utilized to introduce the flexibility of constant addition [7] utilizing the *carry* bits from the prefix tree. This would enable the sum of two operands, A and B to be augmented/decremented by a constant M.

This paper proposes to incorporate additional hardware within the prefix adder to further enhance the performance of the design allowing the third operand, M to be any arbitrary number instead of limiting the utilization of the technique for constant addition. The cost of additional hardware is compensated for by the performance gain in terms of speed and the elimination of utilizing a dedicated arithmetic unit to perform three-input addition. This paper investigates the performance of three input flagged prefix adders (TIFPA) for 16-bit operand sizes modified to incorporate flag logic necessary to compute the sum of three numbers. The performance of the new architecture is also compared to the performance of 16-bit carry save designs.

The concept of flagged prefix addition is reviewed in Section II. Section III describes the implementation of the new hardware within the prefix adder design to add three numbers. Section IV provides the area, speed and power results for 16-bit designs along with a performance comparison to a 16-bit carry save design. Section V presents the conclusions.

II. BACKGROUND

Consider a parallel prefix adder, with two input operands, A and B. The result of adding these two numbers is represented by R. The flagged prefix adder is a modified prefix architecture that generates a new set of intermediate outputs called flag bits. The flag bits flag the bits of the

1-4244-0921-7/07 $25.00 © 2007 IEEE.

result, R that need to be inverted to generate a new result. This set of results includes simple increment decrement operations where the result, R is augmented or decremented by unity. The new result is a simple XOR operation between the flag bits, F and the result bits, R. The flag bits can be computed from the *group propagate* signals according to the following relationship, [6]

$$f_k = P_{k-1:0} \tag{1}$$

f_i represents the flag bit for the k^{th} bit position (f_{-1}=0) and $P_{k-1:0}$ represent group propagate signals from significance 0 to $k-1$. The post-processing stage of the prefix adder is replaced by a new stage comprising of the *flagged inversion cells* (FIC) to generate the new result [6].

This design therefore causes a regular prefix adder to take two operands as inputs but generate various other useful results without having to utilize a second adder to generate the same set of results. The extra hardware that needs to be incorporated to accomplish this flexibility is insignificant compared to the utilization of the dual adder scheme [6].

The concept of flag generation has been extended to compute a new result, $A+B+M$ where M can be a constant or any arbitrary number. The following section describes the generation of the flag bits and the implementation of the flag logic to produce the new result.

III. Flag Logic for M

In [7], a method to generate a new set of flag bits in order to compute $(A+B+M)$, is proposed where M is any constant. Assume that R is the result of adding two arbitrary inputs, A and B, and R needs to be augmented/decremented by a value, M. The full adder equations can be written as [8]

$$S_k = R_k \oplus M_k \oplus c_k$$

$$c_{k+1} = \begin{cases} R_k \cdot c_k & if \;\; M_k = 0 \\ R_k + c_k & if \;\; M_k = 1 \end{cases} \tag{2}$$

In Eq.2, S_k and c_k represent the new set of sum and carry bits for every bit position. Utilizing the new set of equations, the new sum needs to be computed such that, $R+M=R \oplus F$, where F is the flag function. The flag bits can be seen as bits that indicate whether the current value is flagged to change. Consequently, the flag bits can be computed based on speculative elements of the constant. Two bits of the constant are examined to determine whether or not the carry bit from the constant affects the current position. As derived in [7], the flag bits are computed according to Table I [7] [8]. Table I tabulates all the flag logic gates that might be required for the k^{th} bit position depending on the corresponding bits from the third operand and the carry bit for that position. The initial conditions are assumed as given in Eq. 3 [7].

$$R_{-1} = M_{-1} = F_{-1} = 0 \tag{3}$$

Table I Flag Logic Utilizing Carry Produced from Prefix Tree

M_k	M_{k-1}	$F_k(c_k=0)$	$F_k(c_k=1)$
0	0	$R_{k-1} \cdot F_{k-1}$	$\overline{R_{k-1}} \cdot F_{k-1}$
0	1	$R_{k-1} + \overline{F_{k-1}}$	$\overline{R_{k-1} \cdot F_{k-1}}$
1	0	$\overline{R_{k-1} \cdot F_{k-1}}$	$R_{k-1} + \overline{F_{k-1}}$
1	1	$\overline{R_{k-1}} \cdot F_{k-1}$	$R_{k-1} \cdot F_{k-1}$

As can be seen from Table I, the flag logic depends on the bits from the third input operand and the carry bits from the prefix tree. The FIC, therefore will be implemented according to Table II. A straightforward implementation of this is shown in Fig. 1. However it is important to notice that the order of the logic gates in the first stage will depend on the third input operand [9]. Therefore, this implementation can work only when the third operand is a constant and never changes [9]. Fig. 1 shows the implementation for the four different combinations depending on the two bits of the constant. In order to eliminate the limitation of having the third operand as a constant, the FIC in Fig.1 are modified to get a multilevel set of FIC.

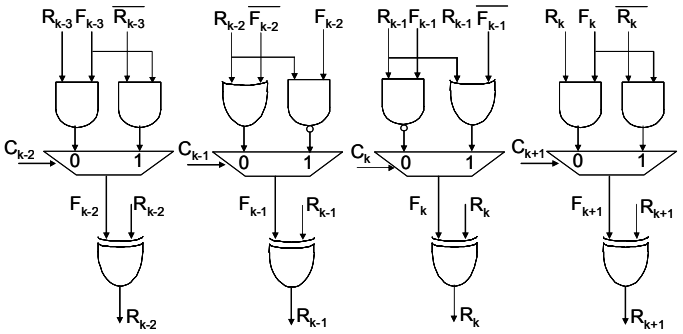

Figure 1 Flag Inversion Cells for Constant Addition

It can be deduced observing Fig. 1 and Table I, that a minimum of four logic gates are required to compute the flag bits based on eight different combinations of the constant and carry bits. The four logic gates and their inputs are summarized in Table II. Table III summarizes the combinations of the constant and carry bits that select one of the four logic gates mentioned in Table II to compute the flag bit in each successive position. The FIC will therefore be a multi-level structure as shown in Fig. 2. Notice, that the flag bit computation for each bit position depends on the flag bit computed in the preceding position, thereby causing a rippling effect, making this the critical path of the circuit.

Table II Minimum Flag Logic Gates

Gate	Inputs
AND1	R_{k-1}, F_{k-1}
OR	$R_{k-1}, \overline{F_{k-1}}$
NAND	R_{k-1}, F_{k-1}
AND2	$\overline{R_{k-1}}, F_{k-1}$

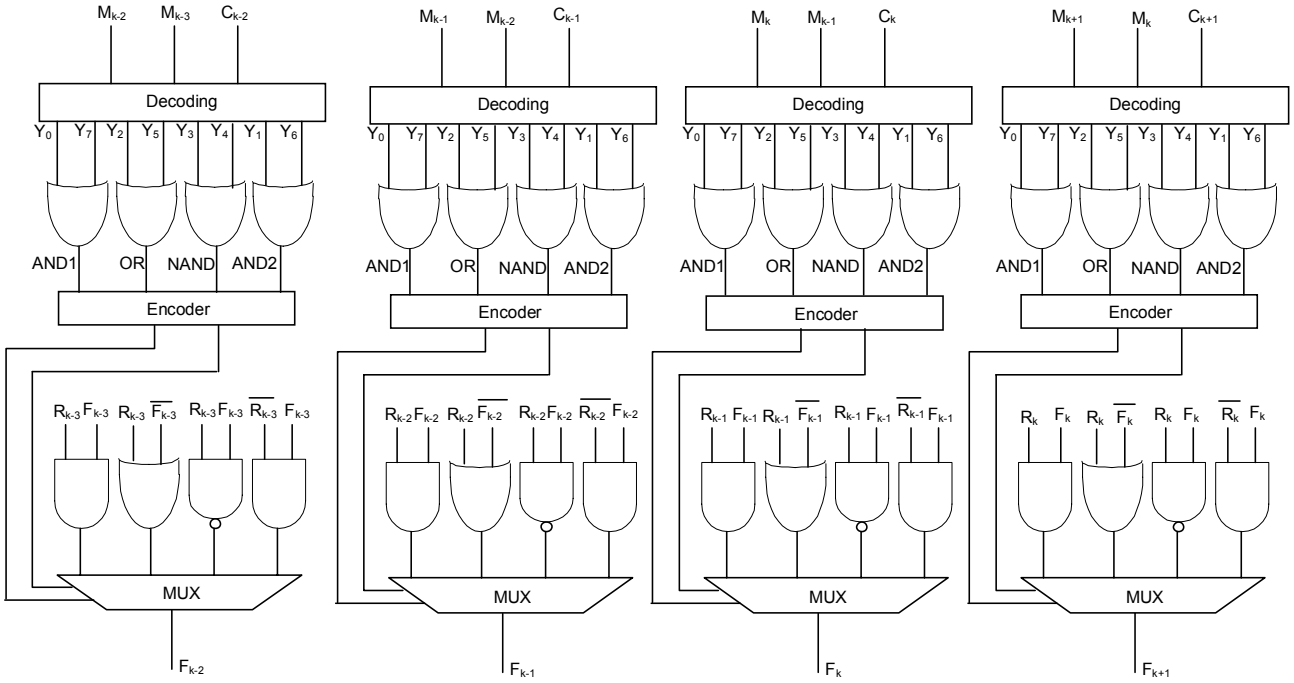

Figure 2 Flag Inversion Cells for Three-Input Addition

Table III Logic Gate Combinations

M_k	M_{k-1}	C_k	Gate
0	0	0	AND1
1	1	1	
0	1	0	OR
1	0	1	
0	1	1	NAND
1	0	0	
0	0	1	AND2
1	1	0	

IV. RESULTS

The parallel prefix adders were incorporated with the extra hardware as described in Section III. This made the prefix adder capable of adding three numbers at the cost of extra hardware. This added flexibility however adds to the functionality of the adder taking advantage of the regular layout and high speed of the prefix structures.

A. Hardware Cost

Each prefix tree utilizes a set of black and gray cells [5] arranged in a regular fashion to obtain the carry signals in parallel, eliminating the rippling effect of the carry.

Each 16-bit parallel prefix adder will therefore have 16 *bit propagate* and *bit generate* cells in the pre-processing stage. The prefix carry tree will comprise of k black cells and 15 gray cells [6]. k is the multiplying factor that determines the number of prefix cells within the tree. It is a variable depending on the kind of tree under consideration. The post processing stage will compute the R bits as a result of adding two operands, A and B using 16 XOR gates. This will be followed by a set of 16 3-8 decoders which are utilized to select the flag logic gates in Table III. Depending on the eight combinations of the constant bits and the carry bits for each bit position, each gate can be selected by a pair of M_k, M_{k-1}, c_k combinations. Due to this redundancy, a stage of 16X4 OR gates is utilized. The outputs of these OR gates will be encoded to act as a control signal to the multiplexers which will finalize the flag bit for that bit position (Fig. 3). Therefore, the extra hardware will also comprise of 16 encoders, 16 multiplexers and 16X4 flag logic gates in addition to the OR gates. The ultimate result is increasing the depth of the prefix adder by 4 logic levels, which will impact the delay of the circuit. Also the dependence of the flag bits on the preceding flag bits affects the critical delay of the circuit.

Each adder was designed with the extra hardware to get reasonable estimates of the performance of this design in terms of area, delay and power.

B. Synthesis Results

An analysis was performed on all adders with regards to, area, delay, and power. The designs are implemented in the TSMC 0.18µm technology System-on-Chip design flow to investigate the power, area, and delay tradeoffs. Synthesis is performed with Cadence Build Gates and Encounter. The nominal operating voltage is 1.8V and simulation is performed at T=25^{o}C. Layouts are generated for each adder design and parasitically extracted to obtain numbers for area, delay and power.

Results for 16-bit parallel prefix adders, without any hardware modifications are shown in Table IV. This set of results will be used as a reference to understand the cost of extra hardware after modification and the impact on the critical delay of the circuit.

Table IV Post Layout Estimates for Conventional Adders

Adder/Parameters	Area (mm2)	Delay (ns)	Power (mW)
Brent-Kung	0.2756	0.27	5.63E-04
Ladner-Fischer	0.2763	0.16	5.81E-04
Kogge-Stone	0.4961	0.04	7.78E-04

Post-Layout Estimates are also obtained for designs where the third operand is a constant. This gives an estimate for the delay and area of the schematic when the third operand never changes. This implementation is based on the design represented in Fig. 1. The results are summarized in Table V [9]. The increase in area for this design is approximately 6%. The rise in delay is accounted for by the additional levels of logic and only increases by approximately 7% compared to the conventional design for Brent-Kung and Ladner-Fischer adders.

Table V Post Layout estimates for Enhanced Flagged Prefix Adders

Adder/Parameters	Area (mm2)	Delay (ns)	Power (mW)
Brent-Kung	0.2921	0.29	8.08E-04
Ladner-Fischer	0.2933	0.19	8.34E-04
Kogge-Stone	0.5462	0.08	1.12E-03

Post-layout estimates for the TIFPA designs are presented in Table VII. Post-layout estimates are also obtained for a 16-bit carry-save adder. The carry-save adder is a popular multi-operand adder and serves as a good benchmark for the proposed design. The last stage in the carry-save adder architecture is a ripple-carry adder.

TABLE VI Post Layout Estimates for Three-Input Adders

Adder/Parameters	Area (mm2)	Delay (ns)	Power (mW)
Brent-Kung	0.3271	0.33	1.26E-03
Ladner-Fischer	0.3183	0.24	1.57E-03
Kogge-Stone	0.5957	0.13	1.89E-03
Carry Save	0.3448	0.3	1.17E-03

In terms of speed, the Ladner-Fischer tree is seen to have a better performance compared to a conventional carry-save adder. The increase in area after incorporating the flag logic is around 15% for all three structures. This stays consistent with each adder design since the flag logic does not change. The Ladner-Fischer tree consumes less area than the carry-save adder even after the extra logic is incorporated. The Kogge-Stone adder is the fastest, but consumes a lot of space. This can be attributed to the large number of prefix cells and higher number of lateral wires at each level of the prefix tree. Figs. 3 and 4 display the results to get further clarification. The Brent-Kung and the carry-save adder have a very close performance in terms of speed.

V. CONCLUSIONS

The technique of generating flag bits to perform three input addition utilizing parallel prefix adders adds flexibility and eliminates the need of a dedicated adder unit to perform multi-operand addition. The speed of this design is favorable over carry-save adders which are conventionally used for multi-operand addition. For 16-bit operand sizes, the Ladner-Fischer tree provides a good compromise with regards to area and delay. The Kogge-Stone adder performs best in terms of speed but the Ladner-Fischer adder consumes less area and is also faster than the carry-save design.

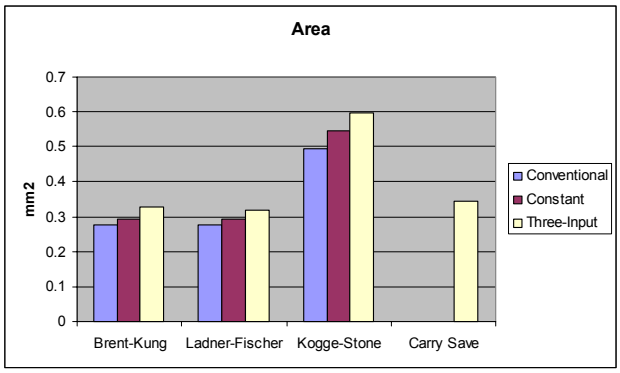

Figure 3. Area Measurements for 16 bit designs

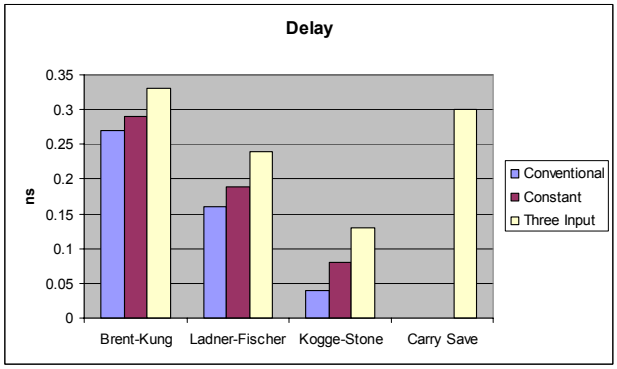

Figure 4. Delay Measurements for 16-bit designs

REFERENCES

[1] R. Brent, and H.Kung. A regular layout for parallel adders. *IEEE Transactions on Computers,* C-31:260-264,1982.

[2] R.E Ladner, and M.J Fischer. Parallel Prefix Computation. *Journal of the ACM,* 27:831-838,1980

[3] P. Kogge, and H..Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trasanctions on Computers,* C-22:783-791,1973.

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

[5] Milos Ercogavac and Tomas Lang. *Digital Arithmetic*. Morgan Kauffmann, First Edition, 2004.

[6] N.Burgess. The flagged Prefix Adder and its Applications in Integer Arithmetic. *Journal of VLSI Signal Processing*, 31(3):263-271,2002.

[7] James Stine, Chris Babb, V. Dave. Constant Addition utilizing Flagged Prefix Structures. In *7th Euromicro Conference on Digital System Design*, 2004.

[8] V.Dave, E. Oruklu, and J. Saniie, "Performance Evaluation of Flagged Prefix Adders for Constant Addition," in 6th IEEE International Conference on Electro/Information Technology, 2006

[9] V.Dave, E. Oruklu, and J. Saniie, "Analysis, Design and Synthesis of Flagged Binary Adders with Constant Addition," in 49th IEEE International Midwest Symposium on Circuits and Systems, 2006.