A Methodology for Generating Application Specific Tree Multipliers

S. Ramanathan Nibedita Mohanty V. Visvanathan

Computer Aided Design Laboratory
Indian Institute of Science
Bangalore-560012.

Abstract

Low latency, application specific multipliers are required for many DSP algorithms. Tree multipliers are an obvious answer to this requirement. However, tree architectures have not been considered for automatic multiplier generation because they have been considered to be irregular. In this paper, a recursive methodology for generating $n \rightarrow 2$ compressors (for n in the range $3 \leq n \leq 64$) using the basic cells $3 \rightarrow 2$ and $4 \rightarrow 2$ is presented. This methodology results in a highly regular and modular layout that can be automatically generated. The performance of the resulting compressors is competitive with detailed full-custom design. The area and latency of the resulting layout for any n is predictable to a fair degree of accuracy.

1 Introduction

Application specific high performance integrated circuits (IC's) for signal processing need multipliers which have specific number of bits (multiplier and multiplicand word size). For example video filters require 8 bits for the signal and 8 to 12 bits for the filter coefficients depending on the algorithm and implementation. Audio filters require 14 to 16 bits for the signal and up to 24 bits for the coefficient. Therefore, automatic data-path synthesis for high-performance signal processing IC's requires a multiplier generator which is bit specific. The traditional approach to this problem is to use array multipliers [1]. But array multipliers suffer from latency which increases linearly with the increase in n (where n is the minimum of multiplier and multiplicand word size).

For low latency applications a tree architecture is attractive. So far, tree architecture has not been considered for automatic multiplier generation because they have been considered to be irregular. However, the recent results of [2], [3] which have been in the context of full custom floating point multiplier design, have modified the architecture to make it regular and relatively easy to layout than previously known. We extend these results in such a way so that it is possible to automatically generate efficient $n \rightarrow 2$ compressors which form the core of the tree multipliers. We introduce a tree architecture with new circuit synthesis and layout synthesis techniques which is extremely regular and easy to generate for n in the range $3 \leq n \leq 64$. Latency of the resulting architecture is the same as that of a Wallace tree [4] except for certain values of n for which the latency is greater than that of the Wallace approach by one $3 \rightarrow 2$ compressor delay. Further, this architecture can be easily pipelined.

The paper is organised as follows. We begin in Section 2 with a review of existing carry-save tree architectures. In Section 3, we present the new tree architecture with the algorithm for circuit synthesis which helps in automating the layout generation for any n in the range $3 \leq n \leq 64$. This is a recursive methodology of assembling compressors for any n from lower sized compressors. Also the predicted latency of the resulting circuit is presented in the form of a procedure for all n in the range $3 \leq n \leq 64$.

In Section 4, we present the layout synthesis procedure with an example layout of $11 \rightarrow 2$ compressor. The hardware resource required for any $n \rightarrow 2$ compressor is also discussed. In Section 5, the features of this new methodology for circuit and layout synthesis is presented. The actual latency found using the timing simulator SLS [5] on the circuit extracted from the layout is compared with the latency predicted using the procedure developed in Section 3. In Section 6, we summarise the work and also provide a brief description of the further developments that are envisaged.

2 Existing Tree Architectures

All existing tree multipliers use the carry-save principle. The important functional stages in a carry-save tree multiplier are the following: (i) the first stage is the partial product generation, (ii) the second stage is the partial product compression stage which forms the core of the tree multiplier and is the focus of this paper, and (iii) the third stage which adds the sum and carry outputs of the second stage to yield the result. We begin our discussion of partial product compression with a review of the existing tree architectures.

The Wallace tree architecture supports fully parallel partial product reduction [4]. The classic 3-input Wallace tree element is a carry-save adder which accepts 3-bit wide operands and exports a 2-bit wide result, i.e., the $3 \rightarrow 2$ compressor takes 3-inputs of same weight and produces 2 outputs, a sum of weight 1 and a carry of weight 2. Given the nature of the $3 \rightarrow 2$ compressor, it is impossible to build a completely regular

0-8186-3180-5/92 $3.00 © 1992 IEEE

tree architecture [2].

To overcome the above mentioned problem with the Wallace tree, one can use a 4→2 compressor as the basic building block [2]. The 2-to-1 reduction of the 4→2 compressor produces a binary tree structure which is much more regular than the Wallace tree which uses only 3→2 compressors. An extension of this idea for any n in the range $3 \leq n \leq 64$ will not give a solution with minimal hardware and latency for all n (except for the case where n is a power of 2).

Song and DeMicheli [3] have used a 9→2 compressor family which consists of a set of compressors that includes 3→2, 4→2, 6→2 and 9→2 compressors to construct a 27→2 compressor for implementing a floating-point double-precision multiplier. An extension of this approach will not give a solution with minimal hardware and latency for all n in the range $3 \leq n \leq 64$ (except for a particular strict subset of multiples of 3 or 4).

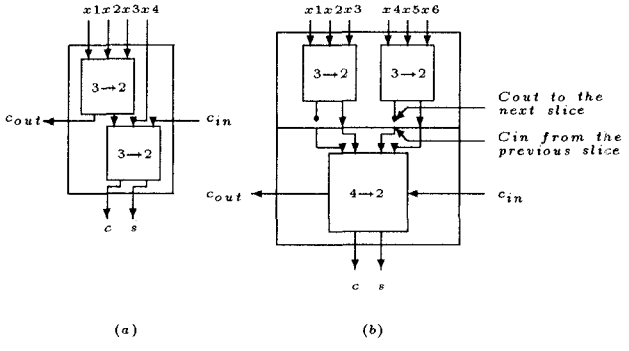

Figure 1: Schematic circuit diagram of (a) 4→2 compressor slice (b) 6→2 compressor slice.

3 New Tree Architecture

From our previous discussion, one can infer that a circuit synthesis methodology for tree multipliers for all n in the range $3 \leq n \leq 64$ cannot be realised from the existing approaches. Our solution to the problem is presented below.

We propose a recursive methodology of assembling a compressor for any n in the range $3 \leq n \leq 64$ from lower sized compressors, with 3→2 and 4→2 compressors as the basic building blocks. This recursive circuit synthesis leads to a highly regular and modular layout. The basic cell library consists of 3→2 and 4→2 compressors. The 3→2 compressor is the standard carry-save adder and the layout is designed using a standard full adder circuit [6]. The 4→2 compressor is constructed using two 3→2 compressors as reported in [2](refer Figure 1(a)).

Given the basic cells 3→2 and 4→2 compressors, we use a recursive algorithm for circuit synthesis for any n→2 compressor for n in the range $3 \leq n \leq 64$ as follows[1]:

[1]The 5→2 compressor is not included in the procedure as it is not required for any other value of n. If a specific 5→2 compressor is required then it can be constructed as a basic cell or one can use a 6→2 compressor with one of its input line grounded.

Algorithm for Circuit Synthesis

TOP if n lies in the range $3 \leq n \leq 11$ then
{
  if n lies in the range $3 \leq n \leq 4$ then
  {
    the required slice is a basic n→2 compressor. /* for $n = 5$ see footnote 1 */
  }
  if n lies in the range $6 \leq n \leq 8$ then
  {
    split n into n_1 and n_2 such that
    $0 \leq |n_1 - n_2| \leq 1$.
    n→2 is now composed of basic cells n_1→2 and n_2→2 in the first stage followed by a 4→2 in the second stage.
  }
  if n lies in the range $9 \leq n \leq 11$ then
  {
    split n into n_1, n_2 and n_3 such that
    $0 \leq |n_1 - n_2|, |n_2 - n_3|, |n_3 - n_1| \leq 1$.
    n→2 is now composed of basic cells n_1→2, n_2→2 and n_3→2 in the first stage followed by 6→2 in the second stage. For $n = 6$ goto TOP.
  }
}
else if n lies in the range $12 \leq n \leq 24$ then
{
  split n into n_1 and n_2 such that
  $0 \leq |n_1 - n_2| \leq 1$
  n→2 is now composed of n_1→2 and n_2→2 in the first stage followed by a 4→2 in the second stage.
  For $n = n_1$ and $n = n_2$ goto TOP.
}
else if n lies in the range $25 \leq n \leq 64$ then
{
  split n into n_1, n_2 and n_3 such that
  $0 \leq |n_1 - n_2|, |n_2 - n_3|, |n_3 - n_1| \leq 1$
  n→2 is now composed of n_1→2, n_2→2 and n_3→2 in the first stage followed by a 6→2 in the second stage. For $n = n_1$, $n = n_2$, $n = n_3$ and $n = 6$ goto TOP.
}

Latency of the New Tree Architecture

The latency[2] of an n→2 compressor is expressed in terms of the number of 3→2 compressor delay. Normalising the latency of the 3→2 compressor as 1 (i.e., $\mathcal{L}[3\rightarrow 2] = 1$), the latency of the 4→2 compressor is 2 (i.e., $\mathcal{L}[4\rightarrow 2] = 2$). Note that a 6→2 compressor synthesised by the above procedure (refer Figure 1(b)) has a latency $\mathcal{L}[6\rightarrow 2] = 3$. The latency of these cells are the same as the corresponding Wallace tree. Latency for any n can be found using the following procedure[3].

[2]Latency is abbreviated as $\mathcal{L}$

[3]Note that $\lceil n \rceil$ denotes the ceiling function of n.

$\mathcal{L}(n)$
{

If n lies in the range $3 \leq n \leq 4$, then
$\mathcal{L}(n)$ = Latency of the basic cell.
else if n lies in the range $6 \leq n \leq 8$, then
$\mathcal{L}(n) = 2 + \mathcal{L}(\lceil n/2 \rceil)$.
else if n lies in the range $9 \leq n \leq 11$, then
$\mathcal{L}(n) = 3 + \mathcal{L}(\lceil n/3 \rceil)$.
else if n lies in the range $12 \leq n \leq 24$, then
$\mathcal{L}(n) = 2 + \mathcal{L}(\lceil n/2 \rceil)$.
else if n lies in the range $25 \leq n \leq 64$, then
$\mathcal{L}(n) = 3 + \mathcal{L}(\lceil n/3 \rceil)$.

}

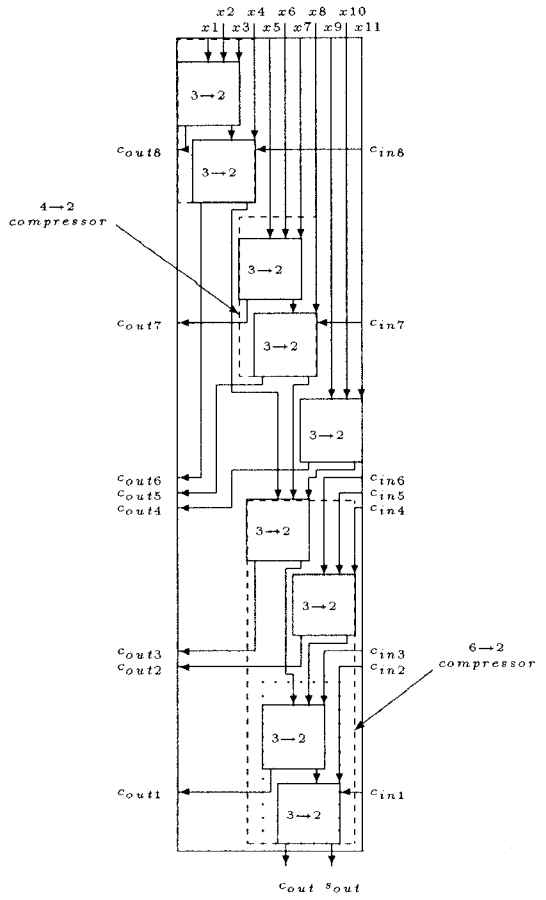

Figure 2: Floorplan of a 11→2 compressor slice.

4 Layout Synthesis

The layout of a 3→2 compressor slice is a resized version of the standard combinational adder layout [6] and implemented in the *C3TU*, double metal single poly 1.6-μm CMOS process using the NELSIS IC design system [5]. It has an area of $92.8\mu m \times 80.8\mu m$. The floorplan of the 4→2 compressor slice (embedded in the floorplan of 11→2 compressor slice as shown in Figure 2) is assembled using two 3→2 compressors and these 3→2 compressors are stacked with C_{in} and C_{out} lines appropriately placed in order to facilitate abutment of slices. The same methodology is used to assemble higher-valued compressor slices. The floorplan of a 6→2 compressor slice (refer Figure 1(b) for the circuit) is also embedded in the floorplan of 11→2 compressor slice shown in Figure 2.

For any general n→2 compressor the circuit synthesis is done recursively using the algorithm described in the previous Section. In the corresponding layout synthesis which is recursive, the compressors used in each stage are placed one below the other followed by a 4→2 or a 6→2 compressor depending on the value of n. This recursion continues until each compressor is split into lower order compressors which are basic cells (namely 3→2 and 4→2 compressors). The final layout for the slice is therefore a stack of appropriately placed 3→2 compressors with C_{in} and C_{out} lines that abut correctly with neighbouring slices.

The floorplan of a 11→2 compressor slice using this synthesis methodology is shown in Figure 2. Following the floorplan from the top of the figure, one can see two 4→2 compressors, a 3→2 compressor and then a 6→2 compressor. The 6→2 compressor itself consists of two 3→2 compressors followed by a 4→2 compressor. Note that in the actual layout there is no vertical space between the compressor slices, i.e., all the basic cells in a compressor slice abut vertically. The spacing has been introduced in Figure 2 merely for a clear presentation of the routing.

Hardware Resources

Let $H_{3\rightarrow 2}$ and $W_{3\rightarrow 2}$ denote the height and width of a 3→2 compressor cell respectively. In our case $H_{3\rightarrow 2} = 80.8\mu m$ and $W_{3\rightarrow 2} = 92.8\mu m$. Let M_w and M_s be the metal width and metal separation respectively. In *C3TU* these parameters are $2.4\mu m$ each. The height of an n→2 compressor slice is given by $H_{slice} = (n-2) \times H_{3\rightarrow 2}$ and the width is given by $W_{slice} = W_{3\rightarrow 2} + (n-3) \times [M_w + M_s]$. For an unsigned $(m \times n)$ tree multiplication whose outputs are in carry and save form, one would require $(m+n-1)$ number of n→2 compressor slices. The area of such a multiplier is given by $Area_{mult} = (m+n-1) \times W_{slice} \times H_{slice}$.

5 Discussion and Evaluation

The features of the new methodology for circuit and layout synthesis of the tree architecture can be summarised as follows:

- No input line to any stage in a n→2 compressor slice is routed directly to the next stage without being compressed at that stage. This feature has enabled us to come up with a new recursive methodology for circuit and layout synthesis which results in a highly regular and modular tree architecture and facilitates easy automatic layout generation.

- The number of basic 3→2 compressor slices that are used in the construction of an n→2 compressor slice is $(n-2)$ which is same as the number of

3→2 compressor slices that are required for the Wallace tree.

- The latency of the proposed tree architecture is the same as that of a Wallace tree for all n in the range $3 \leq n \leq 64$ except for the following 18 values of n: 13, 19, 28, 37 to 42, 55 to 63. The latency for these n listed here is greater than Wallace approach by one 3→2 compressor delay.

- The 8→2 compressor obtained using this circuit and layout synthesis procedure is the same as that reported in [2]. The 6→2, 9→2 and 27→2 compressors obtained using this methodology are the same as that reported in [3]. Thus, existing compressors are special cases of our methodology.

- It is easy to pipeline this architecture by considering a single slice of the multiplier. Depending on the degree of pipelining required, the appropriate number of pipelining registers are introduced and retimed to pipeline the slice into stages with equal delay.

- Extension to two's complement number system is straightforward. The difference lies in the number of slices of n→2 compressor required for 2's complement multiplier and partial product generation based on Baugh-Wooley algorithm.

- Since there are only two basic cells, namely 3→2 and 4→2 compressor slices, any advances in circuit design or technology that lead to better basic cells can be easily incorporated into our synthesis methodology.

The delay of the 3→2 compressor simulated using SPICE at 27°C is 3.44ns. The simulation carried out in timing simulator SLS [5] gives a delay of 3.6ns. The higher order n→2 compressors are simulated using the timing simulator SLS. The comparison table for predicted latency using the procedure discussed before and the actual value measured by the timing simulator SLS on the circuit extracted from the layout (which takes into account the parasitics capacitances and resistances) is given below. Note the effect of routing that results in an actual delay that is more than that predicted by the procedure of Section 3. The additional delay can in fact also be predicted (without simulation) since the critical input line and its length can be calculated for each value of n as part of the synthesis procedure.

n→2 compressor	*Predicted Latency*	*Actual Latency*
4→2	7.2ns	7.4ns
6→2	10.8ns	12.58ns
7→2	14.4ns	16.13ns
8→2	14.4ns	17.08ns
9→2	14.4ns	17.21ns
10→2	17.6ns	20.30ns
11→2	17.6ns	20.8ns

Table 1: Latency comparison for certain values of n.

6 Conclusions and Future work

We have presented a new tree architecture with a recursive algorithm for circuit and layout synthesis which helps in easy automatic layout generation for any n→2 compressor for n in the range $3 \leq n \leq 64$. The architecture leads to a regular and modular layout and hence easy layout generation. Further, the recursive nature of the synthesis procedure makes it very easy to assemble a high order compressor, once the lower order compressors that it needs are ready. The area for any n→2 compressor is predicted accurately. The latency for all n→2 compressors can also be predicted. It has been shown that performance of the compressors is close to what is achievable with detailed full-custom design. The methodology has been used to develop various sized compressors, including a 37→2 compressor.

The present layout synthesis methodology suffers from two drawbacks, namely the effective multiplier area has a parallelogram shape and hence results in area of the multiplier being used inefficiently and secondly the partial product generation has to be done external to the partial compression area. We are currently addressing these two issues by pursuing a layout synthesis methodology which is a generalisation of the layout technique used in [7]. In this alternate technique the partial product generation will be incorporated along with the compression. Initial investigation indicate that the resulting layout will be nearly square with significant overall area reduction.

Acknowledgements

This work was funded in part by grants from the Department of Electronics, Government of India.

References

[1] "European Silicon Structures (ES2) - Cell Libraries," Oct 1991.

[2] M. R. Santoro and M. A. Horowitz, "SPIM: A pipelined 64×64-bit iterative multiplier," *IEEE Journal of Solid-State Circuits*, Vol. 24, No. 2, pp. 487-483, Apr 1989.

[3] P. J. Song and G. D. Michelli, "Circuit and architecture trade-offs for high-speed multiplication," *IEEE Journal of Solid-State Circuits*, Vol. 26, No. 9, pp. 1184-1198, Sep 1991.

[4] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transaction of Electronic Computers*, Vol. EC-13, pp. 14-17, Feb 1964.

[5] P. van der Wolf et.al., *An Introduction to the NELSIS IC Design System*, Delft University of Technology, The Netherlands, Aug 1990.

[6] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley Publishing Company, 1985.

[7] T. Sato, M. Nakajima, T. Sukemura, G. Goto, "A regularly structured 54-bit modified-Wallace-tree multiplier," *IFIP Transactions A1, VLSI91*, pp. 1-9, North-Holland, 1992.