Implementation of Multioperand Redundant Adders Using Compressor Trees

A. MADHU
M.Tech Student
Dept of ECE
Sri Indu College of Engineering & Technology
Hyderabad

G. RAJ KUMAR
Assistant Professor
Dept of ECE
Sri Indu College of Engineering & Technology
Hyderabad

K. ASHOK BABU
Professor & Head
Dept of ECE
Sri Indu College of Engineering & Technology
Hyderabad

Abstract—Although redundant addition is widely used to design parallel multi operand adders for ASIC implementations, the use of redundant adders on Field Programmable Gate Arrays (FPGAs) has generally been avoided. The main reasons are the efficient implementation of carry propagate adders (CPAs) on these devices (due to their specialized carry-chain resources) as well as the area overhead of the redundant adders when they are implemented on FPGAs. This paper presents different approaches to the efficient implementation of generic carry-save compressor trees on FPGAs. They present a fast critical path, independent of bit width, with practically no area overhead compared to CPA trees. Along with the classic carry-save compressor tree, we present a novel linear array structure, which efficiently uses the fast carry-chain resources. This approach is defined in a parameterizable HDL code based on CPAs, which makes it compatible with any FPGA family or vendor. A detailed study is provided for a wide range of bit widths and large number of operands. Compared to binary and ternary CPA trees, speedups of up to 2.29 and 2.14 are achieved for 16-bit width and up to 3.81 and 3.11 for 64-bit width.

Keywords—Computer arithmetic, reconfigurable hardware, compressor trees, multi operand addition, redundant representation, carry-save adders

I. INTRODUCTION

The use of Field Programmable Gate Arrays (FPGAs) to implement digital circuits has been growing in recent years. In addition to their reconfiguration capabilities, modern FPGAs allow high parallel computing. FPGAs achieve speedups of two orders of magnitude over a general-purpose processor for arithmetic intensive algorithms [1]. Thus, these kinds of devices are increasingly selected as the target technology for many applications, especially in digital signal processing [2], cryptography [3], and much more. Therefore, the efficient implementation of generalized operators on FPGAs is of great relevance. The typical structure of an FPGA device is a matrix of configurable logic elements (LEs), each one surrounded by interconnection resources. In general, each configurable element is basically composed of one or several n-input lookup tables (N-LUT) and flip-flops. However, in modern FPGA architectures, the array of LEs has been augmented by including specialized circuitry, such as dedicated multipliers, block RAM, and so on.

Multi operand addition appears in many algorithms, such as multiplication, filters, SAD and others. To achieve efficient implementations of this operation, redundant adders are extensively used. Redundant representation reduces the addition time by limiting the length of the carry-propagation chains. The most usual representations are carry-save (CS) and signed-digit (SD). A CS adder (CSA) adds three numbers using an array of Full-Adders (FAs), but without propagating the carries. In this case, the FA is usually known as a 3:2 counter. The result is a CS number, which is composed of a sum-word and a carry-word. Therefore, the CSA result is obtained without any carry propagation in the time taken by only one FA. The addition of two CS numbers requires an array of 4:2 compressors, which can be implemented by two 3:2 counters. The conversion to non redundant representation is achieved by adding the sum and carry word in a conventional CPA.

In this paper, we study the efficient implementation of Multi operand redundant compressor trees in modern FPGAs by using their fast carry resources. Our approaches strongly reduce delay and they generally present no area overhead compared to a CPA tree. Moreover, they could be defined at a high...
level based on an array of standard CPAs. As a consequence, they are compatible with any FPGA family or brand, and any improvement in the CPA system of future FPGA families would also benefit from them. Furthermore, due to its simple structure, it is easy to design a parametric HDL core, which allows synthesizing a compressor tree for any number of operands of any bit width. Compared to previous approaches, our design presents better performance, is easier to implement, and offers direct portability.

This paper is organized as follows: Section 2 reviews previous work on redundant addition on FPGAs. In Section 3, we present our proposals for implementing multi operand redundant compressor trees on FPGAs and a theoretical analysis of their performance. In Section 4, we compare the results of implementation using different approaches. Finally, the conclusions are presented in Section 5.

II. PREVIOUS WORK

The optimization of redundant addition on FPGAs has been addressed using different approaches:

1. The efficient mapping of isolated redundant adders on an Inner structure of FPGAs [5], [6];
2. Utilizing different heuristics to design compressor trees based on bit counters [7];
3. Proposing hardware modifications to existing FPGA architectures [8], [9]; and
4. Specific applications [10]

III. COMPRESSOR TREES ON FPGAs

In this section, we present different approaches to efficiently map CS compressor trees on FPGA devices. In addition, approximate area and delay analysis are conducted for the general case.

Let us consider a generic compressor tree of $N_{op}$ input operands with $N$ bit width each. We also assume the same bit width for input and output operands. Thus, input operands should have previously been zero or sign extended to guarantee that no overflow occurs.

3.1. Regular CS compressor tree design

The design of a multi operand CS compressor tree attempts to reduce the number of levels in its structure. The implementation of a generic CS compressor tree requires $[N_{op}/2]-1$ 4:2 compressors (because each one eliminates two signals), whereas a carry-propagate tree uses $N_{op}-1$ CPAs (since each one eliminates one signal). If we bear in mind that a 4:2 compressor uses practically double the amount of resources as CPAs [28], both trees basically require the same area. On the other hand, the speed of a compressor tree is determined by the number of levels required. In this case, because each level halves the number of input signals, the critical path delay ($D$) is approximately

$$L_{4:2}=[\log_2(N_{op})]-1$$

$$D=L_{4:2} \cdot d_{4:2}$$

Where $L_{4:2}$ is the number of levels of the compressor tree and $d_{4:2}$ is the delay of a 4:2 compressor level (including routing).

We now generalize this idea to compressors of any size by proposing a different approach based on linear arrays. This reduces the critical path of the compressor tree when it is implemented on FPGAs with specialized carry-chains.

3.2 Linear Array Structure

In the previous approach, specialized carry resources are only used in the design of a single 4:2 compressor, but these resources have not been considered in the design of the whole compressor tree structure. However, in our case, given the two output words of each adder (sum-word and carry-word), only the carry-word is connected from each CSA to the next, whereas the sum words are connected to lower levels of the array.

Fig. 1 shows an example for a 9:2 compressor tree designed using the proposed linear structure, where all lines are $N$ bit width buses, and carry signal are correctly shifted. For the CSA, we have to distinguish between the regular inputs (A and B) and the carry input ($C_i$ in the figure), whereas the dashed line between the carry input and output represents the fast carry resources. With the exception of the first CSA, where $C_i$ is used to introduce an input operand, on each CSA $C_i$ is connected to the carry output ($C_o$) of the previous CSA, as shown in Fig. 1. Thus, the whole carry-chain is preserved from the input to the output of the compressor tree (from I0 to Cf). First, the two regular inputs on each CSA are used to add all the input operands (Iii). When all the input operands have been introduced in the array, the partial sum-words (Si) previously generated are then added in order (i.e.,
the first generated partial sums are added first) as shown in Fig. 1. In this way, we maximize the overlap between propagation through regular signals and carry-chains.

Regarding the area, the implementation of a generic compressor tree based on N bit width CSAs requires Nop – 2 of these elements (because each CSA eliminates one input signal). Therefore, considering that a CSA could be implemented using the same number of resources as a binary CPA (as shown below), the proposed linear array, the 4:2 compressor tree, and the binary CPA tree have approximately the same hardware cost.

Fig 2. Time model of the proposed CS 9:2 compressor tree.

In relation to the delay analysis, from a classic point of view our compressor tree has Nop - 2 levels. This is much more than a classic Wallace tree structure and, thus, a longer critical path. Nevertheless, because we are targeting an FPGA implementation, we temporarily assume that there is no delay for the carry-chain path. Under this assumption, the carry signal connections could be eliminated from the critical path analysis and our linear array could be represented as a hypothetical tree, as shown in Fig. 2 (where the carry-chain is represented in gray). To compute the number of effective time levels (ETL) of this hypothetical tree, each CSA is considered a 2:1 adder, except for the first, which is considered a 3:1 adder. Thus, the first level of adders is formed by the first \([(N_{\text{op}}-1)/2]\) CSAs (which correspond to partial addition of the input operands). This first ETL produces \([(N_{\text{op}}-1)/2]\) partial sum-words that are added to a second level of CSAs (together with the last input operand if Nop is even) and so on, in such a way that each ETL of CSAs halves the number of inputs to the next level. Therefore, the total ETLs in this hypothetical tree are

\[
L = \lfloor \log_2(N_{\text{op}}-1) \rfloor
\]

and the delay of this tree is approximately L times the delay of a single ETL.

The delay of the carry-chain is comparatively low, but not null. Let us consider just two global values for the delay: \(d_{\text{carry}}\), which is the delay for the path between the carry inputs (Ci) of two consecutive CSAs (see Fig. 2); and \(d_{\text{sum}}\), which is the delay from one general input of a CSA (Aor B) to a general input of a directly connected CSA, i.e., the time taken by the data to go from an ETL to the next one (see Fig. 2). Even under this simplified scenario, it is unfeasible to obtain a general analytical expression for the delay of our compressor tree structure. On each ETL, the propagation through carry-chains and the general paths are overlapped and this overlap depends on multiple factors. First, it depends on the relative relationship between the values of \(d_{\text{carry}}\) and \(d_{\text{sum}}\) (which is associated with the FPGA family used). Second, it depends on the number of operands that affect both the delay of the carry-chain of each ETL and the internal structure of the hypothetical tree. Even though the former could be expressed as an analytical formula, the latter cannot be expressed in this way (especially when N_{\text{op}} - 1 is not a power of two). However, it is possible to bound the critical path delay by considering two extreme options.

One extreme situation occurs when the delay of the whole carry-chain corresponding to each ETL (\(d_{\text{carry}}\times\) the number of CSAs of the ETL) is always greater than the delay from an ETL to the next one (\(d_{\text{sum}}\)). In this case, the timing behavior corresponds to a linear array and the critical path is represented in Fig. 3. Initially, the first carry out signal is generated from I1, I2, I3 in the first CSA and then the carry signal is propagated through the whole carry-chain until the output. Thus, the delay of the critical path has two components corresponding to the generation of the first carry signal and the propagation through the carry-chain. If we characterize the delay from a general input to the carry output in the first CSA (including later routing) as \(d_{\text{sum}}\), then the estimated lower bound for the delay of the compressor tree is

\[
D_{\text{low}}=d_{\text{sum}}+(N_{\text{op}}-3).d_{\text{carry}}
\]

\(d_{\text{sum}}\) is usually one order of magnitude greater than \(d_{\text{carry}}\).
The other extreme situation occurs when the delay of the carry-chain on each ETL (d_{carry} x the number of CSAs of the ETL) is always less than the delay from an ETL to the next one (d_{sum}). In this case, we obtain the hypothetical tree presented in Fig. 2. The critical path is shown in Fig. 4. It begins as in the previous case: A first carry generation from the general inputs and carry propagation through the carry-chain of the first ETL, because all internal general paths of the CSAs corresponding to the first ETL are updated in parallel and they need the carry input to generate the output signals. However, due to the initial premise, and because sum signals arrive at the first CSAs of each ETL earlier than at the last ones, after the first ETL the critical path goes from one ETL to the next one through the general routing. Therefore, the delay of the critical path has three components corresponding to the generation of the first carry signal, propagation through the carry-chain of the first ETL, and propagation across the remaining of the ETLs. In this case, taking into account that the delay between the carry input and the sum output (including later routing) is approximately d_{carry}, the estimated upper bound for the delay of the compressor tree is d_{sum} x the number of ETLs + d_{carry} x the number of CSAs of the first ETL, that is

\[ D_{up} = \log_2(N_{op} - 1) d_{sum} + \frac{(N_{op} - 1)}{2} d_{carry} \] (5)

This scenario is very frequent because if the delay through the carry-chain of the first ETL is less than d_{sum}, then the next ETLs hold this condition. Thus, only the following condition:

\[ d_{sum} \geq \frac{N_{op} - 1}{2}, d_{carry} \] (6)

needs to be fulfilled. Therefore, for values of N_{op} up to 2d_{sum}/d_{carry}, the delay of the linear array compressor tree is very close to D_{up}. However, for greater values of N_{op}, the delay of the compressor tree is between D_{low} and D_{up}, because the hypothetical structure of the compressor tree is a mix of both situations: The first CSAs form a linear array until the delay of the carry-chain in an ETL is lower than d_{sum}, and then the remaining ones form a hypothetical tree.

3.3 High-Level Implementation

A high-level description of an FPGA design using HDL presents some significant advantages, such as portability among different families (even from different brands), easier generalization, lower error production during the design process, and so on. A disadvantage is that control over the final implementation is lost, which could produce unexpected results. Thus, the HDL code needs to be carefully selected, especially when specific inner resources of the FPGA are used. In fact, a low-level design of the basic building block, which is instantiated from the higher level circuit, is sometimes the only way to achieve efficient implementations. Thus, the design is anchored to a specific FPGA inner structure.

As an example, a more detailed description of the linear array structure for an N-bit width 5:2 compressor tree is shown in Fig. 5. This circuit could be implemented by connecting the FAs (which have to be designed at a low level) as shown in Fig. 6b, while carefully choosing the routing nets to guarantee the preservation of the carry chains. However, at a high level, the only way to guarantee the utilization of the specialized carry logic and the preservation of the carry-chains is by using standard CPAs as basic building blocks. In addition, this allows portability between different FPGA families. A careful examination of Fig. 5b shows that the linear array of three CSAs (see Fig. 5a) can be interpreted as an array of 3-bit width CPAs diagonally deployed (see Fig. 5c'), as highlighted in Fig. 5b for one of them. Hence, the proposed linear compressor tree of N_{op} inputs could be implemented by using an array of N_{op}-2-bit width CPAs. Although both circuits are exactly the same, we have simply provided a new interpretation at the semantic level. However, this new interpretation allows the implementation of the proposed circuit by means of standard CPAs instantiation.

3.4 Improvement for Ternary Adders

To improve the performance of multioperand addition, the newest FPGA families, such as Virtex-
5/7 or Stratix-II/V, can efficiently implement ternary addition \((A+B+C=D)\). On these FPGAs, a ternary adder requires the same amount of resources as a simple 2-input adder while showing a similar speed. Since each ternary adder eliminates two operands, the number of adders required for a compressor tree is \(((N_{op}-1)/2\), which is almost half the amount needed in the binary case. On the other hand, the number of levels is
\[
L_{3:1} = \lceil \log_3(N_{op}) \rceil \tag{7}
\]
which is considerably faster than the one based on binary adders. Therefore, the ternary adder is preferred to implement multioperand parallel addition when targeting these devices. We now present how our linear array compressor tree design is adapted to take advantage of this new resource.

The ternary adder structure is based on the integration of an initial 3:2 CSA together with a binary CPA in the same LEs. Thus, a 1-bit element of the ternary adder has three operand input signals, one sum output signal and two carry signals having two inputs and two outputs. One carry signal (cA), which is related to the CSA, has limited delay, i.e., the generation of this carry signal does not depend on the previous value of that carry. The other (cB), which is related to the CPA, forms a standard carry-chain. Hence, as shown in Fig.6, an efficient compressor tree could be created by constructing a linear array of 5:3 CAs. Similar to the previous approach (see Section 3.2), and with the exception of the first CA, where two additional operands are introduced, the two carry input signals are connected to the corresponding carry outputs of the previous adder (preserving the carry-chain constituted by cB signal), whereas the partial sum-words generated are added in the same order as they were produced after adding all the input operands.

The number of operands \(N_{op}\) should be an odd number and one input zero-word must be used when necessary. In addition, the last partial sum requires a final addition with two zero-words to guarantee a zero output in the last cA signals (see Fig.6), because only the last cB signals make up the final carry-word.

Taking all these considerations into account, along with the fact that the first 5:3 CA adds five operands, the number of 5:3 CAs required to implement a compressor tree is
\[
N_{Add} = \frac{N_{op}-1}{2} \tag{8}
\]
because two operands are eliminated by each CA. Given that a classic carry-propagate compressor tree requires the same number of ternary adders, and each \(N\)-bit width 5:3 CA uses the same FPGA resources as a ternary adder, the proposed CS compressor tree has the same area cost as its equivalent carry-propagate compressor tree.

This new compressor tree design (see example in Fig. 6) could also be implemented at a high-level description using ternary CPAs as a basic building block (see Fig.5). n ternary adders of \(N_{Add}\)-bit width diagonally arranged are required to implement it (simplifying the extreme cases). Once again, the most significant sum-bit of each ternary adder comprises the sum-word of the compressor tree, whereas the last cB out is the final carry-word. Except for the ending adders, each ternary adder sums one bit of each operand and partial sum, varying the bit weight depending on its relative position.

This new compressor tree design could also be implemented at a high-level description using ternary CPAs as a basic building block (see Fig.5). n ternary adders of \(N_{Add}\)-bit width diagonally arranged are required to implement it (simplifying the extreme cases). Once again, the most significant sum-bit of each ternary adder comprises the sum-word of the compressor tree, whereas the last cB out is the final carry-word. Except for the ending adders, each ternary adder sums one bit of each operand and partial sum, varying the bit weight depending on its relative position.

This new compressor tree design could also be implemented at a high-level description using ternary CPAs as a basic building block (see Fig.5). n ternary adders of \(N_{Add}\)-bit width diagonally arranged are required to implement it (simplifying the extreme cases). Once again, the most significant sum-bit of each ternary adder comprises the sum-word of the compressor tree, whereas the last cB out is the final carry-word. Except for the ending adders, each ternary adder sums one bit of each operand and partial sum, varying the bit weight depending on its relative position.

This new compressor tree design could also be implemented at a high-level description using ternary CPAs as a basic building block (see Fig.5). n ternary adders of \(N_{Add}\)-bit width diagonally arranged are required to implement it (simplifying the extreme cases). Once again, the most significant sum-bit of each ternary adder comprises the sum-word of the compressor tree, whereas the last cB out is the final carry-word. Except for the ending adders, each ternary adder sums one bit of each operand and partial sum, varying the bit weight depending on its relative position.

This new compressor tree design could also be implemented at a high-level description using ternary CPAs as a basic building block (see Fig.5). n ternary adders of \(N_{Add}\)-bit width diagonally arranged are required to implement it (simplifying the extreme cases). Once again, the most significant sum-bit of each ternary adder comprises the sum-word of the compressor tree, whereas the last cB out is the final carry-word. Except for the ending adders, each ternary adder sums one bit of each operand and partial sum, varying the bit weight depending on its relative position.

3.5 Pipelining

One of the main advantages of CS compressor trees is that they are very suitable for pipelining. An \(M\)-stage pipeline compressor tree of \(L\) levels of compressors is implemented simply by introducing one level of registers for each \(L/M\) levels of compressors. However, the proposed designs are defined using an array of CPAs, which complicates the introduction of registers in the linear array compressor tree. However, the proposed approach is still easy to apply in pipeline designs. The pipeline compressor tree is designed by utilizing smaller linear array compressor trees as basic building blocks, which are registered in the input or output. For instance, a two-stage 18:2 compressor tree is built by using three 6:2 linear array compressor trees.
IV. RESULT ANALYSIS

To measure the effectiveness of the designs presented in this paper, we have developed two generic VHDL modules implementing the proposed compressor tree structures: First, the linear array implemented by using CPAs (binary and ternary) and, second, the 4:2 compressor tree using the design of the compressor presented in [28]. Both modules provide the output result in CS format and allow the selection of different parameters such as: The number of operands (Nop), the number of bits per operand (N), and the basic building blocks (i.e., binary or ternary adder) for the linear array. For the purposes of comparison, similar modules, which implement classic adder tree structures based on binary CPAs and ternary CPAs, have also been developed. All these modules were simulated using Modelsim SE 6.3f and they were synthesized using Xilinx ISE 9.2, targeting Spartan-3A, Virtex-4, and Virtex-5 devices. A generic ternary adder module was designed following the recommendations of Xilinx, because this adder is not automatically supported by ISE 9.2. Furthermore, to investigate their portability, compressor trees based on ternary CPAs were also synthesized to target the Altera Stratix-II family. In this case, the ternary adders are directly instantiated at a high level.

For the sake of simplicity, of the two FPGA families tested (Spartan-3A and Virtex-4), only the results corresponding to the Virtex-4 family are presented, because the results are very similar for both families. On these FPGAs, the compressor trees based on ternary adders are not efficiently implemented, and thus, we have only tested the ones based on binary adders. For purposes of clarity, let us denote as CPA tree the classic tree structure based on CPAs, OUR array the proposed linear array structure based on 3:2 CSAs, and 4:2 tree the classic tree structure based on a 4:2 compressors.

Regarding the area, Fig. 7 shows the number of LUTs required by the different compressor tree structures when varying Nop from 4 to 128 operands, for 16- and 64-bit widths. With the exception of 4- and 5-operand compressor trees, which we consider separately, the area used for the three compressor trees is very similar and varies linearly with the number of operands and the bit width, as expected. Specifically, the area of CPA tree and OUR array is practically identical, whereas the 4:2 tree requires a little more area (up to 6 percent for a 16-bit width and up to 2 percent for a 64-bit width), due to the implementation of boundary bits on the 4:2 CA. Let us now consider the cases of four and five operands. The CPAs involved in the implementation of OUR array are only 2- and 3-bit width for all operand sizes. Given this small size, the synthesis tool implements these CPAs by exclusively using LUTs, and not the specialized carry-chain, because this produces faster circuits. As a consequence, there is an increase in area for these particular cases (as shown in Fig. 7), which could be eliminated by manually designing low-level CPAs or by changing the synthesis tool.

Regarding speed, Fig. 8 shows the speedup achieved when using OUR array instead of CPA tree, for different numbers of operands and varying the number of bits from 16- to 96-bit width. As can be seen, OUR array is always faster than CPA tree, and the speedup practically grow linearly in relation to number of bits. This is due to the linear dependency of the delay on the CPAs, whereas the delay remains constant for CSAs. Thus, although the speedup achieved for 16-bit width is moderate, i.e., 12 to 50 percent faster (in the range of values selected for Nop and excluding 4- and 5-operand compressor trees), for 64-bit width the speedup ranges from 44 to 104 percent faster. As mentioned above, 4- and 5-operand compressor trees achieve high speed up due to the small CPAs required.

Fig. 7. Area (LUTs) of the different compressor tree approaches implemented on Virtex-4 when varying the number of input operands (Nop for (a) 16-bit width and (b) 64-bit width.

Fig. 8. Speedup achieved on Virtex-4 by using OUR array instead of CPA tree for different number of operand (Nop) when varying the number of bits (N).
V. CONCLUSIONS

Efficiently implementing CS compressor trees on FPGA, in terms of area and speed, is made possible by using the specialized carry-chains of these devices in a novel way. Similar to what happens when using ASIC technology, the proposed CS linear array compressor trees lead to marked improvements in speed compared to CPA approaches and, in general, with no additional hardware cost. Furthermore, the proposed high-level definition of CSA arrays based on CPAs facilitates ease-of-use and portability, even in relation to future FPGA architectures, because CPAs will probably remain a key element in the next generations of FPGA.

REFERENCES


AUTHOR’S PROFILE

- A. Madhu pursuing Master of Technology (VLSI DESIGN) from JNTU-H, he received B-tech. [ECE] from JNTU-H. His research interests are VLSI, DSP & Communication

- K. Ashok Babu Prof & Head of Sri Indu College of Engineering & Technology, He has published number of papers in International & National journals

- V. Rajukumar Assoc.Prof of Sri Indu College of Engineering & Technology, Hyderabad He received B.E. [ECE] from JNTUH, M.Tech. [Embedded Systems] from JNTUH, His research interests are ES, VLSI, and DSP.