International Journal of Scientific Engineering and Technology Research

ISSN 2319-8885
Vol.04,Issue.04
February-2015,
Pages:0625-0628

www.ijsetr.com

Multi Operand Redundant Adders on FPGAs

METHUKU RAMU[1], G. DEVENDHAR[2]

[1]PG Scholar, TRRITS, Hyderabad, TS, India, E-mail: methukuramu@gmail.com.

[2]Assistant Professor, TRRITS, Hyderabad, TS, India.

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, increases speedups for 16-bit width.

Keywords: Modelsim 6.4b, Xilinx ISE 10.1LanguagesVerilog HDL.

I. INTRODUCTION

As the scale of integration keeps growing, more and more sophisticated signal processing systems are being implemented on a VLSI chip. These signal processing applications not only demand great computation capacity but also consume considerable amount of energy. While performance and Area remain to be the two major design tolls, power consumption has become a critical concern in today's VLSI system design[1]. The need for low-power VLSI system arises from two main forces. First, with the steady growth of operating frequency and processing capacity per chip, large currents have to be delivered and the heat due to large power consumption must be removed by proper cooling techniques. Second, battery life in portable electronic devices is limited. Low power design directly leads to prolonged operation time in these portable devices. Addition usually impacts widely the overall performance of digital systems and a crucial arithmetic function. In electronic applications adders are most widely used. Applications where these are used are multipliers, DSP to execute various algorithms like FFT, FIR and IIR. Wherever concept of multiplication comes adders come in to the picture. As we know millions of instructions per second are performed in microprocessors.

So, speed of operation is the most important constraint to be considered while designing multipliers. Due to device portability miniaturization of device should be high and power consumption should be low. Devices like Mobile, Laptops etc. require more battery backup. So, a VLSI designer has to optimize these three parameters in a design. These constraints are very difficult to achieve so depending on demand or application some compromise between constraints has to be made. Ripple carry adders exhibits the most compact design but the slowest in speed. Whereas carry look ahead is the fastest one but consumes more area. Carry select adders act as a compromise between the two adders. In 2002, a new concept of hybrid adders is presented to speed up addition process by Wang et al. that gives hybrid carry look-ahead/carry select adders design. In 2008, low power multipliers based on new hybrid full adders is presented.

II. NEED FOR LOW POWER DESIGN

The design of portable devices requires consideration for peak power consumption to ensure reliability and proper operation. However, the time averaged power is often more critical as it is linearly related to the battery life. There are four sources of power dissipation in digital CMOS circuits: switching power, short-circuit power, leakage power and static power. The following equation describes these four components of power

$$P_{avg} = P_{switching} + P_{short-circuit} + P_{leakage} + P_{static}$$
$$= \alpha C_L V_{dd} V_s f_{ck} + I_{sc} V_{dd} + I_{leakage} V_{dd} + I_{static} V_{dd} \quad (1)$$

A. Low Voltage

Power consumption is linearly proportional to voltage swing (V_s) and supply voltage (V_{dd}) as indicated in Eq. (2.5). For most CMOS logic families, the swing is typically rail-to-rail. Hence, power consumption is also said to be proportional to the square of the supply voltage, V_{dd}.

Therefore, lowering the V_{dd} is an efficient approach to reduce both energy and power, presuming that the signal voltage swing can be freely chosen. This is, however, at the expense of the delay of circuits. The delay, td, can be shown to be proportional to. The exponent is between 1 and 2. It tends to be closer to 1 for MOS transistors that are in deep sub-micrometer region, where carrier velocity saturation may occur. increases toward 2 for longer channel transistors. The current technology trends are to reduce feature size and lower supply voltage. Lowering V_{dd} leads to increased circuit delays and therefore lower functional throughput. Smaller feature size, however, reduces gate delay, as it is inversely proportional to the square of the effective channel length of the devices. In addition, thinner gate oxides impose voltage limitation for reliability reasons. Hence, the supply voltage must be lowered for smaller geometries. The net effect is that circuit performance improves as CMOS technologies scale down, despite of the V_{dd} reduction. Therefore, the new technology has made it possible to fulfill the contradicting requirements of low-power and high throughput.

III. EXISTING SYSTEM

In this paper, we study the efficient implementation of multioperand 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.

IV. PROPOSED SYSTEM

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 Nop 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. A detailed analysis of the number of leading guard bits required for multi operand CS addition is provided.

Applications:

- Digital signal processors.
- Microprocessors.
- Controllers.

Objective: The area overhead of the redundant adders when they are implemented on FPGAs Present a fast critical path, independent of bit width, with practically no area overhead compared to CPA trees.

A. Deign Implementation

The classic design of a multi-operand CS compressor tree attempts to reduce the number of levels in its structure. The 3:2 counter or the 4:2 compressor are the most widely known building blocks to implement it. We select a 4:2 compressor as the basic building block, because it could be efficiently implemented on Xilinx FPGAs. The implement-tation of a generic CS compressor tree requires [Nop/2]-1, 4:2 compressors (because each one eliminates two signals), whereas a carry-propagate tree uses. 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. To optimize the use of the carry resources, we propose a compressor tree structure similar to the classic linear array of CSAs.

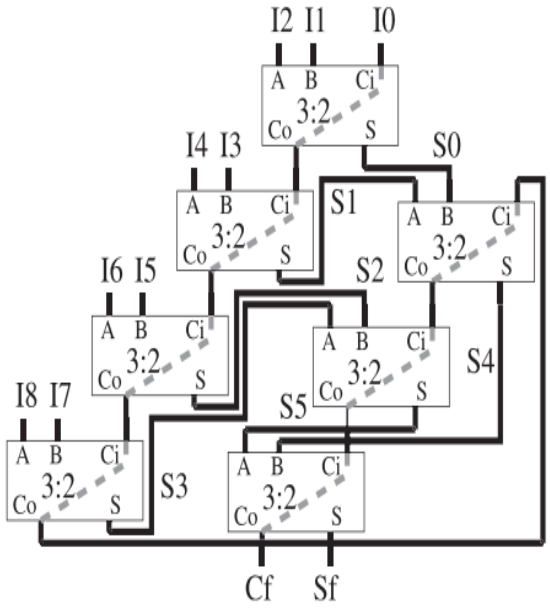

Fig.1.

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 (Co) 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 C_f). First, the two regular inputs on each CSA are used to add all the input operands (Ii). When all the input operands have been introduced in the array, the partial sum-words (S_i) previously generated are then added in order (i.e., the first generated partial sums are added first).

V. IMPLEMENTATION RESULTS AND COMPARISON

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 [46], because this adder is not automatically supported by ISE 9.2. Furthermore, to investigate their portability, compressor trees based onnternary CPAs were also synthesized to target the Altera Stratix-II family. In this case, the ternary adders are directlyinstantiated at a high level. We now summarize the main results obtained in this study.

A. Results on FPGA Families with Support for Binary CPAs

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. 8 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. 8), which could be eliminated by manually designing low-level CPAs or by changing the synthesis tool. On the other hand, this faster CPA implementation leads to more significant speedups for these cases, as shown in the following.

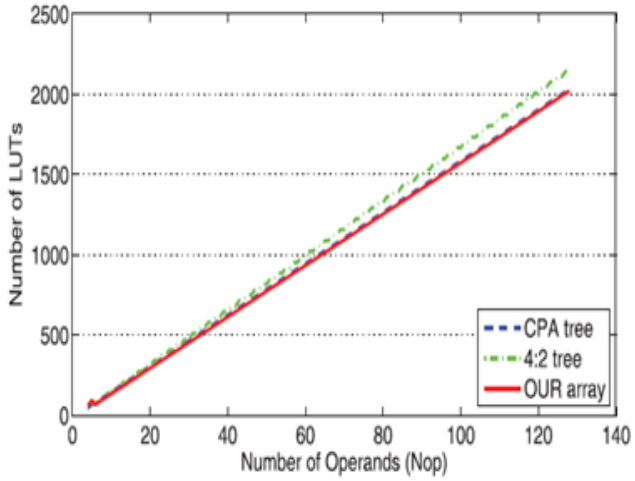

Fig.2.

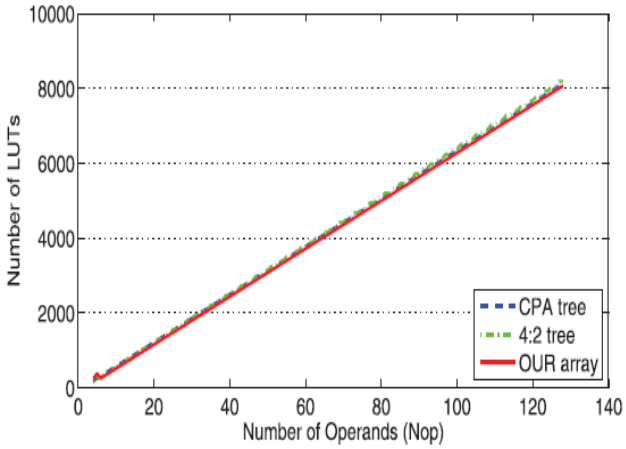

Fig.3.

Regarding speed, Fig. 9 shows the speedup achieved when using OUR array instead of CPA tree, for different numbers of operands and varying the number of bits from16- 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 excluding4- 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 speedup due to the small CPAs required. The speedup achieved is very dependent on the number of operands Nop. On the one hand, the growth of the speedup

in relation to N (i.e., the slope) increases when Nop decreases. On the other hand, the starting point of the speedup line generally decreases when Nop increases, although it presents a strong increase for certain values of Nop. For this reason, and due to compressing the maximum amount of information while keeping the image simple, we have used values of Nop, which can be used as bounds of monotonic intervals. In this way, for values of Nop between five and eight operands, the line of the speedup is between these bound lines; for values of Nop between 9 and16 operands, the line is between these bound lines, and soon.

In addition, inside these ranges, and for a fixed number of bits, the speedup always decreases when the number of operands increases. Fig. 10 represents the delay changes in relation Nop from 4 to 128 operands for N equal to 16 and 64 bits. For all cases, OUR array is the fastest solution, whereas CPA tree is faster than 4:2 tree when N is 16 (except for Nop lower than 9); we obtain the opposite result when N is 64 bits. The graphs corresponding to CPA tree and 4:2 tree are arranged in steps. They present an almost imperceptible growth in delay when Nop increases, but a very high growth occurs when Nop is a power of two. This behavior is due to the introduction of a new level of adders at these points. Nevertheless, the behavior of OUR array is smoother because the delay of the proposed structure depends on both the number of levels and the number of adders on each level (see Section 3.2). This figure makes clear the behavior shown in Fig. 9. Similar behavior in relation to Nop is observed when comparing OUR array with 4:2 tree. However, in this case, the speedup achieved is practically independent of the number of bits N.

We should also note that when the bit width N is very low compared to Nop, the maximum delay of OUR array is limited due to the effect of the ending CPAs, as stated in Section 3.3. For this reason, in Fig. 10, the delay of OUR array for more than 64 operands is smaller for 16-bit width see Fig. 10a) than for 64-bit width (see Fig. 10b). Table 1 presents the bounds and the average of the speedups obtained by using OUR array or 4:2 tree instead of CPA tree, for different ranges of Nop. The OUR array approach is clearly superior to the classic 4:2 tree, and it could achieve strongly reduced delays, even for short bit widths.

VI. CONCLUSION

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. We have compared our architectures, implemented on different FPGA families, to several designs and have provided a qualitative and quantitative study of the benefits of our proposals.

VII. REFERENCES

[1] B. Cope, P. Cheung, W. Luk, and L. Howes, ―Performance Comparison of Graphics Processors to Reconfigurable Logic: A Case Study,‖IEEE Trans. Computers, vol. 59, no. 4, pp. 433-448, Apr. 2010. [2] S. Dikmese, A. Kavak, K. Kucuk, S. Sahin, A. Tangel, and H. Dincer, ―Digital Signal Processor against Field Programmable Gate Array Implementations of Space-Code Correlator Beamformer for Smart Antennas, ‖IET Microwaves, Antennas Propagation, vol. 4, no. 5,pp. 593-599, May 2010. [3] S. Roy and P. Banerjee, ―An Algorithm for Trading off Quantization Error with Hardware Resources for MATLAB-based FPGA Design, ‖IEEE Trans. Computers, vol. 54, no. 7, pp. 886-896, July 2005. [4] F. Schneider, A. Agarwal, Y.M. Yoo, T. Fukuoka, and Y. Kim,―A Fully Programmable Computing Architecture for Medical Ultrasound Machines, ‖IEEE Trans. Information Technology in Biomedicine, vol. 14, no. 2, pp. 538-540, Mar. 2010. [5] J. Hill, ―The Soft-Core Discrete-Time Signal Processor Peripheral[Applications Corner],‖ IEEE Signal Processing Magazine, vol. 26,no. 2, pp. 112-115, Mar. 2009. [6] J.S. Kim, L. Deng, P. Mangalagiri, K. Irick, K. Sobti, M. Kandemir,V. Narayanan, C. Chakrabarti, N. Pitsianis, and X. Sun, ―An Automated Framework for Accelerating Numerical Algorithmson Reconfigurable Platforms Using Algorithmic/Architectural Optimization, ‖IEEE Trans. Computers, vol. 58, no. 12, pp. 1654-1667, Dec. 2009. [7] H. Lange and A. Koch, ―Architectures and Execution Models for Hardware/Software Compilation and their System-LevelRealization, ‖IEEE Trans. Computers, vol. 59, no. 10, pp. 1363-1377, Oct. 2010. [8] L. Zhuo and V. Prasanna, ―High-Performance Designs for Linear Algebra Operations on Reconfigurable Hardware, ‖IEEETrans. Computers,vol. 57, no. 8, pp. 1057-1071, Aug. 2008.