www.ijsetr.com

International Journal of Scientific Engineering and Technology Research

ISSN 2319-8885
Vol.03,Issue.47
December-2014,
Pages:9462-9466

# Multioperand Redundant Adders on FPGAs

MUDASIR MD[1], TULASI SANATH KUMAR[2]
[1]PG Scholar, Dept of ECE, ASCET, Gudur, AP, India, Email: mudasir.md786@gmail.com.
[2]Assistant Professor, Dept of ECE, ASCET, Gudur, AP, India, Email: tulasisanath@gmail.com.

**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, 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. Thus, these kinds of devices are increasingly selected as the target technology for many applications, especially in digital signal processing hardware accelerators cryptography 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.

In the authors demonstrate that the intensive use of these new elements reduces the performance GAP between FPGA and ASIC implementations. One of these resources is the carry-chain system, which is used to improve the implementation of carry propagate adders (CPAs). It mainly consists of additional specialized logic to deal with the carry signals, and specific fast routing lines between consecutive LEs, as shown in Fig.1. This resource is presented in most current FPGA devices from low-cost ones to high-end families, and it accelerates the carry propagation by more than one order of magnitude compared to its implementation using general resources. Apart from the CPA implementation, many studies have demonstrated the importance of using this resource to achieve designs with better performance and/or less area requirements, and even for implementing non arithmetic circuits. Multioperand 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 CS 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. However, due to the efficient implementation of CPAs, the use of redundant adders has usually been rejected when targeting FPGA technology. A direct implementation of a 3:2 counter usually doubles the area requirements of its equivalent CPA and improved speed is only noticeable for long bit widths. Nevertheless, several recent studies have demonstrated that redundant adders can be efficiently mapped on FPGA structures, reducing area overhead and improving speed, as described in Section 2. Despite the important advances represented by these previous studies, the solutions proposed require either (or sometimes both) the use of a sophisticated heuristic to generate each compressor tree or a low-level design. The latter impedes portability, because it is highly dependent on the inner structure. In addition, their area and speed could be improved, because the use of a specialized fast carry-chain is very limited.

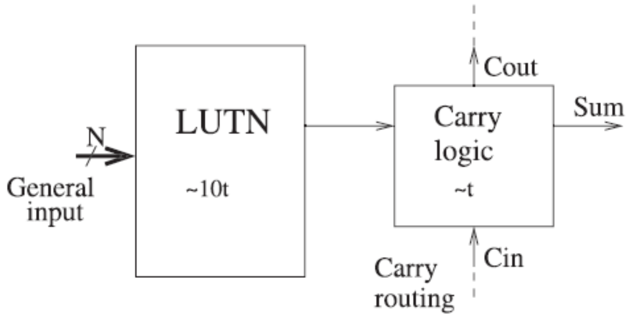

**Fig.1. General Scheme of dedicated carry-chain resources included in modern FPGA devices.**

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. The rest of the paper focuses on CS representation, because the extension to SD representation could be simply achieved by inverting certain input and output signals from and to the compressor tree, as was demonstrated. Since it is unnecessary to make any internal changes to the array structure, these small modifications do not significantly modify compressor tree performance. The remainder of this paper is organized as follows: SectionII Carry Save Adders on FPGA. In SectionIII, we present Efficient Mapping of Carry - Save Adder in FPGA. In SectionIV, we compare the results of implementation using different approaches. Finally, the conclusions are presented in SectionV.

## II. CARRY SAVE ADDERS ON FPGA

This paper focuses mainly on the inner architecture of FPGAs with specialized carry-logic like Virtex 2, 4and Spartan 2, 3 of Xilinx and 4-input Look up tables. In spite of new generation Field programmable gate arrays which are having new inner architecture, FPGAs with four-input LUTs are widely used for medium complex applications due to low cost and low power consumption. It describes architecture of a slice implementing a CPA. Each slice includes two four-input Look up tables, two flip-flops, the specialized carry-logic and the necessary logic and multiplexers. These elements are connected as shown in the figure to operate like a CPA: the lower slice generates a carry bit ($c_i$+1) and a sum bit ($s_i$) from three input bits $x_i$, $c_i$. By using the carry propagation logic the carry bit $c_i$+1 is then passed to the upper slice, where it will be added with $x_i$+1 and $y_i$+1, generating the next sum and carry bits,$s_i$+1 and $c_i$+2. Thus, each slice allocates the full addition of two pairs of bits. If we use a carry-save adder, $s_i$ and $c_i$+1 should be computed in parallel for all bits comprising the input operands, independently from input and output carries. But this is not possible between the lower and upper parts of the slice. This means that hardware design tools allocate two Look up tables one for sum computation and carry computation. When they are provided with a CSA HDL description, i. e., they assign a full slice to the whole computation of one pair of bits.

In carry save addition (CSA) implementation on FPGA, the carry-out bit and the sum bit are generated using two LUTs whereas a carry propagate addition (CPA) we need only one LUT. Thus, the hardware required for a Carry save adder is double than that for a CPA. Besides, the CSA implementation does not take advantage of the carry propagation logic. In an attempt to use the available carry-logic while keeping an adder maximum delay bounded regardless of the word length, authors present a solution making use of a high radix carry-save representation. Due to this high radix representation, initially introduced to reduce the number of wires and registers required to store a value, the sum word from a carry-save number is represented in radix- r (i. e. log2r bits per digit) and the carry word requires one only bit per radix- r digit. This representation allows the use of standard CPAs to add each of the sum word radix- r digit, connecting the carry word to the Carry propagate adder carry-in inputs, hence obtaining the final carry word at the CPA carry-out outputs. When this adder is implemented in an FPGA, we use the whole slice resources, including the carry logic, while increasing the addition delay. However, due to the great optimization of FPGAs carry logic, this delay increase is not very significative if the radix r is not high.

The main drawback in high radix carry save representation is that, the numbers shifts are not an easy task. In this case, complete shifts are only available for radix- r digits, i. e., shifts are only allowed for multiple of r numbers. This restriction comes from the carry word processing, since it is only available at some specific positions within the addition operation. These limitations becomes an important obstacle when applying the high radix carry save representation to many shifts and add based algorithms, and even the work presented has to deal with this problem. For this reason, it is interesting to look for some other ways of using the carry logic when implementing carry save adders.

## III. EFFICIENT MAPPING OF CARRY - SAVE ADDER IN FPGA

Two different solutions to obtain a more efficient implementation of carry-save adders on FPGAs than the one presented in this section.

### A. Using Half of a Slice for a 3:2 Counter

The first proposed solution makes use of only half of a slice for a 1-bit 3:2 carry-save adder implementation. However, the remaining half of slice cannot be fully used, since the carry bit produced by 3:2 counter computations is feeded into it, disabling a possible use for the rest of the carry propagation logic. In this solution it is not possible to implement two 1-bit 3:2 CSAs within a single FPGA slice. Nevertheless, the free semi-slice resources can still be used by some other type of logic computation which does not need to take advantage of the carry logic. Fig.2 depicts how this solution is mapped into a slice.

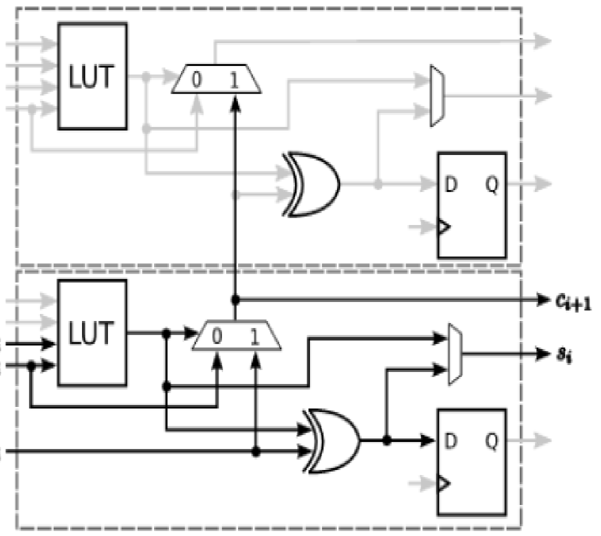

**Fig.2. Efficient slice mapping for a 1-bit 3:2 CSA implementation.**

The main drawback in this case is that the upper semi-slice (the one left free) often remains unused within their application. As a consequence, the area requirements for this approach are higher than the one obtained by the solution described by them. Some other example applications, such as constant multiplier and an additive range reduction are developed. Where we have successfully taken advantage of the upper semi-slice using it as a table look-up. From the results obtained, we can conclude that this solution is convenient for those applications where the upper semi-slice can be used.

### B. Implementing A 4:2 Compressor

To overcome the drawback we cannot always guarantee a successful use of the upper semi-slice, for example for the commonly used multi operand addition. For this reason, here we propose a new type of mapping where we fully use whole slice hardware resources. The new approach lies in a 4:2 compressor implementation instead of a single 3:2 counter. Fig.3 depicts a typical 4:2 compressor scheme based on 3:2 counters, and Fig.4 shows how this 4:2 compressor can be efficiently mapped into an FPGA slice. In order to achieve this goal, we have to map some parts from the addition of different weighted bits within the same slice. Specifically, the piece of hardware highlighted in Fig.3 is implemented into single slice.

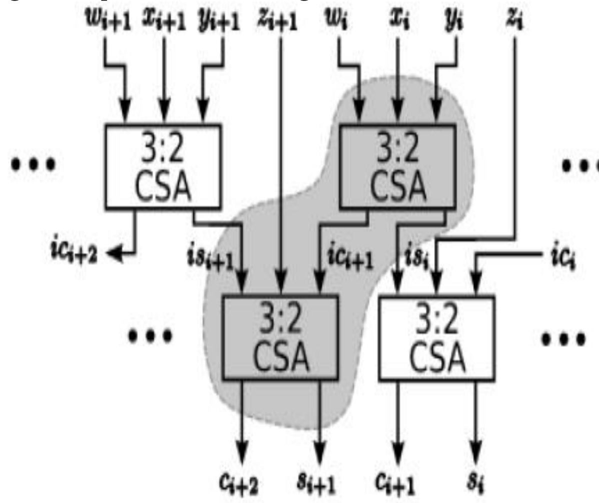

**Fig.3. 4:2 compressor implementation using 3:2 counters.**

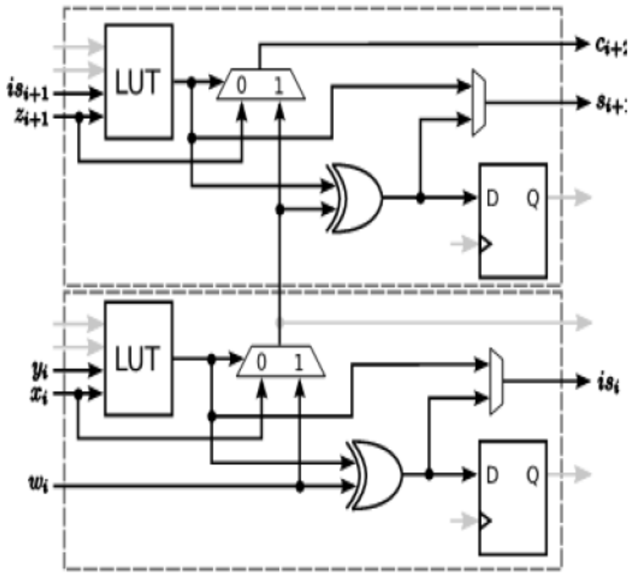

**Fig.4. Mapping of 4:2 compressors into a slice.**

The upper semi-slice implements a second level 3:2 CSA, whereas the bottom semi-slice is in charge of implementing a first level 3:2 CSA In order to take advantage of the carry propagation logic, a single slice implements the first level addition for bits with weight 2^i and the second level addition for bits with weight 2^i+1. In this way, all the slice resources are used.

## IV. RESULTS PROPOSED METHOD

Results of this paper is shown in bellow Figs. 5 to 12.

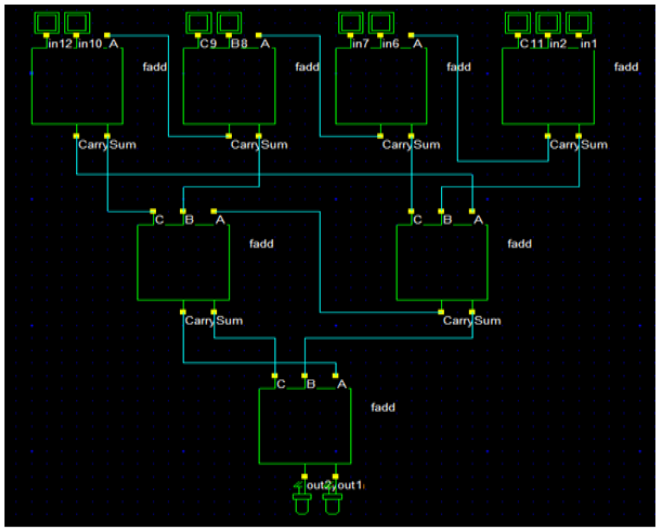

Fig.5.Proposed adder design based on compressor Simulation.

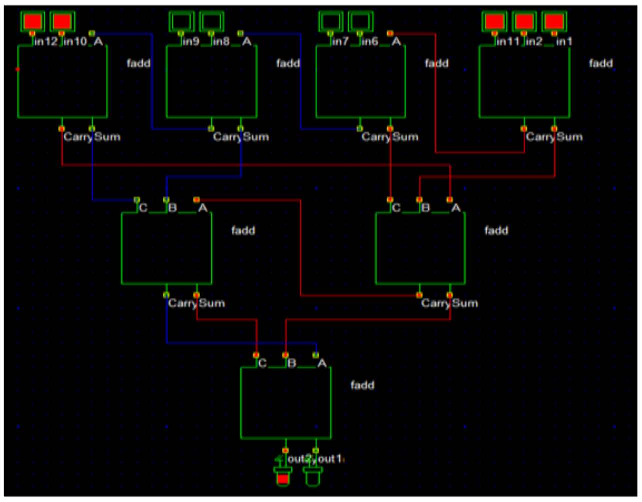

Fig.6. Compressor simulation based on the linear model approach Waveform.

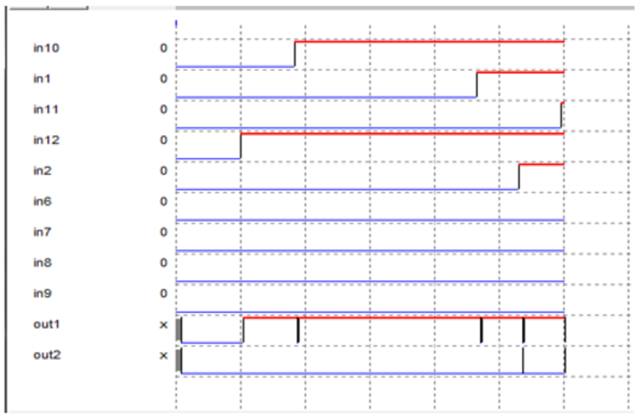

Fig.7. The outputs based on the inputs as sum is out1 and out2.

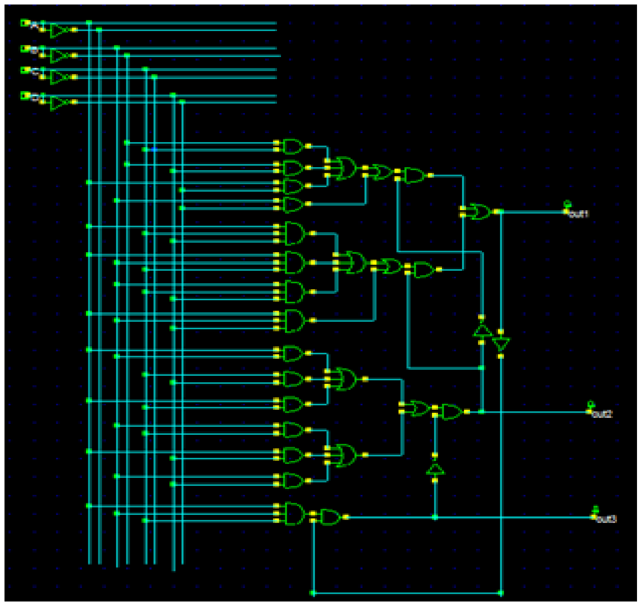

Fig.8. Compressor based adder.

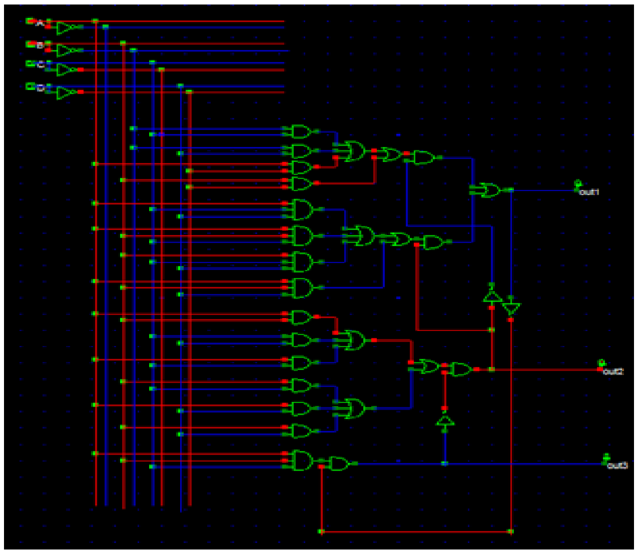

Fig.9.Simulation.

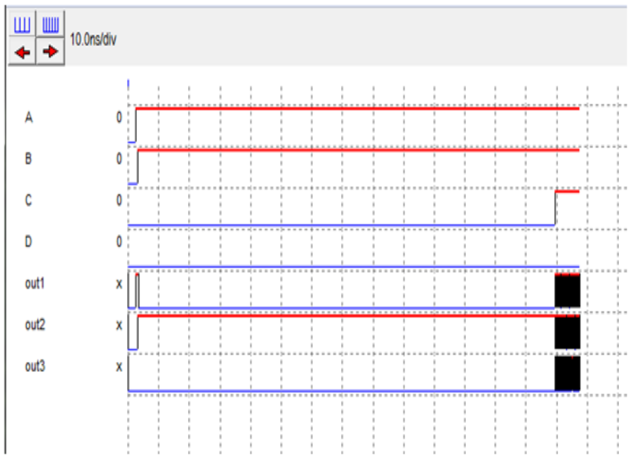

Fig.10. Waveform.

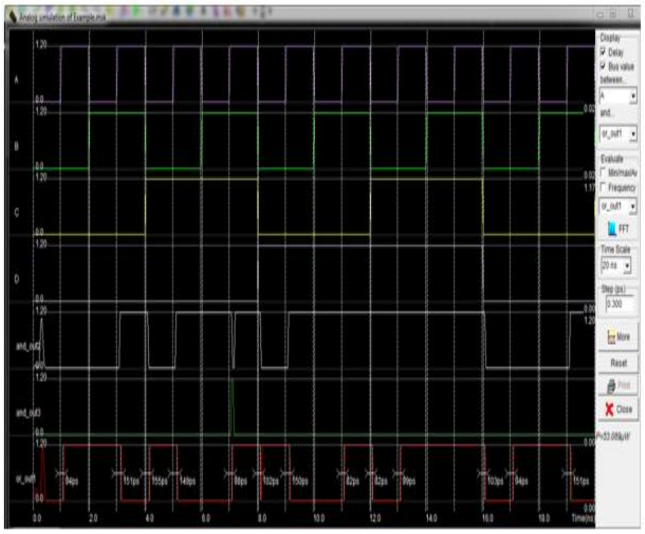

Fig.11. Proposed power result.

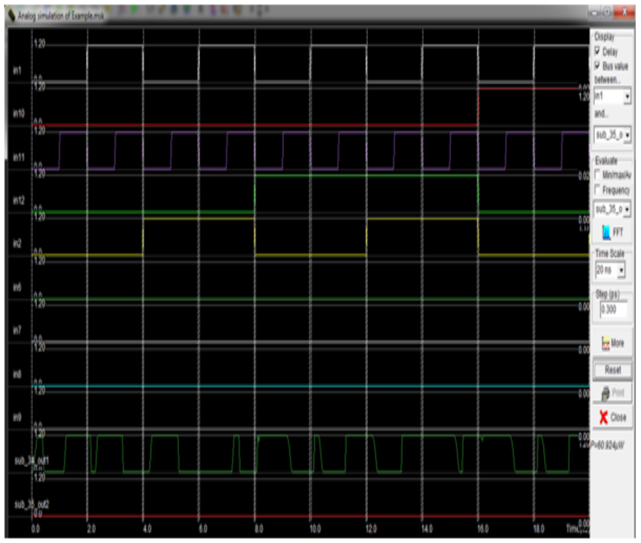

Fig.12. Cyclic adder result.

## V. CONCLUSION AND FUTURE SCOPE

### A. Conclusion

It was shown in this work that better compressing elements can be found by evaluating the low level structure of the FPGA. Novel compressing elements for modern Xilinx devices were proposed including different GPCs and a 4:2 compressor based on a ternary adder. Better efficiency, a lower delay or both compared to previous compressing elements. They can be pipelined without overhead using the otherwise unused flip-flops in the device. A design example of a pipelined compressor tree showed the effectiveness of the 4:2 compressors.

### B. Future Scope

Further work has to be done in the automatic synthesis of pipelined compressor trees. Previous work only focused on non-pipelined compressor trees although this quite limits the speed of the design. However, the strategy for pipelining is different as each input bit has to be covered by at least a single flip-flop for each compression stage to get a balanced pipeline. Another issue is the selection of the best compressing element.

## VI. REFERENCES

[1] Javier Hormigo, Julio Villalba, Member, IEEE, and Emilio L. Zapata, "Multioperand Redundant Adders on FPGAS", IEEE Transactions on Computers, Vol. 62, No. 10, October 2013.

[2] 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.

[3] 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 Beam former for Smart Antennas," IET Microwaves, Antennas Propagation, vol. 4, no. 5, pp. 593-599, May 2010.

[4] 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.

[5] 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.

[6] 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.

[7] 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 Algorithms on Reconfigurable Platforms Using Algorithmic/Architectural Optimization," IEEE Trans. Computers, vol. 58, no. 12, pp. 1654-1667, Dec. 2009.

[8] H. Lange and A. Koch, "Architectures and Execution Models for Hardware/Software Compilation and their System-Level Realization," IEEE Trans. Computers, vol. 59, no. 10, pp. 1363-1377, Oct. 2010.

[9] L. Zhuo and V. Prasanna, "High-Performance Designs for Linear Algebra Operations on Reconfigurable Hardware," IEEE Trans. Computers, vol. 57, no. 8, pp. 1057-1071, Aug. 2008.

[10] C. Mancillas-Lopez, D. Chakraborty, and F.R. Henriquez, "Reconfigurable Hardware Implementations of Tweak able Enciphering Schemes," IEEE Trans. Computers,, vol. 59, no. 11, pp. 1547-1561, Nov. 2010.

[11] T. Guneysu, T. Kasper, M. Novotny, C. Paar, and A. Rupp, "Cryptanalysis with COPACOBANA," IEEE Trans. Computers, vol. 57, no. 11, pp. 1498-1513, Nov. 2008.

[12] I. Kuon and J. Rose, "Measuring the Gap between FPGAs and ASICs," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. 26, no. 2, pp. 203-215, Feb. 2007.