International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-1 Issue-6, August 2012

Implementation of carry-save adders in FPGA

S. Ravi Chandra Kishore, K.V. Ramana Rao

Abstract*—The addition operations can be optimized through a special purpose carry propagation logic in most of the FPGAs. The delay is same for small size operands and this redundant adders require more hardware resources than carry propagate adders. Therefore, carry-save adders are not usually implemented on FPGA devices, although they are very useful in ASIC implementations. In this paper we have showed that it is possible to implement redundant adders with a hardware cost close to that of a carry propagate adder. Redundant adders are clearly faster for 16 bits and bigger word lengths and have an area requirement similar to carry propagate adders. Among all the redundant adders studied, the 4:2 compressor is the fastest one, presents the best exploitation of the logic resources within FPGA slices and the easiest way to adapt classical algorithms to efficiently fit FPGA resources. This design aimed to be implemented in Spartan-3E FPGA. The CSA architecture uses 1215 LUT's out of available 3840 and 96 IO blocks and the average fan-out of non clock nets is 4.73 and the peak memory usage is 148 MB.

***Index Terms*—ASIC, redundant adders, FPGA.**

I. INTRODUCTION

In despite of specific purpose ASIC designs always show a better performance regarding to area and time than FPGA designs, the use of FPGA devices has extended in the hardware design community in the last years due to its use facility, flexibility, liability, low cost and short development time. An FPGA device has a special inner structure that tries to cover the most general case of designs. Basically, an FPGA is structured as a grid of small elements which are able to implement basic logic operations and store resources, together with routes for interconnecting these elements . Besides, some other hardware resources have been recently added in order to accelerate some specific operations. However, most current hardware-oriented algorithms are intended to be implemented on ASIC-based chips, thus they do not take into account FPGAs especial configuration. Because of this, a big amount of hardware resources from within FPGAs are wasted in many designs. Due to this fact, recently there is an increasing interest of the scientific community to design new algorithms which take advantage of the special FPGA inner architecture [1]. The most usual operations in any design is addition. The architecture of most of the modern FPGA devices use to have a special hardware in charge of dealing with addition, which is mainly focused on improving the performance of carry propagate adders (CPA). More specifically, the path for carry propagation has been specially optimized so that it goes from one basic element to the next one using a specific fast route, together with some specific carry-logic to add and propagate the carry value. Because of this reason, carry propagated adders are preferred than carry-save adders (CSA) for implementation on FPGA devices, since, for non very long word lengths, when compared with CPAs, CSAs have similar delays, but double the number of logic resources [2]. Nevertheless, we think that this is due to the fact that the software tool does not efficiently manage the system resources when mapping the carry-save adders into an FPGA Platform.

In this paper we prove that there is possibility to implement carry-save adders on FPGA devices with a similar hardware cost to that of carry-propagate adders, while keeping a constant computation time, in such a way that considering operands with number of bits greater or equal to 16, the speed gain is notorious, this process is similar to an ASIC-based design.

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.

Fig. 1 describes an 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 (ci+1) and a sum bit (si) from three input bits xi, , ci.

By using the carry propagation logic the carry bit ci+1 is then passed to the upper slice, where it will be added with xi+1 and yi+1, generating the next sum and carry bits,si+1 and ci+2. Thus, each slice allocates the full addition of two pairs ofbits. If we use a carry-save adder, si and ci+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, as we can see in Fig. 1. 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.

Manuscript published on 30 August 2012.
***** Correspondence Author (s)
S. Ravi Chandra Kishore*, M.Tech ECE Department,JNTU Kakinada University/ Pydah College of Engineering and Tehnology/Visakhapatnam, India.
K.V. Ramana Rao, Assoc.Professor & Head, Dept. of ECE, Pydah College of Engineering & Technology, India.

In an attempt to use the available carry-logic while keeping an adder maximum delay bounded regardless of the wordlength, authors from [1] 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, as shown in [6].

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. This limitation becomes an important obstacle when applying the high radix carry save representation to many shift and add based algorithms, and even the work presented in [1] 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.

II. 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 [1] are shown 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 computation 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.

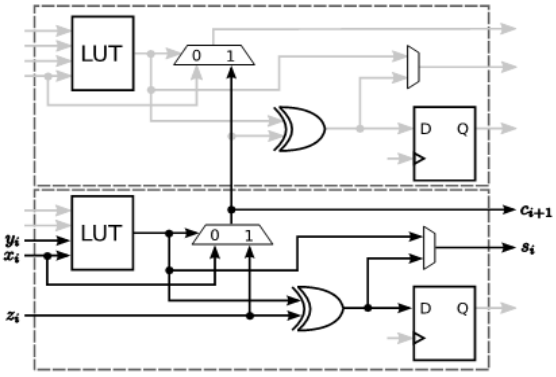

Fig. 2. Efficient slice mapping for a 1-bit 3:2 CSA implementation

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 is higher than the one obtained by the solution described by them. Some other example applications, such as a 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 shown in Section III-A, i. e., 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 a 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 asingle slice.

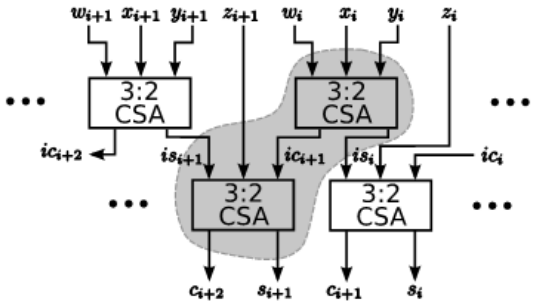

Fig. 3. 4:2 compressor implementation using 3:2 counters

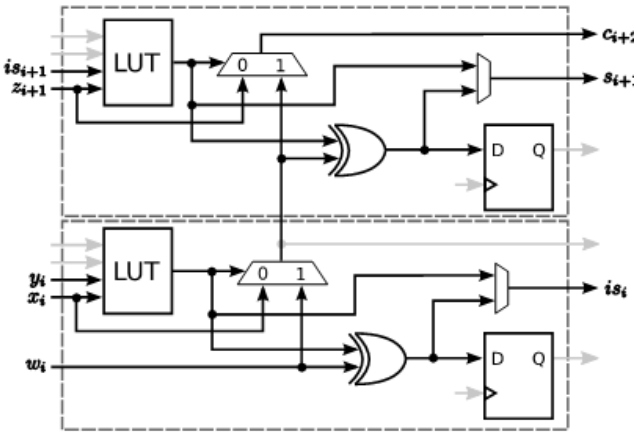

Fig. 4. Mapping of a 4:2 compressor into a slice

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.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

In this way, all the slice resources are used.

III. IMPLEMENTATION RESULTS

This section deals with two kind of matters. First of all,we analyze the circumstances where it is worth a carry save adder implementation instead of a carry propagate one. Then,we compare the proposed CSA implementations described in Section III together with the existing CSA solutions for FPGA

Implementations shown in Section II. This analysis is based on the experimental results obtained from the synthesis and simulation of the given implementations using ModelSim over a Spartan 3 FPGA. Table I shows the deviceutilization summary, the high radix CSA implementation proposed in [1] and the 3:2 counter seen in Section III-A. Theoretically, the delay for redundant adders must be independent of the input parameters word lengths. However, the experimental results for numbers with representations between 2 and 512 bits show different delays, which is due to the circuit routing. For this reason, Table I just shows the boundary results obtained, the intermediate results are not relevant for a comparison. The better performance we get for a CSA implementation if the word length is bigger, as compared to a CPA one. For a word length smaller than 16 bits, the routing influence is more no ticeable and the gain obtained by the CSAs is not beneficial as it is for higher word lengths. For 16 bits and longer numbers the delay increase in the Carry propagate adder implementation is approximately doubled when the amount of bits is also doubled. On the other hand, the delay increase for the Carry save adder implementations is negligible, since it basically comes from the circuit routing. We can also observe that the fastest solution is the 3:2 counter, although the radix-4 CSA has a very close delay obviously, the bigger the radix is, the bigger delay we get. Regarding to the area requirements, the 3:2 counter from Section III-A needs the same amount of slices as a CPA implementation, but just in case each of the free semi-slices from the 3:2 counter are able to be used for a further logic computation (no needing the carry logic). In case we can not use those free semi-slices, the most convenient implementation would be the radix-4 one, since it has a very close delay to the 3:2 counter, but it requires the same area as a CPA. Another remarkable aspect related to this adder, , lies in the number of registers and wires needed to store and transmit any number, for example for a pipelined fashion design. A carry-save representation requires an amount of registers double than the one needed by a non-redundant representation, since a carry-save one needs a carry word. According to this, a high radix CSA requires less registers than a standard radix-2 carry-save, due to a smaller carry word. E.g.: a radix-4 CSA needs a 25% less registers whereas a radix-16 needs a 37.5% less registers. A comparison with the 4:2 compressor requires a different analysis, since it performs an operation slightly different to the one carried out by the previously described adders. The main difference lies in the type of operands accepted by the adders: both, the 3:2 counter and the high radix adder can perform an addition of one redundant number with a not her conventional one, whereas the 4:2 compressor is able to perform an addition with two redundant numbers. For this reason, for a fair comparison, we will compare a 4:2 compressor, not with a single high radix CSA, but with a completed one able to perform a full addition of two high radix numbers (xs , xc) + (ys, yc),let us call it a radix-4 CS compressor. In order to do so, we need to connect two radix-4 CSAs: the first one adds xs, xc and ys , whereas the second one adds the first radix-4 CSA output together with yc . Note that yc have a smaller word length, thus it has to be completed with some extra bits set to 0 alternatively (... yci− 1 0 yci 0 yci+1 ...). If the radix-4 CS compressor is intended to add non-conventional numbers, then there is no need to insert the extra 0-bits, resulting in a different hardware. A comparison between a 3:2 counter and a 4:2 compressor is not worth, since a 4:2 compressor is comprised of two 3:2 counters which optimally use the resources available within a slice (see Figs. 3 and 4).Analogously to what happened with the previous CSAs, the inherent delays for the 4:2 compressor when considering different word lengths, is also due to the routing process, that is why we have also calculated the maximum delays for the4:2 compressor and the equivalent high radix CSA. This design aimed to be implemented in Spartan-3E FPGA. The CSA architecture uses 1215 LUT's out of available 3840 and 96 IO blocks and the average fan-out of non clock nets is 4.73 and the peak memory usage is 148 MB.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	1,215	3,840	31%	
Number of occupied Slices	666	1,920	34%	
Number of Slices containing only related logic	666	666	100%	
Number of Slices containing unrelated logic	0	666	0%	
Total Number of 4 input LUTs	1,215	3,840	31%	
Number of bonded IOBs	96	173	55%	
Average Fanout of Non-Clock Nets	4.73			

IV. CONCLUSION

As a final conclusion, we state that CSA implementations could be easily included into FPGAs hardware design tools. In this case, we could easily take advantage of FPGAs resources when porting applications using redundant arithmetic from an ASIC platform to an FPGA platform.

REFERENCES

[1] J.-L. Beuchat and J.-M. Muller, "Automatic generation of modular mul-tipliers for fpga applications," IEEE Transactions on Computers, vol. 57, no. 12, pp. 1600–1613, December 2008.

[2] J. Detrey, F. de Dinechin, and X. Pujol, "Return of the hardware floating-point elementary function," in Proceedings of the 18th IEEE Symposium on Computer Arithmetic (Montpellier, France), Kornerup and Muller, Eds. Los Alamitos, CA: IEEE Computer Society Press, June 2007, pp. 161–168.

[3] H. Eberle, G. N., S. Shantz, V. G upta, L. Rarick, and S. Sundaram, "A public-key cryptographic processor for RSA and ECC," in Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors (ASAP2004), September 2004.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

[4] H. R. Ismail, R.C., "High performance complex number multiplier using booth-wallace algorithm," in IEEE International Conference on Semiconductor Electronics ICSE, November 2006.
[5] K. Manochehri and S. Pourmozafari, "Modified radix-2 montgomery modular multiplication to make it faster and simpler," in IEEE International Conference on Information Technology: Coding and Computing, ITCC 2005 , April 2005.
[6] M.D.ErcegovacandT.Lang, Digital Arithmetic. Morgan Kaufmann Publishers, 2004.

Mr S.Ravi Chandra Kishore is pursuing M.Tech(VLSI) in the Department of ECE at Pydah College of Engineering & Technology, Visakhapatnam, A.P., India. His research interest includes VLSI Design and computer aided design.

Mr K.V. Ramana Rao working as Assoc. Professor & Head, Department of ECE at Pydah College of Engineering & Technology, Visakhapatnam, A.P., India. His research interest includes Digital Signal Processing and VLSI Design.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)