Robert SMYK[1], Maciej CZYŻAK[1]

Gdańsk University of Technology (1)

doi:10.15199/48.2018.02.39

Implementation of multi-operand addition in FPGA using high-level synthesis

Abstract. *The paper presents the results of high-level synthesis (HLS) of multi-operand adders in FPGA using the Vivado Xilinx environment. The aim was to estimate the hardware amount and latency of adders described in C-code. The main task of the presented experiments was to compare the implementations of the carry-save adder (CSA) type multi-operand adders obtained as the effect of the HLS synthesis and those based on the basic component being 4-operand adder with fast carry-chain available in FPGA's implemented in Verilog. However, the HLS synthesis is simplifies the design and prototyping process but the received results indicate that the circuit obtained as the result of such synthesis requires twice more resources and is slower than its counterpart design using Verilog.*

Streszczenie. *W pracy zaprezentowano rezultaty syntezy wysokopoziomowej sumatorów wielo-operandowych w środowisku Vivado Xilinx. Celem pracy była ocena złożoności sprzętowej i opóźnienia sumatorów uzyskanych poprzez opis w języku C. Głównym zadaniem przeprowadzonych eksperymentów było porównanie implementacji sumatorów zachowujących przeniesienie otrzymanych w wyniku syntezy wysokopoziomowej i tych implementowanych w języku Verilog wykorzystujących łańcuch szybkich przeniesień w FPGA. Uzyskane rezultaty wskazują, że wprawdzie synteza wysokopoziomowa układów jest znacznie prostsza i pozwala na szybsze uzyskanie implementacji, jednak otrzymuje się struktury wymagające dwukrotnie większych zasobów sprzętowych niż to ma miejsce w przypadku użycia języka Verilog. (**Implementacja sumowania wielooperandowego w FPGA przy zastosowaniu syntezy wysokopoziomowej**).*

Keywords: carry-save adders, generalized parallel counters, multi-operand addition, FPGA.
Słowa kluczowe: sumatory zachowujące przeniesienia, uogólnione liczniki równoległe, wielooperandowe sumowanie, FPGA.

Introduction

Multi-operand addition can be implemented in the most direct way by using two-operand carry-propagate adder (CPA) trees. But this approach is generally ineffective due to large area and long delay. Much better results can be obtained using compressor trees. Usually multi-operand addition is realized in two phases where the number of addends is compressed to two using a compressor tree and next the CPA is applied. The two classical forms of compressors are Wallace [1] and Dadda [2] trees. These trees use 3-input 2-output counters being full adders (FA) as carry-save adders (CSA) or 2-input 2-output counters. Such approach became a standard in multipliers constructed by application specific integrated circuit (ASIC) designers. Early work on parallel compressors was also presented by Gajski [3]. In the more general approach n-input m-output generalized parallel counters (GPC) are utilized for synthesis of high-speed compression trees. Their number, form and internal connection allows to roughly determine the area, delay and power consumption. The GPC synthesis was already described by Dormido *et. al.* [4]. The design perspective has changed with the introduction of FPGAs with their architecture containing LUTs, fast carry chains and DSP blocks [5,6,7]. The realization of multi-operand addition based on the tree of two-input ripple-carry adders became more viable but with such an approach available LUTs would be used only marginally. The use of modern 6-input LUTs gives much wider possibilities of mapping GPCs onto LUTs. In the direct approach a simple compressor that compresses six bits to three can be built using three LUTs with the common inputs but in such a case fast carry logic is not utilized. The fundamental work on compression trees in FPGAs was done by Parandeh-Afshar *et. al.* [8-12]. Their works also contain the review of the state-of-the art of compression trees. For the design of GPCs they used LUTs and short carry chains with 2 to 3 FAs in series. Kumm and Zipf [13,14] have examined initially GPCs from [15,16]. They have also implemented GPCs from [8] but with improved mappings and have proposed new GPCs. In order to verify their results they synthesized for Xilinx Virtex 6 a compression tree with eight 10-bit inputs. It turned out that best efficiency was attained when 4:2 compressors and ternary adders were used. The ternary adder based solution required 49 LUTs and had a delay of 2.03ns whereas 4:2 compressor tree called for 65 LUTs with the delay of 1.52 ns. They compared their results with those obtained by using FloPoCo [16]. In this case in dependence upon design requirement they obtained one solution with 70 LUTs and the delay of 2.30ns and the other 145 LUTs with the delay of 1.08ns. Further work was done by Khurshid and Mir [17] which proposed a heuristic that mapped GPCs onto minimum possible LUTs number by using the dual output in Xilinx FPGAs. Their GPCs in certain cases had the delays smaller by 10-15% than these from [8,13,14]. Also an original approach was proposed by Matsunaga [18-20], where mapping of GPCs was formulated as the integer linear programming (ILP) problem with speed and power as optimisation goals. This approach resulted in 28% reduction when compared to [9]. The reduction of GPC count leads to the reduction of compression tree stages thereby the delay and power consumption are reduced.

Principles of high level synthesis

The above syntheses can be performed using behavioral or structural descriptions with hardware description languages such as Verilog or VHDL. In general, the design of complex digital systems using these languages is laborious and difficult. In order to make the design process faster and more effective the design method has been introduced that makes uses of C/C++ languages [21,22]. The digital system description in this case has the form of a program in C/C++. However, the program must have the form that could be translatable into the register transfer level (RTL) form. For this reason only a subset of type and grammar constructs are allowed. As the data types in C/C++, the primitive types as unsigned char, unsigned short int, unsigned int together with their signed counterparts, and real types float and double can be applied. The important feature of C/C++ version used in the HLS is the availability of arbitrary precision integer and floating-point types. Also the composite types as array, struct and also statically determinable pointers can be used. For C++ version classes and templates are acceptable.

This approach is termed the High Level Synthesis (HLS). The digital system can be captured, simulated and synthesized to the HDL (Hardware Description Language) format and further to an FPGA implementation. The HLS takes the complete system description and series of directives that control the form of the implementation. The HLS has many advantages as compared to VHDL or Verilog. The principal advantage pertains to the shortening of the time needed for system description and elaboration of test-benches and faster testing. Other advantages result from the fact the elaborated system description has the form of a program. As such it can be modified in many ways in order to obtain the description that would give after the synthesis the FPGA subsystems with the needed properties. These properties pertain mainly to the occupied area, the latency or pipelining rate. We have only single source program and we can have multiple implementations by inserting compilation directives that control the form of the targeted implementation. From the program point of view it might denote the loop unrolling, loop pipelining (a new iteration starts before the previous iteration is complete), array partitioning or pipelining at the register level. Array partitioning allows for various memory distributions between RAMs and LUTs.

Basics of multi-operand addition

The basic case of multi-operand addition is three operand addition. Such adder is termed sometimes the ternary adder. There are two basic forms of such addition. The first can be based on the tree of two binary adders as shown in Fig. 1.

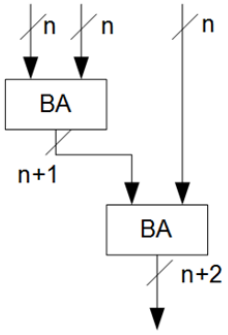

Fig.1. Three operand (ternary) adder

The delay when two independent adders are used eg. Brent-Kung [23], is equal to $t^{(3)}_{sum(n)} = t_{BA(n)} + t_{BA(n+1)}$. In this case we obtain the sum as one binary vector. In case of the greater number of operands such solution seems to be ineffective because of its delay. Alternatively we can use the CSA structure as given in Fig. 2. When summing more than three operands we can use the appropriate number of such structures in the tree form and postpone the final two-operand addition to the final stage.

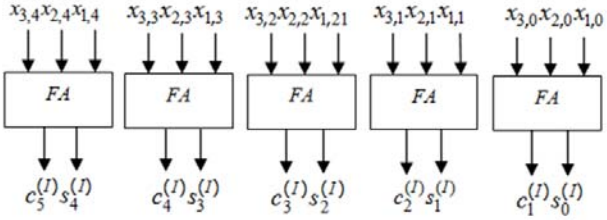

Fig.2. 5-bit CSA adder

At the output we obtain

(1) $$S^{(I)} \leftrightarrow S^{(I)} = (0, s_4^{(I)}, s_3^{(I)}, s_2^{(I)}, s_1^{(I)}, s_0^{(I)})$$

(2) $$C^{(I)} \leftrightarrow C^{(I)} = (c_5^{(I)}, c_4^{(I)}, c_3^{(I)}, c_2^{(I)}, c_1^{(I)}, 0)$$

In this case the delay is equal to t_{FA}. For construction of larger multi-operand adders the more effective can be four-operand CSA tree adder shown in Fig.3. This component allows to construct a regular structure.

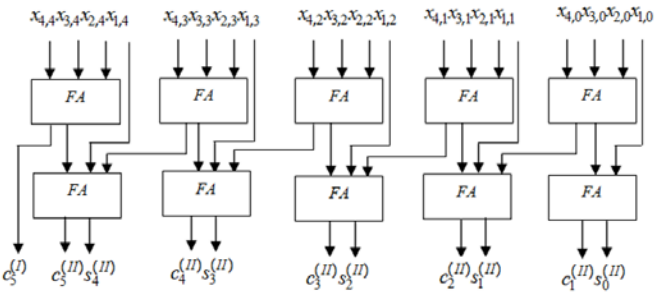

Fig.3. 5-bit 4-operand CSA adder

As an example we consider the use of this component for synthesis of 8-operand n-bit adder. Such structure is shown in Fig.4.

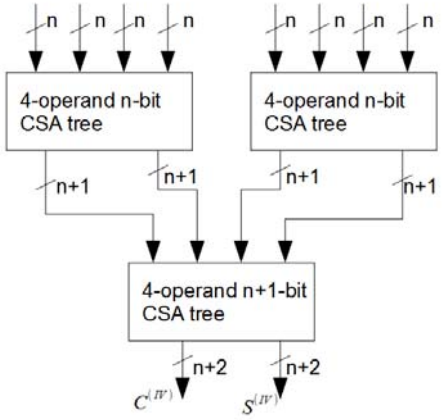

Fig.4. 8-operand CSA tree based on 4-bit adder CSA trees

At the output of the 8-operand CSA tree we obtain two vectors

(3) $$S^{(IV)} \leftrightarrow S^{(IV)} = (c_5^{(III)}, s_5^{(IV)}, s_4^{(IV)}, s_3^{(IV)}, s_2^{(IV)}, s_1^{(IV)}, s_0^{(IV)})$$

(4) $$C^{(IV)} \leftrightarrow C^{(IV)} = (c_6^{(IV)}, c_5^{(IV)}, c_4^{(IV)}, c_3^{(IV)}, c_2^{(IV)} c_1^{(IV)}, 0)$$

This form (Fig. 1-3) or Dadda trees can be used for multi-operand addition in ASICs. The implementation of such structure in FPGAs does not make use of its specific features when we directly map FAs onto LUTs. The more effective way is to use basic components in Configurable Logic Blocks (CLB) as LUTs and specialized fast carry chains included within FPGAs to accelerate carry generation. The communication between LUT slices is organized using the crossbar system that introduces certain delay. In order to shorten the delay the specific vertical carry propagation in columns of LUTs is used. This allows to avoid the FPGA general communication system by crossbars. The direct mapping of the structure from Fig. 2 would require the simulation of individual FAs using LUTs. That would be ineffective because the existing carry chains in the FPGA structure would not be exploited. Their utilization calls for the modification of the structure to the form given in Fig. 5. In such a case instead of simulating directly the second layer of FAs, we can use the fast carry chain (Fig. 6-7).

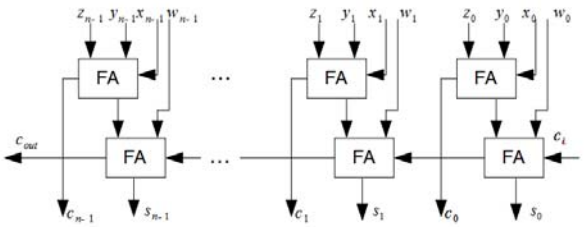

Fig.5. 4-operand n-bit CSA type adder modified for fast carry chain use

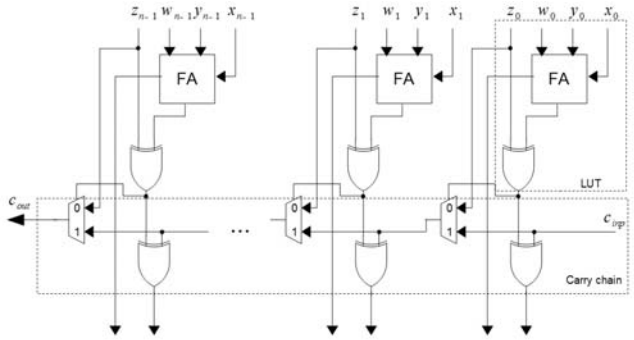

Fig.6. Xilinx FPGA realization of the 4-operand n-bit CSA type adder (based on Fig. 5)

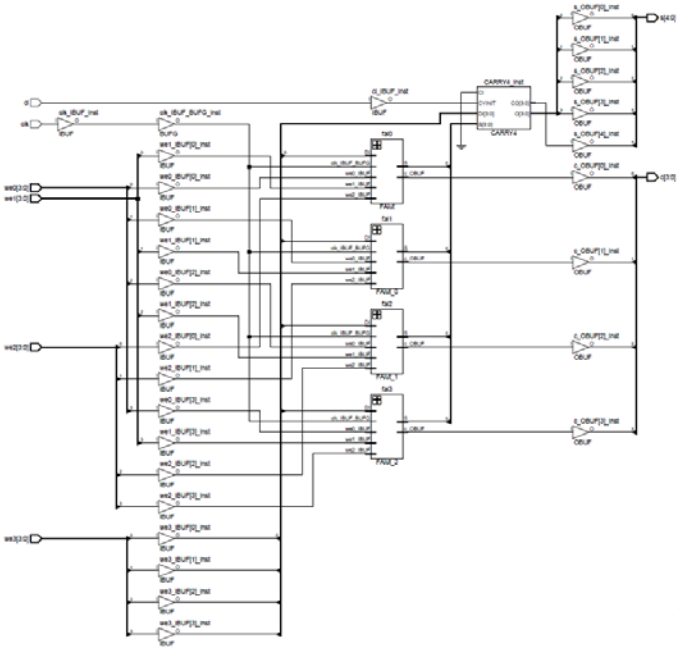

Fig.7. FPGA post-synthesis implementation of 4:2 compressor with the fast carry-chain

HLS synthesis of 4-operand adders

The aim of the presented experiments was to check the influence of the form of the adder description in C/C++ code in the HLS environment on the resulting adder architecture and to estimate hardware amount and latency of four-operand adders. Also the goal was to examine the structure of the circuit obtained as a result of the synthesis. These syntheses have been performed in order to get to know the form of the synthesized adders. It pertains before all to whether carry chains are used or other components. For this purpose two adder configurations have been implemented: one level 4:1 adder, two-level adder 4:2:1 adder. The HLS description of these adders is given in Fig. 8.

```
//#define SUM_2_2
#define SUM_4


void sum_4xN(csa_in a,
             csa_in b,
             csa_in c,
             csa_in d,
             ap_uint<N+3> &wyj)
{
#pragma HLS PIPELINE

#ifdef SUM_2_2
    ap_uint<N+3> wt1, wt2;
    wt1 = a + b;
    wt2 = c + d;
    wyj = wt1 + wt2;
#endif

#ifdef SUM_4
    wyj = a + b + c + d;
#endif
}
```

Fig.8. HLS code for 4:1 and 4:2:1 adder configuration

It was stated that if we exclude the use of DSP48 blocks only LUTs are used without fast carry chains.

The main task of the presented experiments is to compare the implementations of the CSA type multi-operand adders obtained as the effect of the HLS synthesis and those based on the basic component being 4-operand adder that uses the fast carry-chain available in FPGA's. The CSA from Fig. 3 has been implemented algorithmically by generating the connection network as the loop in the HLS code of the circuit. The HLS C-Code of the 4:2 CSA has about 45 lines. The basic components used for Verilog and HLS synthesis were different. The block used for Verilog synthesis makes use of the fast carry chain within the basic four-operand adder. In case of HLS synthesis the direct mapping of this structure into the program code would make it necessary to simulate the carry chain at the software level. But it is known as it was stated that HLS synthesis does not make use of this carry chain in this case. Thus such approach would introduce hardware overhead due to the utilization of additional LUTs to simulate the carry chain.

The results of the HLS synthesis performed for target device 6vlx240tff1156 are given in Table 1. It is worth to remark that before the HLS synthesis we have to assume the maximum acceptable delay as a imposed project datum that must be preserved in the design generated by the HLS synthesizer. It is seen that the latency in both cases is smaller than the assumed clock cycle equal to $t_{clk} = 10ns$. 4:2 CSA requires the largest hardware amount and has the greatest latency. 4:1 adder calls for the smallest hardware amount and 2:2 adder has the smallest latency. This leads to the conclusion that probably in the case of four-operand adders the automatic HLS generation of adder structures may give better results than the algorithmic simulation of the CSA at the bit level.

Table 1. HLS synthesis results for 4-operand 5-bit adders

Adder configuration	LUTs	Input data latency [clk]	Clk timing [ns]
4:1 4-operand	48	1	6,72
2:2 4-operand	61	1	5,78

In the Table 2 and in Table 3 the synthesis results of 4-operand CSA type adder trees with wordlengths n=4,8,16,32 bits are shown. The ratio of the used LUTs in both cases falls off when the adder length increases. Thus the HLS synthesis may be more profitable for long adders. As mentioned above the process of the HLS synthesis

requires the imposition of the maximum delay and this imposed value influences the synthesis process and resulting latency and the number of used components.

Table 2. Verilog synthesis results of 4-operand CSA type adder trees with the dedicated fast carry chain with wordlengths n=4,8,16,32 bit

Component	n=4	n=8	n=16	n=32
FD	40	80	160	320
IBUF	16	32	64	129
OBUF	9	17	33	65
LUT2	12	24	48	96
LUT4	4	8	16	32
CARRY4	1	2	4	8
Timing parameters				
Min. period [ns]	1.098	1.098	1.098	1.098
Max freq [MHz]	910.7	910.7	910.7	910.7

Table 3. HLS synthesis results for 4-operand CSA type adders with operand wordlengths n=8,16,32

Component	n=8	n=16	n=32
FD	128	152	184
LUT	156	211	297
Timing parameters			
Min. period	6.56ns	6.56ns	6.56ns

As it can be seen from Table 4 for the greater number of operands and wordlengths of operands the results become convergent. However, there is difference in LUTs but the delay of the HLS synthesized adder is only two times greater.

Table 4. Comparison of 8-operand 16-bit CSA type adders synthesized with HLS and Verilog

Component	HLS	Verilog
FD	687	448
LUT	787	192
CARRY4	0	12
Timing parameters		
Min. period	6.56ns	3.27ns

Conclusions

We have analysed the design of multi-operand adders in FPGAs using the HLS in Xilinx Vivado environment. The obtained results indicate that the circuit obtained as a result of the HLS synthesis requires more resources and is slower than its counterpart design using Verilog. It seems profitable to elaborate a basic component used in the construction in the multi-operand CSA type adders that utilizes the specific properties of the FPGA such as fast carry chain as an element of the basic component which would be used in ASIC solution. For example, for 8-operand 16-bit CSA type adder synthesized with HLS we obtained 787 LUTs and the delay 6.56ns for 8-operand 16-bit CSA type adder synthesized in Verilog we obtained 192 LUTs and 3.27ns. The numbers of LUTs cited here include various types of LUTs from 2-input to 6-input so when recalculated into equivalent 6-input LUTs their number would be remarkably smaller if we consider the equivalence of the memory sized used. The data given in literature does not usually differentiate LUTs with respect to their number of inputs.

***Authors**: dr inż. Robert Smyk, Faculty of Electrical and Control Engineering, Gdansk University of Technology, ul. G. Narutowicza 11/12, 80-233 Gdańsk, E-mail: robert.smyk@pg.edu.pl,*
dr hab. inż. Maciej Czyżak, Faculty of Electrical and Control Engineering, Gdansk University of Technology, ul. G. Narutowicza 11/12, 80-233 Gdańsk, Poland, E-mail: maciej.czyzak@pg.edu.pl

REFERENCES

[1] Wallace C. S., A Suggestion for a Fast Multiplier, *IEEE Transactions on Electronic Computers*, 13 (1964), No. 1, 14-17
[2] Dadda L., Some schemes for fast serial input multipliers, *Alta Frequenza*, 53 (1965), No. 34, 349-356
[3] Gajski D. D., Parallel Compressors, *IEEE Transactions on Computers*, C-29 (1980), No. 5, 393-398
[4] Dormido S., Canto M., Synthesis of Generalized Parallel Counters, *IEEE Transactions on Electronic Computers*, C-30 (1981), No. 9, 699-703
[5] Altera, Stratix-IV device handbook, 2015
[6] Xilinx, Virtex-5 family overview lx, lxt, and sxt platforms, *Xilinx Inc, San Jose, Calif, USA*, 2010
[7] Xilinx, Virtex-6 FPGA data sheets, *Xilinx Inc, San Jose, Calif, USA,* 2010
[8] Parandeh-Afshar H., Neogy A., Brisk P., Ienne P., Compressor tree synthesis on commercial high-performance FPGAs, *ACM Transactions on Reconfigurable Technology and Systems*, 4 (2011), No. 4, art. no. 39
[9] Parandeh-Afshar H., Neogy A., Brisk P., Ienne P., Efficient synthesis of compressor trees on fpgas, *In Proceedings of the 2008 Asia and South Pacific Design Automation Conference, ASPDAC '08, IEEE Computer Society Press, Los Alamitos, CA, USA*, 2011, 138-143
[10] Parandeh-Afshar H., Neogy A., Brisk P., Ienne P., Exploiting fast carrychains of FPGAs for designing compressor trees, *Proceedings of the 19th International Conference on Field Programmable Logic and Applications*, 2009, 242-249
[11] Parandeh-Afshar H., Neogy A., Brisk P., Ienne P., Improving synthesis of compressor trees on FPGAs via integer linear programming, *Proceedings of the Design, Automation and Test Conference in Europe (DATE '08)*, 2008, 1256-1261
[12] Parandeh-Afshar H., Closing the gap between FPGA and ASIC: Balancing flexibility and efficiency, *PhD thesis, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE,* 2012
[13] Kumm M., Zipf P., Efficient high speed compression trees on Xilinx FPGAs, *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen(MBMV '14),* 2014
[14] Kumm M., Zipf P., Pipelined compressor tree optimization using integer linear programming, *Proceedings of the 24th International Conference on Field Programmable Logic and Applications*, 2014, 1-8
[15] Brunie N., de Dinechin F., Istoan M., Sergent G., Illyes K., Popa B., Arithmetic core generation using bit heaps, *3rd International Conference on Field Programmable Logic and Applications, Porto, Portugal*, 2013, 1-8
[16] De Dinechin F., FloPoCo project, [web page] http://http://flopoco.gforge.inria.fr/, Accessed on 28 Aug. 2017
[17] Khurshid B., Mir R.N., High Efficiency Generalized Parallel Counters for Xilinx FPGAs, *EEE 22nd International Conference on High Performance Computing (HiPC)*, 2015, 40-46
[18] Matsunaga T., Kimura S., Matsunaga Y., Power and delay aware synthesis of multi-operand adders targeting LUT-based FPGAs, *Proceedings of the 17th IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED '11)*, 2011, 217-222
[19] Matsunaga T., Kimura S., Matsunaga Y., Multi-operand adder synthesis targeting fpgas, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences,* 94 (2011), No. 12, 2579-2586
[20] Matsunaga T., Kimura S., Matsunaga Y., An exact approach for gpc-based compressor tree synthesis, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences,* E96-A (2013), No. 12, 2553-2560
[21] Xilinx, Vivado design suite user guide : High-level synthesis, ug871, *lXilinx Inc, San Jose, Calif, USA,* 2014
[22] Cony J. et al., High-level synthesis for FPGAs: from prototyping to deployment, *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 30 (2011), No. 4, 473-491
[23] Brent, R.P., Kung, H.T., A Regular Layout for Parallel Adders, *IEEE Transactions on Computers*, C-31 (1982), No. 3, 260-264