Dynamic Constant Coefficient Convolvers Implemented in FPGAs

Ernest Jamro, Kazimierz Wiatr

AGH Technical University, Institute of Electronics
Mickiewicza 30, 30-059 Kraków, POLAND
email: jamro/wiatr@agh.edu.pl

Abstract. This paper describes different techniques for implementing in FPGAs convolution operation for which coefficients are relatively constant. Multiplication is a very basic operation required for convolution, therefore a thorough study of different multipliers (Constant Coefficient Multiplier vs. Dynamic Constant Coefficient Multiplier vs. Variable Coefficient Multiplier), their hardware efficiency and dynamic change of the coefficient are described. Further, different optimisation techniques such as grouping similar coefficients, pipelining optimisation and sharing a common circuit required for dynamic change of coefficients are described.

1 Introduction

An *N*-tap convolution (FIR filter) can be expressed by an arithmetic sum of products:

$$y(i) = \sum_{k=0}^{N-1} h(k) \cdot x(i-k) \qquad \textbf{(1-1)}$$

where: *y(i), x(i)* and *h(i)* represent response, input at the time *i* and the convolution coefficients, respectively.

Multiplication is the most complex operation in the convolver, and several techniques have been adopted to perform it more efficiently. First at all, coefficient values are usually constant, therefore the values of the coefficient can be built-in the circuit, this solution is further denoted as a Constant Coefficient Multiplier (KCMs). The KCM occupies 17÷23% on average or 29÷41% on maximum [1], area of a fully functional, Variable Coefficient Multiplier (VCM). Consequently, the KCM should be implemented whenever the coefficient values are constant. Alternatively, when coefficients are changed infrequently, a part of the FPGA can be reconfigured in order to change the coefficient. A FPGA can be reconfigured in time of a few milliseconds, nevertheless a new multiplier circuit must be redesign and re-implemented, which is much more time consuming than the FPGA reconfiguration. Therefore, this approach can be practically adopted provided that only a finite number of coefficient values are allowed. In this case every coefficient value has a separate pre-implemented entry which can be quickly download into the FPGA.

FPGAs implement logic cells as a Look Up Table (LUT) memory, therefore the inherent way of performing multiplication seems the LUT based Multiplication (LM) [2, 3] where the value of the coefficient is coded into the contents of the LUT

memory. The coefficient change can be obtained by a proper sequence of writes into the LUT memory [1, 4]. This multiplication technique allows for quick change of the coefficient and is further denoted as Dynamic Constant Coefficient Multiplier (DKCM).

The convolver is usually implemented employing multipliers and finally the adders. Nevertheless, disregarding the multipliers entities causes a substantial hardware savings, therefore the convolution operation should not be considered as a separated sum of delayed products. In the case when dynamic change of coefficients is implemented the circuit optimisation is even more complicated as some blocks (RAM programming unit or address multiplexers) can be shared by all multipliers. This causes substantial hardware savings, however causes that much more idle (RAM programming) clock cycles have to be inserted. In conclusion, this paper studies different options of convolvers especially when dynamic change of the coefficients is required.

There are also alternative techniques performing convolution operation which are beyond the scope of this paper basically because they are typical only for the constant coefficient option. First of them is Distributed Arithmetic Convolver (DAC) [5, 6]. The DAC similarly like the LM employs LUT memory however the convolution is carried out in a bit-plane order. Consequently all inputs to the DAC LUTs are at the same bit-significance and therefore the width of the LUT data bus is smaller. The LUTs contents of the DAC are obtained in a more sophisticated way than for the LM. Therefore, in practice, for the DAC it is prohibitively difficult to produce a proper LUT write sequence inside the FPGA circuit therefore the DKCM should be used.

The second technique is Multiplierless Convolution [9] which is carried out in a similar way as for the Multiplierless Multiplication (MM) which is described in Section 2.1. However for the Multiplierless Convolution, a more complex substructure sharing algorithm is required in comparison to the MM.

2. Constant Coefficient Multipliers (KCMs)

2.1 Multiplierless Multiplication (MM)

The KCM is usually implemented in a multiplierless fashion by using only hardwired shifts and adders from the binary representation of the multiplicand. For example, A multiplied by $B= 14= 1110_2$ can be implemented as $(A<<1)+(A<<2)+(A<<3)$, where '<<' denotes a shift to the left.

There are optimisation techniques described below, which attempts to reduce the area of the MM. Canonic Sign Digit (CSD) [7, 8] representation attempts to reduce the number of non-zero digits. The CSD representation is a signed power-of-two representation where each of the bits is in the set $\{0,1,\bar{1}\}$ (0 – no operation, 1 – addition, $\bar{1}$ – subtraction). The modified conversion algorithm proposed in [3] has been implemented hereby, for this algorithm the digit $\bar{1}$ is introduced only when the total number of non-zero symbols is reduced.

Substructure sharing (SS) [9] is another optimisation technique implemented in this paper. For example, multiplication by $27=11011_2$ can be implemented by the use of

an intermediate variable *tmp,* as it is shown in the following equations: *tmp= a + (a<<1),* and *27·a= tmp + (tmp<<3).* It should be noted that the SS area-reduction is implemented on the CSD.

2.2 LUT based Multiplier (LM)

In FPGAs, a multiplication can be carried out employing Look-up-table (LUT) based multiplication (LM). In order to reduce the LUT memory size the input argument is split and then a tree of adders is used [2, 3, 5]. An example of this is given in Figure 2-1. In addition, optimisation techniques such as LSBs Address Width Reduction (LAWR), Don't-care Address Width Reduction (DAWR), Memory Sharing (MS) have been implemented [3].

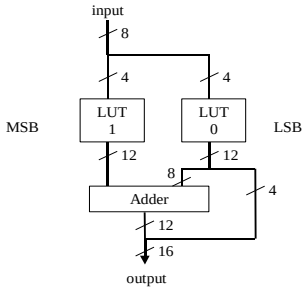

Fig. 2-1. Look-up table based multiplication for 8-bit wide coefficient

Figure 2-1 shows a very simple example for which input is 8 bit wide, therefore the split of the multiplication argument is rather intuitively selected to be 4+4= 8, as FPGAs incorporate mostly 16×1 LUTs. In general case, optimal splitting of the input argument is much more complicated [3]. Besides Virtex family [11] incorporates not only small distributed memory but also several large BlockSelectRAM (BSR) which can be configured for different data bus width: 4k×1, 2k×2, 1k×4, 512×8, 256×16. The area in silicon, occupied by a BSR is equivalent to roughly 16 Virtex CLBs or 64 LEs (a Logic Element (LE) is equivalent to a single 16×1 LUT). However the actual cost (area) of these memories differs with respect to free FPGA resources. For example, a design does not implement any BSRs but uses all available CLBs, therefore it is recommended to allocate more logic into the BSRs.

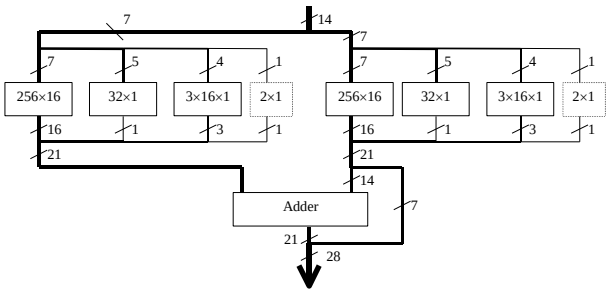

Fig. 2.2. A LM for input and coefficient width equal 14

In order to find an optimal LM architecture, an exhaustive search algorithm (with some obvious simplifications) has been implemented in [3] for which BSRs together with distributed RAMs and adders were combined and the best circuit taken. In order to illustrate considered architectures, an example of the LM for input and coefficient width equal 14 is shown in Figure 2-2.

2.3. Comparison of the KCMs

In Section 2.1 and 2.2 two different multiplication techniques have been presented, and therefore a question arises which of them is more hardware efficient. The statistical cost-relation between the MM and LM for XC4000 is shown in Figure 2-3. A general conclusion can be drawn from Figure 2-3. The MM optimisation techniques (CSD and SS) are more and more efficient with the increase of width *K*. Therefore for greater *K,* the MM is getting more and more attractive in comparison to the LM.

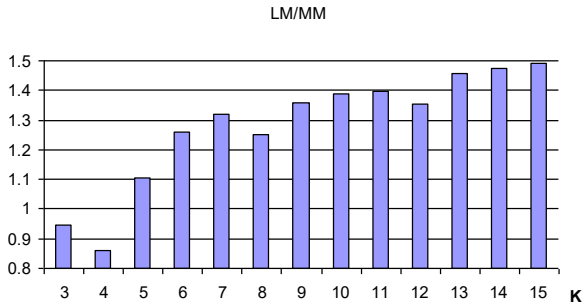

Fig. 2-3. Relation between average area of XC4000 occupied by the LM and MM. Results for the different input width K (input range $0 \div 2^K-1$) and coefficient values $1 \div 2^K-1$

3 Dynamic Constant Coefficient Multiplier (DKCM)

3.1 Concept

The DKCM [1] (or self-configurable binary multiplier [4]) is the LUT based multiplier for which ROMs are replaced by RAMs. The idea behind the dynamic change of the coefficient is to properly change the contents of the memories. This, however, requires an additional RAM Programming Unit (RPU) which feeds RAM memory with proper address and data sequence while changing coefficient value. An example of the DKCM is shown in Figure 3-1. It should be noted that this example is equivalent to the KCM given in Figure 2-1.

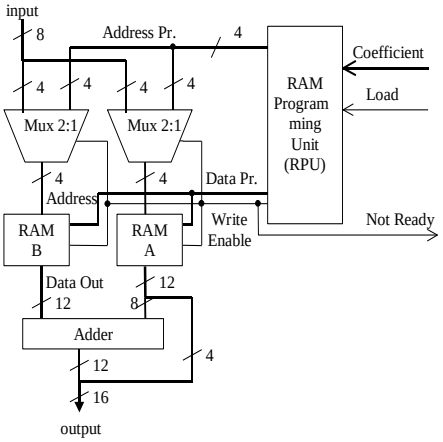

Fig. 3-1. An example of the DKCM for input data and coefficient width equal 8

3.2 Comparison of the KCM, DKCM and VCM

The KCM can be implemented using either the LM or MM (see Section 2) and the MM is getting more and more attractive as the coefficient width increases. Furthermore, even the LM can employ advance optimisation techniques which are suitable only for the KCM [1]. The KCM architecture varies significantly for different coefficients, which causes a great difference in area occupied. Therefore, to compare the DKCM with the corresponding KCMs, two different statistical costs of the KCM can be used: average and maximum area occupied by a KCM for a given coefficient range (usually $1 \div 2^K-1$).

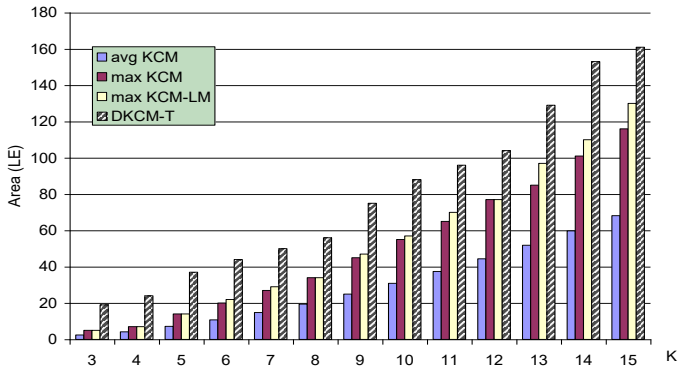

Fig. 3-2. Area for Xilinx XC4000, occupied by: the DKCM-T, maximum area of the KCM-LM and KCM (the best architecture of the MM or LM) and average area for the KCM. The input range $0 \div 2^K-1$ and the coefficient range $1 \div 2^K-1$

The comparison of the KCM and DKCM is given in Figure 3-2. For small values of *K,* area occupied by the DKCM is much greater than for the KCMs due to the strong influence of the RPU; on Figure 3-2 the cost of the RPU is illustrated as the difference between DKCM-T (multiplexing in tri-state buffers) and maximum cost of the KCM-LM. As *K* increases, the relative cost of the RPU decreases, and additional cost of the DKCM over the KCM is related rather to the comparison strategy (the average or maximum cost of the KCM).
The VCM is a fully functional multiplier, usually implemented using AND-gates and adders [2], for which a coefficient-change penalty is not observed. The drawback of the VCM, as can be seen in Figure 3-3, is its large cost in comparison with the DKCM. For small multiplier width *K*, however, the cost of the DKCM is dominated by the RPU, therefore the VCM is recommended.

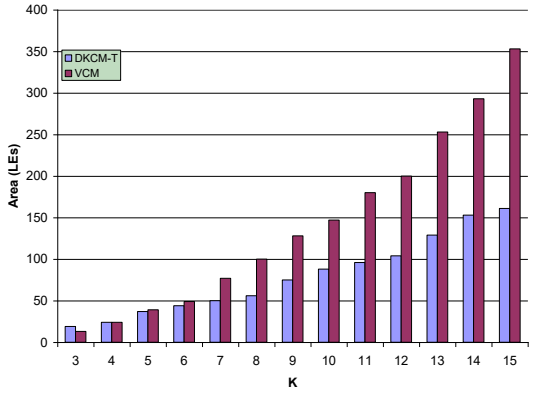

Fig. 3-3. Area (Xilinx XC4000) for the DKCM-T and VCM for different K

Figure 3-3 presents the best results for the DKCM as in real applications the DKCM requires RAMs programming (idle) cycles which decrease the design throughput. Consequently, design density *D,* which is described usually as:

$$D = \frac{1}{A \cdot T}, \tag{3-1}$$

decreases as follows [13]:

$$D_D = \frac{1}{A \cdot T \cdot (1 + \frac{r}{n})} \tag{3-2}$$

where: *D, A, T* – functional density, area and critical delay respectively; *r*- number of reconfiguration cycles, e.g. *r*=16 for 16×1 LUT RAMs; *n*- number of execution cycles between two consecutive reconfigurations.
Comparing Figure 3-3 and eq. 3-2, a conclusion can be drawn that the DKCM should be used for $n>r$ (see [1]), which is the case for most FIR filters, as the coefficients change occurs relatively seldom.

Recently, Xilinx Co. has introduced dedicated 18 bit × 18 bit fully functional multipliers to Virtex II family. Therefore it might seem that the VCM should be employed all the time. Nevertheless the number of dedicated multipliers is limited to 4 for XC2V40 to 192 for XC2V10000, so in a great number of designs the number of

dedicated multipliers is still insufficient, and therefore the standard DKCM or KCM should be still considered.

4 Constant Coefficient LUT based Convolver (KLC)

The structure of the Constant Coefficient LUT based Convolver (KLC) is similar to the sum of products obtained employing LUT based multipliers (LM). However to optimise the structure of the adders, all additions are performed within a single adders block, therefore multiplier entities are disregarded. To illustrate savings obtained by the use of the KLC instead of the sum of the LMs, an example is given in Figure 4-1, for 2-taps convolution and 8×8 multipliers.

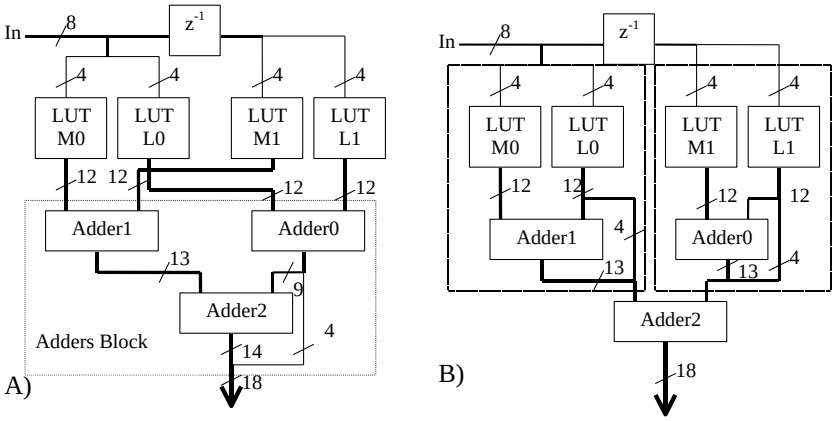

Fig. 4-1. The structure of the convolver for $Y = A \cdot B_0 + z^{-1} \cdot A \cdot B_1$ for input and coefficient width K= 8. A) KLC, B) sum of LM

Figure 4-1 shows 2-tap convolver which is a very simple example. In general case, the adder network is more complicated. Therefore finding an optimal adder tree, i.e. the netlist of adders for which every adder has only two inputs and the total sum of adder widths is the lowest, is a difficult task, which cannot be solved in an intuitive way. Consequently, different optimisation algorithms, such as Exhaustive Search, Greedy Algorithm, Genetic Programming and Simulated Annealing, have been tested [14, 15]. As a result, Greedy Algorithm should be chosen when the circuit generation time is an important factor, otherwise Simulated Annealing is recommended as about 10-20% area reduction of the adders is obtained in comparison to the Greedy Algorithm [15].

The KLC employs the sophisticated optimisation algorithm for the adder tree. However, the KLC requires also optimisation of LUT memory, esp. when different memory modules can be used (see Figure 2-2). For the LM which requires few LUT memories and rather small adder tree, the exhausted search algorithm has been used. Therefore the adder tree and LUT memories are optimised all together. Unfortunately, for the KLC, this optimisation technique is impractical to be implemented. Consequently for the KLC, only the local exhausted search optimisation is implemented, for which each multiplier (the LUT memories and associated adder tree) is optimised separately using the exhausted search technique. Then, all adder

trees associated with each multiplier are merged into a single adder tree which is then separately optimised by the techniques described in [14, 15].

In addition, optimisation techniques characteristic only for convolvers are employed.

Similar Coefficients Optimisation (SCO)

FIR filters are very often implemented as the linear phase filters, for which the impulse respond is symmetric. By exploiting this symmetry the number of multipliers can be nearly halved through mirroring of the signal flow graph in the point of symmetry of the coefficients [8]. Nevertheless, different symmetries and coefficient combinations can be used, especially for 2D filters [16]. Therefore, the AuToCon compares all coefficients and groups them into similar coefficients blocks. Coefficients grouped together can be shifted and negated. Grouped inputs are shifted in respect to the coefficient value and then added (subtracted). Finally, a single multiplier is only implemented. This method allows for reducing the number of multipliers.

For example, for the filter: $H(z)= H_1(z) + 5 \cdot z^{-i} - 5 \cdot z^{-j} - 10 \cdot z^{-k} + 20 \cdot z^{l}$ similar coefficient inputs are added: $A_5= z^{i} - z^{j} - 2 \cdot z^{k} + 4 \cdot z^{l}$, and the final result is: $H(z) = H_1(z) + 5 \cdot A_5$. In this example the number of multipliers has been reduced by 3.

Pipelining Optimization

The AuToCon generates a convolver with a sophisticated pipelining architecture, for which an additional parameter *p* defines maximum number of logic elements between pipelining registers. Figure 4-2a shows an example of a convolver with straightforward pipelining architecture. For this method, however, additional pipelining registers are often required to compensate different pipelining delays. To reduce this drawback, pipelining optimisation is implemented, for which feeding points of arithmetic units are relocated in order to reduce unnecessary registers (similar optimisation is implemented in [9]). A result of the optimisation is shown in Figure 4-2b. It should be noted that the total convolver pipelining delay is often reduced in this method. This optimisation technique is implemented for every architecture described in this paper.

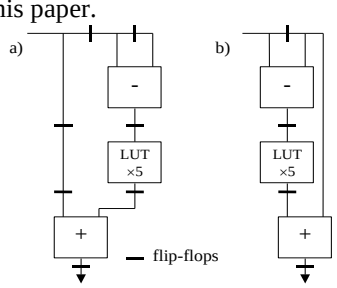

Fig. 4-2. Implementation of $(2 + 5 \cdot z^{-1} - 5 \cdot z^{-2})$ filter a) without b) with pipelining optimization

5 Dynamic Constant coefficients LUT based Convolver (DKLC)

For the DKLC, the value of coefficients can be changed in similar way, as it is in the case for the DKCM; rearranging the order of adders (see Figure 4-1) does not influence the LUTs programming schedule. For the DKCM, address multiplexing is located at the input of each RAM. Similarly for the DKLC, the multiplexer can be placed at the input of each multiplier (RAM). Let us denote this option as DKLC-M. An alternative solution, denoted as DKLC-C, is to place the multiplexer at the convolver input, so the address sequence for programming LUTs will propagate through the convolution delay elements to the input of the LUTs. The drawback of this method is a more sophisticated control logic. Besides, the number of programming cycles increases because of additional propagation time through the filter delay elements.

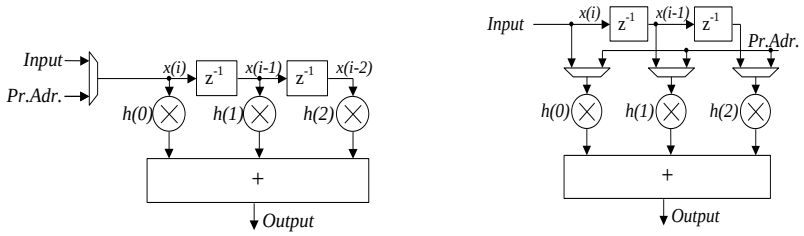

Fig. 5-1. Different strategies for dynamic change of coefficients

It should be noted that the LUTs can be programmed either in sequence: a single multiplier is programmed at the time, or in parallel, when all multipliers are programmed simultaneously. The serial option has longer programming time but a single RAM Programming Unit (RPU) is required. The parallel option has short programming time but each multiplier requires its own RPU and therefore this option occupies more hardware. The choice between the serial and parallel option should be taken after considering the average time between coefficients changes in similar way as it was described in Section 3.2.

6 Conclusions

Convolution is a fundamental operation for digital signal processing, therefore its efficient implementation in FPGAs is getting more and more important, as FPGAs are recognised as the best solution for high speed data driven algorithms. The significant part of this paper describes implementation of different multipliers in FPGA. A multiplication is a very basic operation for the convolution and, what is more important, can be smoothly described by the average or maximum area for the given coefficient and input ranges. The statistic area of the multiplier can be further analysed in order to find the best solution for the given parameters of the convolver.

For example, let's consider a real time image (8-bit representation) convolution for which the coefficient change is required only for different external light conditions (e.g. day/night light). In this case the best solution is employing the KCM (or Distributed Arithmetic Convolution or Multiplierless Convolution) and reconfiguring the FPGA whenever the filter change is required. Another example is a case when

coefficient change may occur after every image frame, in this case the best solution seems the DKCM-C with a single RPU. During image processing often different filter length is required e.g. in order to eliminate padding effect at the beginning and end of each line. In this case DKCM-M and the RPU associated with each multiplier should be considered.

Design automation is one of the most significant design factor, often more important that occupied area. Consequently, an Automated Tool for generating Convolvers in FPGAs (AuToCon) has been designed by the authors of this paper. The AuToCon can therefore automatically generate an optimised and synthesizable VHDL description of the convolver, only input parameters such as: the input width, convolution kernel size, coefficient values, pipelining parameter need to be entered.

References

[1] Wiatr K., Jamro E. *Implementation of Multipliers in FPGA Structures,* Proc. of the IEEE Intern. Symposium on Quality Electronic Design, San Jose, California, 26-28 March 2001, pp. 415-420, IEEE Computer Society Press.

[2] Omondi A.R *Computer Arithmetic Systems. Algorithms Architecture and Implementations,* Prentice Hall 1994

[3] Wiatr K., Jamro E. *Constant Coefficient Multiplication in FPGA Structures*, Proc. of the IEEE Int. Conf. Euromicro, Maastricht, The Netherlands, Sep. 5-7, 2000, Vol. I, pp. 252-259, IEEE Computer Society Press.

[4] Wojko M., ElGindy H., *Configuration Sequencing with Self Configurable Multipliers,* Proc. 10th Sym. on Parallel and Distributed Processing, San Juan, April 1999, pp. 643-651.

[5] Burrus C.S.: *Digital filter structure described by arithmetic*, IEEE Transaction on Circuits and Systems, pp. 674-680, 1977

[6] Do T.T., Reuter C., Pirsch P. *Alternative approaches implementing high-performance FIR filters on lookup table-based FPGAs: A comparison.* SPIE Conference on Configurable Computing and Applications, Boston, Massachusetts, pp. 248-254, 2-3 Nov. 1998.

[7] Garner H. *Number Systems and Arithmetic,* Advances in Computing, vol. 6, pp. 131-194, 1965

[8] Pirsch P., *Architectures for Digital Signal Processing*, Chichester UK, Wiley 1998.

[9] Hartley R.I. *Subexpression Sharing in Filters Using Canonic Signed Digit Multipliers,* IEEE Transactions on Circuits and Systems II – Analog and Digital Signal Processing, vol. 43, no. 10, Oct. 1996.

[10] Xilinx Inc. Core Generator, Foundation 2.1, 1999

[11] Xilinx Inc. *The Programmable Logic Data Book*, San Jose, California, 2000.

[12] Sanchez E., Sipper M., Haenni J., Beuchat J., Perez-Uribe A. *Static and Dynamic Configurable Systems*, IEEE Transactions on Computers, col. 48, no. 6, pp. 556-563, June 1999

[13]Wirthlin M. J., Hutchings B.L., *Improving Functional Density Using Run-Time Circuit Reconfiguration*, IEEE Trans. on VLSI Systems, vol. 6, no. 2, June 1998.

[14] Jamro E., Wiatr K., *Genetic Programming in FPGA Implementation of Addition as a Part of the Convolution*, Proc. of the IEEE Int. Conf. Digital System Design, Warszawa, Poland, 4-6 Sep. 2001, pp. 466-473, IEEE Computer Society Press.

[15] Jamro E., Wiatr K. *Implementation of convolution operation on general purpose processors* Proceedings of the Euromicro Conf. Warszawa, Poland, 4-6 Sep. 2001, pp. 410-417, IEEE Computer Society Press.

[16] Lu W.-S., *Two-Dimensional Digital Filters,* Marcel Dekker, New York, 1992.