# Convolution Operation Implemented in FPGA Structures for Real-Time Image Processing

Ernest Jamro, Kazimierz Wiatr
*AGH Technical University, Institute of Electronics*
*Email: jamro@uci.agh.edu.pl, wiatr@uci.agh.edu.pl*

## Abstract

Addition is an essential operation for the convolution (or FIR filters). In FPGAs, addition should be carried out in a standard way employing ripple-carry adders (rather than carry-save adders), which complicates search for an optimal adder structure as routing order has a substantial influence on the addition cost. Further, complex parameters of addition inputs have been considered e.g. correlation between inputs. These parameters are specified in different ways for different convolver architectures: Multiplierless Multiplication, Look-Up Table based Multiplication, Distributed Arithmetic. Furthermore, different optimisation techniques: Exhausted Search and Simulated Annealing have been implemented, and as a result. Otherwise, the Exhausted Search should be employed for the number of the addition inputs $n\leq 8$, or the Simulated Annealing for $n>8$. Employing the Simulated Annealing gives about 10-20% area reduction in comparison to the Greedy Algorithm. This paper is a part of the research on the AuToCon – Automated Tool for generating Convolution in FPGAs.

## 1 Introduction

An N tap convolution can be expressed by an arithmetic sum of products:

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

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

The multiplication in eq. 1 can be carried out employing three different techniques:

**Multiplierless Multiplication (MM)** [1] where multiplication employs only shifts and additions from the binary representation (BR) 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. To reduce the number of operation (non-zero symbols) required in the coefficient's two's complement representation, canonic signed digit (CSD) representation [2] should be employed. 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). Summing up, for the MM, the FIR filter (convolution) arithmetic can be carried out using only additions and substations.

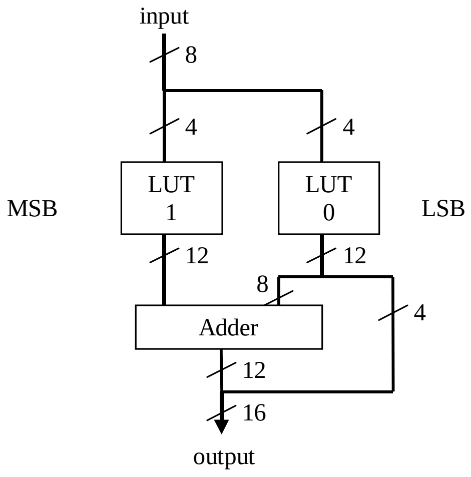

***Figure 1.*** *Look-up table based multiplication for 8-bit wide coefficient*

**Look-up-table based Multiplication (LM).** In principle, the evaluation of any finite function can be carried out using a look-up table (LUT) memory that is addressed with the argument for the evaluation and whose output is the result of the evaluation. Unfortunately, the use of a single LUT for the multiplication is unlikely to be practical for any but the smallest argument, because the table size grows rapidly with the width of the argument. Therefore the solution is to split the argument, use LUTs, and then use a tree of adders [1, 3, 4, 5]. An example of this is given in Figure 1.

**Distributed Arithmetic (DA).** The idea behind the DA [5, 6] is to compute the convolution in different order than for the LM. The following mathematical transformation is employed:

$$y(i) = \sum_{k=0}^{N-1} h(k)\cdot x(i-k) = \sum_{j=0}^{L-1} 2^j \cdot \sum_{i=0}^{N-1} h(k)\cdot x_j(i-k) \quad (2)$$

*where: L- width of the input argument x (in bits), $x_j$(i-k)- - j-th bit of the input argument at time (i-k).*

Consequently, the DA carries out the convolution in a bit-plane order, i.e. every bit of inputs values is considered separately. In comparison with the LM, the DA LUT output width is smaller because the inputs are the same significance. Therefore, the smaller memories are required which makes the DA more hardware-efficient than the LM.

For these three different techniques, the final and very often overlooked operation is addition (subtraction). For example, Thien-Toan Do et. al. [5] constructed the structure of the LM and DA and showed the final adders tree but the order of the additions seems to be intuitive rather than based on a thorough research. This draws a conclusion that general rules for constructing adders tree should be given and/or a design automated tool has to be developed in order to find an optimal order of additions.

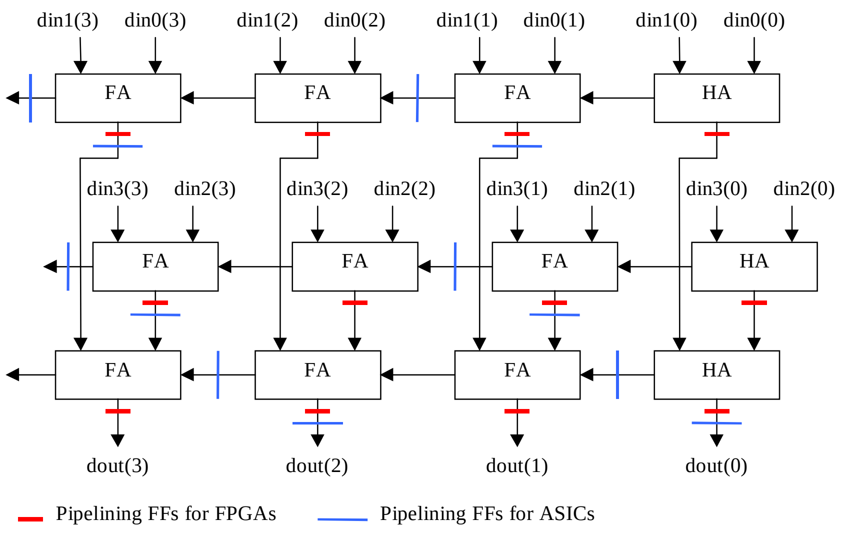

***Figure 2.*** *An example for different pipelining strategies for ASIC's and FPGA's additions for the equation: dout= din0 + din1 + din2 + din3. Pipelining parameters: N=2 (ASIC), K=1 (FPGA)*

For ASIC designs, the classic problem of carry propagation is resolved by numerous techniques, e.g. carry-look-ahead, carry-select [4], which reduce the delay of carry propagation at the expense of great increase in hardware complexity. Another approach to the carry propagation problem is to remove it completely through carry-save addition [4]. Consequently, in ASICs, carry-save addition is a substantial technique implemented in the convolution (FIR filters) designs [7].

FPGAs incorporate a dedicated carry propagate circuit [8, 9] which is so fast and efficient that conventional speed-up methods are meaningless even at the 16-bit level, and of marginal benefit at the 32-bit level [8]. Furthermore the dedicated carry-propagate circuit is implemented outside the standard LUT logic and therefore does not occupy standard logic area. Consequently, using only ripple-carry adders in FPGAs design is the best solution with respect to both the propagation time and occupied area.

As the result, there is a substantial difference between pipelining the ASIC and FPGA additions. For ASIC designs, pipelining flip-flops should be inserted every *N*-logic blocks (where *N* is an integer which value is application specific) therefore the carry-propagation chain is broken as it is shown in Figure 2. For FPGAs, the fast build-in carry logic significantly reduces carry-propagation time and therefore pipelining flip-flops should be rather inserted after every *K* additions (see Figure 2). Nevertheless, the build-in carry logic cannot nullify the carry propagation time, and therefore in the FPGA solution, the most time critical path is the carry-propagate circuit. For example, for Xilinx XC4000, delay through LUT logic, e.g. sum-generation circuit, is approximately six times longer than through the carry-propagate circuit. However, when the programmable interconnects delays are included, which essentially influence overall system performance, the carry propagate delay is much less significant. This holds as FPGAs incorporate dedicated and therefore very fast routing circuit form a carry-out to carry-in. Furthermore the propagation time through the programmable interconnects is usually comparable or even greater than the propagation time through LUT logic.

Nevertheless, in FPGAs, a long-width adder can be divided into several parts by inserting pipelining flip-flops every *M* carry-propagate blocks (like for VLSIs). This solution should be used together with the pipelining solution presented for FPGAs; i.e. a hybrid solution of the FPGA and ASIC designs (see Figure 2) is employed. This, however, would complicate the system design and require additional flip-flops to be inserted according to the cut - set pipelining rule [10]. Therefore, this solution has not been implemented in the presented system, however, is considered in the next step of the design development. This hybrid solution seems to be better than delayed addition technique [11] for which a carry-in does not propagate to carry-out. Conversely, each 4-2 adder has the standard carry-out logic and 2 outputs and therefore 2 flip-flops are required for each 4-2 adder.

Summing up, for FPGAs the best solution seems to be using dedicated addition and pipelining after every *K* additions as it is shown in Figure 2.

## 2 Exhausted search

The best possible result can be always found by searching through all possible solution. The problem of finding the best solution for adders tree is NP-complete and therefore only simple adders blocks can be routed using the exhausted search algorithm. At first, let consider an example of 5 input adder. The following solutions have to be examined (the bottom layer is only taken into consideration, the example shows how inputs (latters: *a* to *e*) are paired together):

$(a+b)+(c+d)+e;$ $(b+c)+(a+d)+e;$
$(c+a)+(b+d)+e;$
$(b+c)+(d+e)+a;$ $(c+d)+(b+e)+a;$
$(d+b)+(c+e)+a;$
$(c+d)+(e+a)+b;$ $(d+e)+(c+a)+b;$
$(e+c)+(b+a)+b;$
$(d+e)+(a+b)+c;$ $(e+a)+(d+b)+c;$
$(a+d)+(e+b)+c;$
$(e+a)+(b+c)+d;$ $(a+b)+(e+c)+d;$
$(b+e)+(a+c)+d;$

In order to find out the number of possible combinations, at first let define the function $S_1(n)$ which returns the number of all possible combinations within a single adder layer for a given number of inputs *n*:

$$S_1(n) = \begin{cases} n \cdot (n-2) \cdot (n-4) \cdot \ldots \cdot 3 \cdot 1 & \text{for odd } n \\ S_1(n-1) & \text{for even } n \end{cases} \qquad (3)$$

The total number of possible solutions $S(n)$ is defined in an iterative way and is a product of the number of combination on this layer and the total number of combination for the upper (closer to the output) layers, i.e. for adders block for which number of inputs is halved:

$$S(n) = S_1(n) \cdot S(\lceil n/2 \rceil) \qquad (4)$$

*where* $\lceil\ \rceil$ *- the ceiling function.*

***Table 1.*** *The number of possible combinations for a given number of inputs n to the adder block*

| *N* | no. layers | no. combinations |
|---|---|---|
| 2 | 1 | 1 |
| 3 | 2 | 3 |
| 4 | 2 | 3 |
| 5 | 3 | 45 |
| 6 | 3 | 45 |
| 8 | 3 | 315 |
| 10 | 4 | 42 525 |
| 12 | 4 | 467 775 |
| 14 | 4 | 42 567 525 |
| 16 | 4 | 638 512 875 |
| 18 | 5 | 1 465 387 048 125 |

It can be seen from Table 1 that the number of possible solutions is growing rapidly, making the exhausted search (ES) method useless for the input number greater then about 11-16.

As the number of possible solutions is growing rapidly with the growing number of adder inputs, a modification of the Exhausted Search (ES) method is here proposed. This method considers at first the cost of the GrA solution for every layer *l*. Consequently, the cost *C(l)* of the partially routed adder (up to the adder layer *l*) is first calculated (initially using the GrA) for every layer *l* and then the similar to exhausted search method is implemented. This method, however, stops calculating a group of solutions in its early stage (on layer *l*) if the cost of the partially routed adder is greater than $C_b(l) + t$; where: $C_b(l)$ – the cost *C(l)* for the best overall solution so far found (initially found by the GrA), *t*- a certain threshold number. The procedure of comparison is executed after every layer of the adders tree is completed.

***Table 2.*** *Theoretical number of considered solutions for different number of adder inputs N*

| *N* | ES | CS (layer 1) | CS (layers 1 and 2) |
|---|---|---|---|
| 6 | 45 | 15 | 45 |
| 8 | 315 | 105 | 315 |
| 10 | 42 525 | 945 | 14 175 |
| 12 | 467 775 | 10 395 | 155 925 |
| 14 | 42 567 525 | 135 135 | 14 189 175 |
| 16 | 638 512 875 | 2 027 025 | 212 837 625 |
| 18 | 1 465 387 048 125 | 34 459 425 | 32 564 156 625 |

The Constrained Search (CS) technique saves the calculation time, as solutions which are less-likely to give the better solution are skipped on a low layer and therefore upper layers and their combinations are not calculated for the given partially routed adder. Conversely, it is possible then an adder block has a very high cost on the bottom layer(s), however the upper layers are much less costy, and therefore this adder block solution is skipped although it would give the best result. Consequently, the key problem is a proper choice of the threshold number *t*. Increase of the threshold number *t* increases the total number of considered solutions but decreases the probability of not finding the best solution.

Table 2 shows the theoretical number of possible solutions for the CS assuming that the calculation process is constrained only to layer 1 and layer 1 and 2. It can been seen that the total number of considered solution has decreased significantly, however it is still unacceptable for the inputs number, *N*, greater than 18.

In this section the results for the greedy algorithm (GrA), exhausted search (ES) and constrained search (CS) algorithms are given. Table ***3***3 shows the cost of the generated circuits by GrA, ES and CS (for different thresholds *t*). Table 4 shows the calculation cost – the number of iteration needed to find the circuit.

***Table 3.*** *The implementation costs (number of full or half adders) for different filters and techniques*

| Filter | no inputs | ES | CS (t=5) | CS (t=2) | CS (t=0) | CS (t=-1) | GrA |
|---|---|---|---|---|---|---|---|
| a | 16 | 93 | 93 | 93 | 93 | 93 | 111 |
| b | 11 | 72 | 72 | 72 | 73 | 73 | 74 |
| c | 13 | 123 | 126 | 126 | 126 | 126 | 128 |

***Table 4.*** *The number of iterations for different filters and techniques*

| Filter | layer 1 | Layer 1,2 | ES | CS (t=5) | CS (t=2) | CS (t=0) | CS (t=-1) |
|---|---|---|---|---|---|---|---|
| a | 2 027 025 | 212 837 625 | 638 512 875 | 9 556 259 | 4 881 543 | 2 963 651 | 2 327 927 |
| b | 945 | 14 175 | 467 775 | 444 927 | 278 735 | 80 051 | 13 173 |
| c | 135 135 | 14 189 175 | 42 567 525 | 3 079 789 | 915 375 | 369 357 | 193 057 |

It can be seen from Table 3 and Table 4 that acceptable results are achieved using only the GrA. The improvement of about 2-7 % can be obtained by the use of the ES. The drawback of the ES is its computation cost therefore the reasonable solution seems the CS (for the number of inputs up to 16). For the threshold *t=-1*, only partial solution which is better than the best found is taken into consideration. This makes the CS (*t=-1*) similar to a GrA for which the step is constrained not only to the single adder (like for the GrA) but for all adders in the layer. Besides for the CS, it is always possible to undo a selection if the upper layers cost is high and therefore the overall cost of the new solution is higher than the best previously found cost. By the increase of the threshold *t*, the number of considered solutions is growing. For *t=0*, not only the best but also all solutions on the same cost are also considered. This however increases the number of iterations but very slightly influences the overall results. Similarly is for *t=2* and *t=5*.

It should be noted that the GrA behaves more poorly for filters for which subtraction is implemented (for negative coefficients, examples A, C) as this algorithm deals with subtraction and addition in the same way.

## 3 Simulated Annealing (SA)

The principle behind the SA [12, 13] is analogous to what happens when metals are cooled at a controlled rate. The slowly falling temperature allows atoms in the molten metal to line themselves up and form a regular crystalline structure that has high density and low energy. In the SA, the value of an objective function which we want to minimise, is analogous to the energy in a thermodynamic system. At high temperatures, SA allows function evaluations at faraway points and it is likely to accept a new point at higher energy. At low temperatures, SA evaluates the objective function only at local points and the likelihood of it accepting a new point with higher energy is much lower.

The SA algorithm, implemented for optimising adders structures, employs the following steps:

**Objective function** calculates the cost *C* of the circuit for the given adders tree.

**Annealing Schedule** regulates how rapidly the temperature, *T*, goes from high to low values, as a function of iteration counts. In our case, the starting temperature $T_1$ equals the cost of a 2-bit wide adder $C_{A2}$, the stopping temperature $T_S$ equals ¼ of the cost of a 1-bit wide adder $C_{A1}/4$. In every iteration, the temperature $T_i$ is decreased according to the following equation:

$$T_{i+1} = \eta \cdot T_i \qquad (5)$$

*where* $\eta = (\frac{T_1}{T_S})^{1/S}$ *, S- the number of iterations.*

**Generating a new adders structure** is obtained by randomly selecting two adders on the same layer; i.e. randomly selecting a first adder (input) from all adders and randomly selecting a second adder from adders at the same layer as the first adder. Examples of possible modification are given in Figure 3.

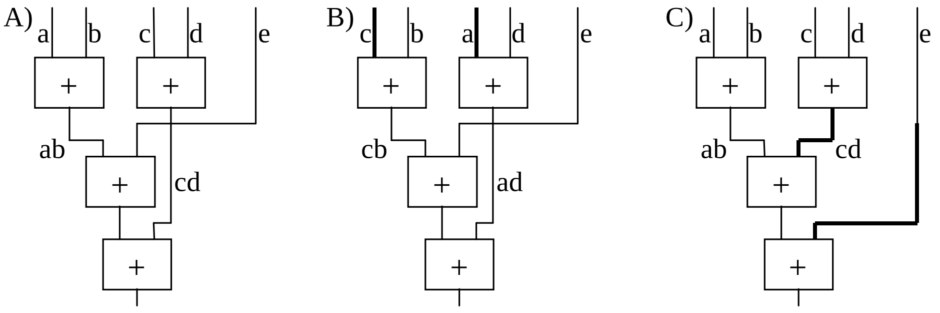

***Figure 3.*** *Examples of possible one-step modification: A) an initial circuit, B) C) the modified circuit A*

Modifications of the circuit are constrained by the temperature $T_i$. In the conventional SA, also known as the Boltzmann machine, the generating function which specifies the change of the input vector, is a Gaussian probability density function [14]. In our approach, the number of possible solutions is finite therefore the Gaussian probability function is useless. An alternative solution is defining a move set [14], denoted by M(x), as a set of legal points available for exploration. However, constructing the move set is rather computationally demanding task thus not implemented. In our approach, therefore, two adders are selected randomly (but at the same adders layer) and then a local acceptance function (LAF), which is further described in the next paragraph, is calculated. The local acceptance function differs from the (global) acceptance function as it takes under consideration only the cost of the two involved adders before and after modification. If modification is not accepted locally, the change is rejected and the next modification is randomly generated (the iteration counter and temperature are not affected in this step).

**Acceptance function**. After a new network of adders has been evaluated, the SA decides whether to accept or reject it based on the value of an acceptance function *h( )*. The acceptance function is the Boltzmann probability distribution:

$$h(\Delta C, T) = \frac{1}{1+\exp(\Delta C / T)} \qquad (6)$$

*where:* $\Delta C = C_{i+1} - C_i$ *- the difference of the adders cost for the previous and current adders tree.*

The new circuit is accepted with probability equal the value of the acceptance function.

The result for the SA, for different circuits are given in Table 5. It can be seen that for the filter *a*, the result equal *103,* the best possible – the same as for the ES, is obtained already for 1000 iteration. For the filter *c*, the cost equal *123,* the same as for the ES, was obtained already for 30k iterations; for the CS, the cost is 126 even for more than 3M iterations. It should be, however, noted that the computation cost of a single iteration is lower for the CS than for the SA. This holds as for the CS and ES, the change in the circuit is well-defined and usually constrained only to the upper layer of the adders and therefore only a part of the circuit has usually to be re-calculated. For the SA, the change is done randomly and on every part of the circuit, therefore cost of the whole circuit has to be calculated again. The lower calculation cost for the CS and ES, does not, however, compensate much greater number of iterations required to obtained the same result. Consequently, the overall calculation cost of the CS is usually greater than for the SA, however for small circuits for which the calculation cost is very low, the CS and ES are good alternatives to the SA.

***Table 5.*** *The circuit costs for the GrA, ES and different number of iterations for the SA; wlaf – without local acceptance function*

| Ex | GrA | SA 1k | SA 30k | SA 1M | ES |
|---|---|---|---|---|---|
| a | 111 | 93±0 | 93±0 | 93±0 | 93 |
| c | 128 | 126.9 ±0.3 | 125 ±1.4 | 123 ±0 | 123 |
| d | 413 | 394 ±3 | 385 ±1 | 382 ±1 | - |
| D(wlaf) | 413 | 398 ±4 | 382 ±1 | 380 ±1 | - |
| e | 1358 | 1346 ±10 | 1299 ±3 | 1292 ±4 | - |
| e (wlaf) | 1358 | 1341 ±9 | 1293 ±4 | 1283 ±4 | - |
| f | 3730 | 3702 ±20 | 3338 ±14 | 3245 ±6 | - |
| f (wlaf) | 3730 | 3706 ±13 | 3419 ±13 | 3296 ±8 | - |

In our solution, the final circuit (obtained in the lowest temperature) is often not the best one. Therefore, the best-obtained circuit is every time stored as the result; this increases calculation cost insignificantly but allows for substantial savings.

Table 5 shows also the results when local acceptance function (LAF) is not implemented (option: wlaf). Calculating local cost before and after modifications, insignificantly influences the total calculation cost and the LAF usually rejects solutions which are unlikely to generate a good global result. Conversely, the LAF constrains search space and therefore may cause some good solutions to be omitted. This is often the case for relatively small adders circuits and for large iteration numbers. For example, it can be seen in Table 5 that not implementing LAF gives better results for the circuit *d* and for small number of iterations, however spoils results for more complicated circuit *f*.

## 4 Conclusions

In this paper, thorough analysis of addition as a part of the FIR filters has been presented. Complex input parameters of the adder block have been considered: inputs range (not only input width), inputs shift and even the correlation between inputs. Consequently, finding an optimal network of the adders tree is a difficult task which has been investigated. Different approaches as: greedy algorithm, exhausted search, simulated annealing and genetic programming have been implemented and the results given. Conversely, the best solution is obtained by checking all possible solutions in the ES, however the calculation time is unacceptable for the number of inputs, *n*, greater than about 12. Therefore, the Constrained Search (modification of the ES) has been proposed. For the CS, each layer of the addition is considered, in some degree, separately. The CS checks less solutions however the number of solutions increases rapidly with growing *n*, and therefore, this solution can be implemented for the number of inputs *n* less than about 14 – insignificant improvement in comparison to the ES. Further, the Simulated Annealing technique has been implemented. For small *n*, the SA usually finds the best solution and requires much lower number of iterations in comparison to the ES. However, for $n\le 8$, the ES searches at most 315 solutions and therefore the computation cost is low. Therefore for $n\le 8$ the ES solution should be implemented. For $n\ge 9$, the ES goes through at least 42 525 solutions therefore the SA should be rather used.

Addition block is a part of the convolution and therefore all procedures, described in this paper, are included in the Automated Tool for Convolution in FPGAs (AuToCon). The AuToCon [1, 15 16] generates an optimised VHDL code of the convolver for the given coefficient values.

## 5 References

[1] Wiatr K., Jamro E.: *Constant Coefficient Multiplication in FPGA Structures*, Proceedings of the 26th Euriomicro Conference, Maastricht, The Netherlands, Sep. 5-7, 2000, Vol. I, pp. 252-259.

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

[3] Chapman K.: *Fast Integer Multiplier fit in FPGA's*, EDN 1993 Design Idea Winner, END May 12th 1994

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

[5] 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, 2-3 Nov. 1998, pp. 248-254

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

[7] Hawley R.A., et, al. *Design Techniques for Silicon Compiler Implementation of High-Speed FIR Digital Filters*, IEEE Journal of Solid-State Circuits, vol. 31, no 5, May 1996, pp. 656-667

[8] Xilinx Co. *The Programmable Logic* Data Book 1999.

[9] Altera Co. *Apex 20K Programmable Logic Device Family, Data Sheet*, ver. 2.05, Nov. 1999

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

[11] Luo Z., Martonosi M. *Using Delayed Addition Techniqus to Accelerate Integer and Floating-Point Calculation in Configurable Hardware,* SPIE Conference on Configurable Computing: Technology and Applications, Boston, Massachusetts, Nov. 1998, Vol. 3526, pp. 202-211

[12] Aarts, E.H., Korst, J. *Simulated Annealing and Boltzman Machines*, Wiley, Chichester, UK, 1989

[13] Kirkpatrick, S. Gelatt, C.D., Vecchi, M.P. *Optimisation by simulated Annealing*, Science, 220 (4598): , May 1983, pp. 671-680

[14] Jang J.S.R., Sun C.T., Mizutani E.: *Neuro-Fuzzy and Soft Computing*, Prentice-Hall, London, UK, 1997

[15] Wiatr K., Jamro E.: *Implementation of Multipliers in FPGA Structure,* Proceedings of the IEEE International Symposium of Quality and Electronic Design, San Jose - March 2001, IEEE Press 2001

[16] Wiatr K. Jamro E.: *Implementation of Image Data Convolutions Operations in FPGA Reconfigurable Structures for Real-Time Vision Systems.* International IEEE Conference on Information Technology: Coding and Computing, Nevada 2000, IEEE Computer Society – Washington – Brussels – Tokyo 2000, pp. 152-157