Implementation of Multipliers in FPGA Structures

Kazimierz Wiatr, Ernest Jamro
AGH Technical University of Cracow, Mickiewicza 30, 30-059 Krakow, Poland

Abstract

This paper studies different solutions for carrying out multiplication: a fully functional multiplier denoted as Variable Coefficient Multiplier (VCM), Constant Coefficient Multiplier (KCM) and self-configurable multiplier denoted as Dynamic Constant Coefficient Multiplier (DKCM). For FPGAs which can be easily reconfigured, the choice between the VCM and KCM cannot be easily defined. Furthermore, the DKCM is an additional, middle-way between the KCM and VCM solution, as it offers shorter reprogramming time but occupies more area in comparison with the KCM. In FPGAs, the choice of the optimum multiplier involves three factors: area, propagation and reconfiguration time, which have been thoroughly studied and respective implementation results given. Furthermore, to speed-up implementation of multipliers a design-automated tool has been developed, which generates optimum (for given input parameters), VHDL description of multipliers.

1. Introduction

Bit-parallel multiplication is a very common operation in digital signal processing and can be carried out implementing three different methodologies. The first is a variable coefficient (fully functional) multiplier (VCM) which can be implemented using for example parallel-array multipliers or Wallace tree multipliers [1]. For the VCM, a coefficient value can be freely changed but the disadvantage of this solution is a relatively high cost. The alternative solution is a Constant Coefficient Multiplier (KCM) which in comparison to the VCM has much lower hardware requirements (26-33% of the VCM [2,3]), and therefore is recommended provided that the coefficient is constant during a calculation process. For ASIC designs the coefficient value once determined cannot be changed. Conversely for FPGAs, the change of a coefficient value can be implemented by reconfiguring the FPGA structure. The process of reconfiguration usually takes several ms [4]; therefore if the calculation process can be ceased for that time and the coefficient is relatively constant during data processing [5], the KCM solution should be considered. The reconfiguration time can be however reduced by the use of a partially reconfigurable FPGA, e.g. a Virtex FPGA [4]. The KCM solution has another drawback that the multiplier circuit has to be redesigned for a different coefficient value. Fortunately, by the use of an Automated Tool (AT) the KCM multiplier can be redesigned [6] within the time of seconds. However a new design has to re-employ a place and route program which fits the new design into the FPGA. The fitting process is usually time-consuming and takes approximately 1min ÷ 1hour. In conclusion, the change of a coefficient value for the KCM requires not only the FPGA to be reconfigured but also the whole design cycle to be reimplied. This causes that the change of a coefficient value for the KCM solution is onerous and often the more-hardware-consuming VCM solution taken instead.

An alternative solution is a Dynamic Constant Coefficient Multiplier (DKCM). The DKCM implements Look up table based Multiplication (LM) [3,7,8] and the change of the coefficient can be achieved by a proper change of LUT memory contents. This solution can implement in-circuit coefficient reconfiguration therefore the multiplier configuration time is shorter and the design fitting into FPGA structure need not be reimplied. A drawback of the solution is that the DKCM occupies more area in comparison with the KCM.

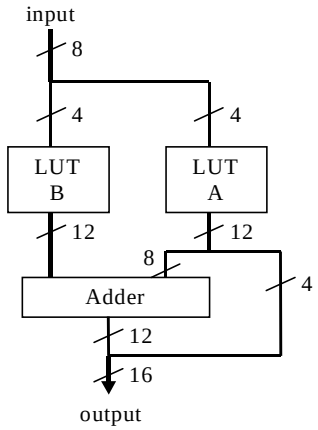

Figure 1. The LM for input argument width K=8 and coefficient width L=8

At the first part of this paper the LUT based multiplication (LM) and its modification – the DKCM is presented. For the DKCM three different options: multiplexing in logic, multiplexing as in tri-state buffer and dual port are studied. Further the comparison of the KCM, DKCM and VKM and their implementation results are given.

2. LUT 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. This, in theory, gives the fastest possible implementation, since no actual arithmetic is required. 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. For example, for the *L*-bits wide argument and *K*-bits wide coefficient, the size of memory is $(L+K)\cdot 2^L$, which for $K=8$, $L=8$ gives 4k bits. It is, however, possible to create a practical implementation of the LM by combining a number of small LUTs and adders. The idea is to split the argument, use LUTs, and then use a tree of adders [2, 7, 8]. An example of the multiplier circuit for $K=8$ and $L=8$ is shown in Figure 1.

3. Dynamic Constant Coefficient Multiplication (DKCM)

The DKCM [6] (or self-configurable binary multiplier [9,10]) is the LUT based multiplier for which ROMs are replaced by RAMs. The idea behind the dynamic change of a coefficient value is to properly change the contents of the memories. This however requires an additional RAM programming interface and imposes constrains on the DKCM architecture in comparison to the KCM.

The additional RAM programming interface can be divided into two parts. The first part allows the RAMs to be programmed and usually consists of address and (rather seldom) data multiplexers. The second part of the additional circuit is RAM Programming Unit (RPU) which produces proper data sequences and control signals for RAM programming. An example of the DKCM is shown in Figure 2. It should be noted that this example is equivalent to the KCM given in Figure 1.

The outputs of the RAM feed the adder block, which structure must also consider the change of the coefficient. The easiest solution is to calculate only the sum of maximum and minimum values of each input (the inputs correlation is not considered), however this usually causes overheads of the adder width. Therefore, the correlation amongst the adder inputs should be considered. An example of the adder width with and without correlation is given in Table 1. It should be noted that for the KCM, the correlation between adder inputs influences only slightly the adder width. Table 1 considers the adder with only two inputs, in general case, however, the correlation between partial sums should be also taken into consideration.

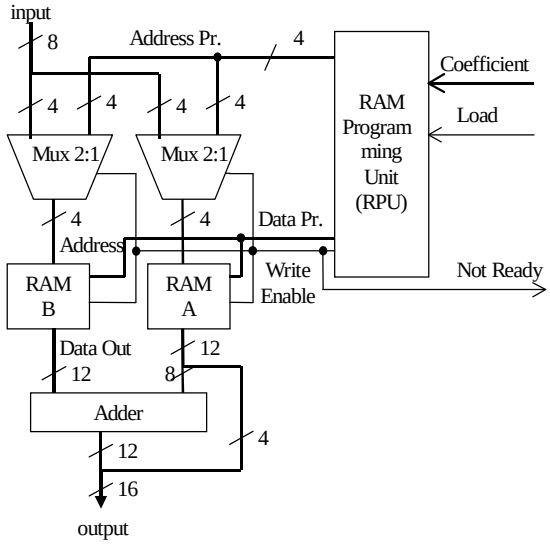

Figure 2. An example of the DKCM for input data and coefficient width equal 8

It can be seen from Figure 2 that RAM memories usually have separated paths for data reads and data writes [4], therefore data multiplexing is not required. Unfortunately, the address bus is the same for reading and writing RAMs, therefore additional multiplexers for switching between these addresses have to be implemented. The multiplexing process can be carried out using Logic Elements (LE), e.g. in Configurable Logic Blocks (CLBs) [4] or tri-state buffers (TSBs). The latest solution consumes no logic area but uses the programmable interconnect resources which can be limited and slower than multiplexing in LEs.

	KCM, coeff= -99			KCM, coeff= 99			DKCM, coeff= -99 ÷ 99		
	min	max	width	Min	max	Width	min	max	width
A	-1 485	0	12	0	1 485	11	-1 485	1 485	12
B	-594	693	11	-693	594	11	-693	693	11
Y no-corr.	-10 989	11 088	15	-11 088	10989	15	-24 453	24 453	16
Y corr.	-9 801	9 801	15	-9801	9801	15	-9 801	9 801	15

Table 1. Minimum and maximum values and width of the outputs: A- RAM A (see Figure 2), B- RAM B, Y- output of the adder without and with correlation between the adder inputs. The range of the input (-99, 99)

The multiplexing process can be skipped by the use of dual-port (DP) RAMs. The DP-RAM solution is usually quicker (without multiplexer delay) but consumes more area. For example, for Virtex, a 16×1DP distributed RAM consumes the area of two single-port (SP) RAMs, but a large 4kb Block Select RAM (BSR) can be used as a DP RAM without any hardware overheads. Summing up,

design optimisation should consider three different options:

- Multiplexing using logic (LEs) recourses (denoted as DKCM-L)
- Multiplexing using programmable interconnect (TSB) resources (DKCM-T)
- Using dual-port RAMs (DKCM-D).

The main task of the RPU is to provide the memory with write address and data. Let consider, at first, the case when input data in always positive and all memory modules have the same address width. In this case all memories are fed with the same address and data, therefore the RPU consists of address counter and the accumulator which starts with value zero and is incremented every clock cycle by the coefficient value [10]. In consequence, the data sequence is as follows:

$$d_0 = 0$$
$$d_1 = d_0 + coeff = coeff$$
$$d_2 = d_1 + coeff = 2 \cdot coeff \qquad (1)$$

where d_i- write data for address value i, coeff- the coefficient value.

It should be noted that the number of memory writes (the number of the multiplier standstill clock cycles) depends on the memory size, e.g. for RAM 16×1, sixteen memory writes are required. Therefore in some application it may be beneficial to use only a part of memory in order to reduce the multiplier standstill time. However, this causes that the multiplier consumes more hardware.

The RPU becomes more complicated if memory sizes (address widths) are different because either different memory modules have been implemented or the input data width can not be evenly distributed into separate memories. In this case, each memory write-enable signal should be disasserted whenever the write address exceeds the memory address width. This however may require additional write-enable logic to be implemented. The write-enable problem can be solved by programming RAMs form the highest address (all ones) down to zero. In this way, all memories can be written disregarding address width because the latest memory writes are always proper and overwrite the previous (improper) writes. The data sequence for programming RAMs is therefore as follows:

$$d_{s-1} = (coeff << w) - coeff = (s-1) \cdot coeff;$$
$$d_{s-2} = d_{s-1} - coeff = (s-2) \cdot coeff$$
$$....$$
$$d_0 = d_1 - coeff = 0 \qquad (2)$$

where: $coeff << w$ - the coefficient shifted w bits to the left, w- maximum width of memory address, s- maximum size of memory $s = 2^w$.

The drawback of the above solution is that an multiplexer 2:1 is required (instead of the reset circuit for eq. 1) for feeding the subtractor either with $(coeff << w)$ or d_{i+1}.

The RPU is further complicated for negative (two's complement) inputs. In this case all RAMs except from the Most Significant Bits (MSBs) RAM, operate on positive inputs therefore can be programmed as above. For the MSB RAM and for the MSB (sign bit) zero, the MSB RAM is programmed as the rest of RAMs. Conversely, if the sign bit is asserted then the RAM has to be programmed with a different data sequence which can be generated by continuing eq. 2, as follows:

$$d_0 = d_1 - coeff = 0$$
$$d_{-1} = d_0 - coeff = -coeff$$
$$d_{-2} = d_{-1} - coeff = -2 \cdot coeff \qquad (3)$$
$$...$$
$$d_{-sn} = d_{-sn+1} - coeff = -sn \cdot coeff$$

where: sn- the size of the MSB RAM divided by 2.

It should be noted that eq. 3 does not require additional hardware, it uses the same address counter and subtractor as eq. 2 does. However, programming two's complement input multiplier requires additional control logic for write-enable signals. Consequently, the MSB RAM write-enable is asserted during whole programming process; for the rest of the RAMs, the write-enable signal is asserted only for eq. 2 and disasserted for the rest of eq. 3. It should be noted that the two's complement input format causes that the multiplier programming (standstill) time is longer.

4. Implementation results

The optimal architecture depends strongly on a given FPGA device; therefore at first implementation results for Xilinx XC4000 family [4] will be studied. The multiplication requires mainly 2:1 multiplexing, addition and RAM units, therefore only these modules will be considered here. The XC4000 incorporates single-port (SP) 16×1 and 32×1 and dual-port (DP) 16×1 distributed RAMs at the cost of 1, 2 and 2 Logic Elements (1LE ≈ 4-input LUT ≈ ½ XC4000 CLB) respectable, and an adder with dedicated ripple carry logic at the cost of 1 LE/bit. A 2:1 multiplexer consumes 1 LE if implemented in logic, or only programmable interconnects resources if implemented as a tri-state buffer.

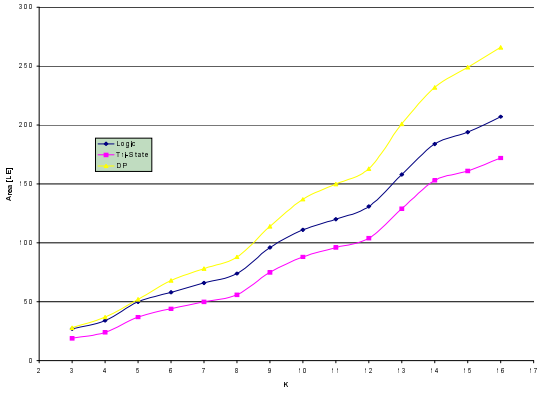

Figure 3. Area occupied by the DKCM for different input and maximum coefficient widths K. Implementation for XC4000 and unsigned coefficients and inputs

The area occupied by the DKCM for different parameters is shown in Figure 3. The relatively high cost

of the multiplier for low values of *K* (input and coefficient width) can be seen in Figure 3. This is a consequence of the RAM programming unit (RPU) which has a great influence on the overall multiplier cost, e.g. the RPU for *K= 3 and 16* occupies 63% and 23% of the whole DKCM-L area respectively.

Up to now, XC4000 family, which incorporates only small distributed RAMs, has been considered, but additional RAM resources are available in Virtex [4] which incorporates large DP 4k×1, 2k×2, 1k×4, 512×8 and 256×16 BlockSelect RAMs (BSRs).

Constructing the optimal multiplier using large BSRs, distributed RAMs and adders is however a difficult task which involves many trade-offs:

- Cost relations between BSRs, logic elements (LEs) and distributed RAMs. The chip area occupied by 1 BSR is roughly 16 Virtex CLBs ≈ 64 LE, but the real cost-relation is application- and resources-dependent, as free BSRs can be implemented instead of fully used logic elements and vice-versa.
- The multiplier programming time is proportional to the memory size, therefore in applications where operation standstill time is a critical factor, the smaller memory blocks are preferable.
- Multiplication delay time tends to be lower with larger memory blocks as the number of arithmetic blocks decreases. Conversely, the memory access time usually increases with the memory size, and routing large RAMs with arithmetic modules is more difficult as the BSRs have fixed position in FPGAs and cannot be freely mixed with adders as it is the case for distributed memories. The case is even more complicated for pipelined architectures where a cost of additional flip-flops and a frequency of the system clock have to be considered.

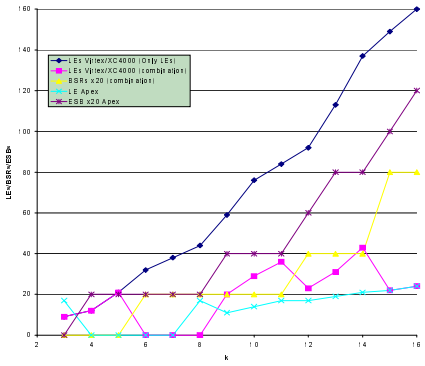

Figure 4. Number of LEs and BSRs (scaled 20:1) for Virtex (using large BSR), and number of LEs for XC4000/Virtex (using only small distributed RAMs), and number of LEs and ESBs (scaled 20:1) for Apex. The RPU is not considered, the equivalent costs: 1 BSR = 1 ESB= 20 LEs

The implementation results for combination of large BSRs and small distributed RAMs, and for only small distributed RAMs is shown in Figure 4. Note that the number of used BSRs depends on the equivalent cost of the BSR; and for the equivalent cost greater than roughly 44 LEs (11 Virtex CLBs)- the BSRs are not used at all.

Altera Apex [11] family also incorporates dedicated ripple carry logic at the cost of roughly 1 LE/bit; but in comparison with the Xilinx FPGAs can implement only large DP RAMs: 2k×1, 1k×2, 512×4, 256×8 or 128×16, one in each Embedded System Block (ESB). Consequently, as it can be seen in Figure 4, the number of required ESBs in comparison with BSRs is greater, however the number of LEs is reduced. The next consequence of the lack of distributed RAMs in Apex FPGAs is the longer coefficient reprogramming time in comparison with the Xilinx FPGAs when only distributed RAMs are used.

The Multiplier Automated Tool (MAT) which was developed in order to generate the multipliers, uses advance full search algorithm which generates the best solution from the given input parameters: input data range, coefficient range and cost relations between adders, memories, multiplexers and flip-flops, etc. These parameters can be freely specified therefore the MAT can generate VHDL code for any FPGAs and even ASICs. In order to visualise architectures analysed by the MAT, an example of the optimum structure of the multiplier for *K*= 12 is given in Figure 5. Note that in this example only dual port memories and AND gates are implemented therefore the input multiplexing is not required, and the RPU is not shown in the figure.

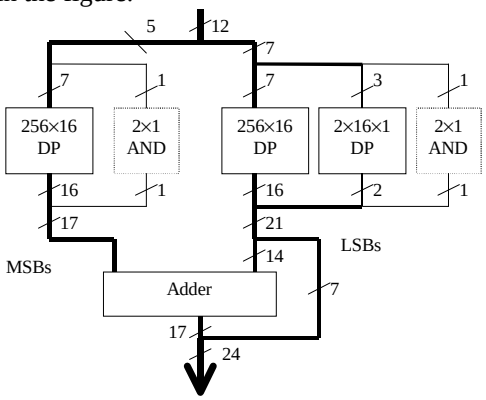

Figure 5. An example of the multiplier for input and coefficient range $0 \div 2^{12}-1$. The optimal architecture for Virtex FPGAs and equivalent cost 1BSR= 20LEs = 5 CLBs

Initially, the architecture of the DKCM does not seem to be much different in caparison with the KCM, only the additional RAM Programming Unit (RPU) and address multiplexing are required (see Figure 2 vs. Figure 1). However, the KCM can be implemented using either the LM or Multiplierless Multiplication (MM) [12] which is carried out using only shifts and additions of the multiplicand. The MM is getting more and more attractive as the coefficient width grows because it employs more efficient optimisation techniques such as Canonic Sign Digit (CSD) [12] and / or Sub-structure Sharing (SS) [13].

Consequently, for Xilinx XC4000, and input and coefficient width greater than 5, the LM consumes on average 25÷50 % more area in comparison with the MM, as it is shown in Figure 6.

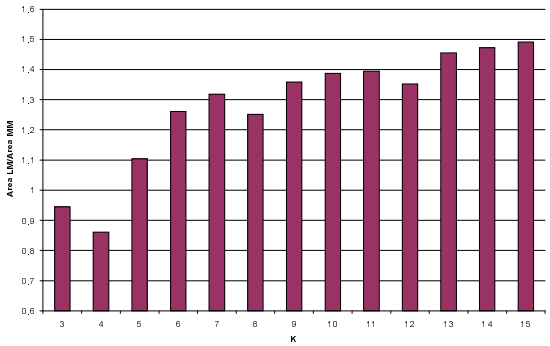

Figure 6. Relation between average area of XC4000 occupied by the LM and MM (for the KCM). Results for the different input width K (input range $0 \div 2^K-1$) and coefficient values $1 \div 2^K-1$

The DKCM in comparison with the KCM can implement a great range of coefficient values, for which, conversely, different KCMs should be developed. Furthermore, a KCM architecture varies significantly for different coefficients, which causes a great difference in area occupied by the KCM. Therefore, to compare the DKCM with the respective KCMs, three different statistical costs of the KCM can be used:

1. Average area occupied by a KCM for a given coefficient range (usually $1 \div 2^K-1$). This value is best suitable for static configurable systems [14], for which the cost of a static KCM and its equivalent static DKCM is compared. The average area of the KCM can also be suitable for dynamic configurable systems [14] for which a great number of KCMs are considered at the time.
2. Maximum area for a given coefficient range – is recommended for dynamic configurable systems, for which the coefficient is changed by FPGA reconfiguration.
3. Maximum area for a given coefficient set - as in point 2, but in the case when the number of possible coefficients is relatively small. This value seems the best for defined designs, however may constrain further changes in the design. This solution however cannot be generalised and therefore is not further referred to.

The VCM is a fully functional multiplier, usually implemented using AND-gates and adders [1], for which a coefficient-change penalty is not observed. However a drawback of the VCM, as can be seen from Figure 7, 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.

According to Figure 7, the DKCM should be implemented for $K \geq 7$. Nevertheless this is the best cost-throughput-relation that in real applications may not be achieved, as the DKCM requires RAMs programming (standstill) cycles which decrease the design throughput and may require design modifications (additional cost). In consequence, two different groups of application can be distinguished:

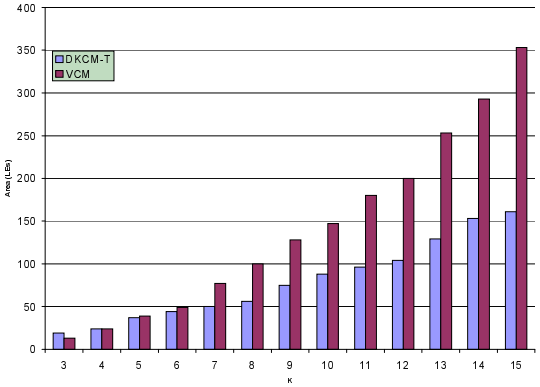

Figure 7. Area XC4000 for the DKCM-T and VCM (different K)

A) Designs without reconfiguration overheads - the change of the coefficient occurs very seldom and / or does not disrupt the system work. For example, a real time image processing system for which the change of the coefficient is carried out during image blank time, or so seldom that it is almost invisible. In this case, the DKCM can be implemented without additional hardware overheads. Furthermore, the KCM and a dynamic reconfiguration system instead of DKCM should be considered to allow additional savings.

It should be noted that for the DKCM-D (dual port DKCM) and for adaptive systems where the difference between the present and new coefficient is very slight, the product obtained while RAM programming is usually only slightly corrupted. Accordingly, the product is in the range of the product calculated for the previous and current coefficient; unless a simultaneous read and write occurs to the same memory address. In the latest case, the output value depends on specifications of the dual port RAM, e.g. for XC4000 16×1 DP-RAMs a simultaneous read and write access is allowed and the output value defined [4].

B) Designs with reconfiguration overheads - the coefficient changes frequently or its change disrupts the system work. In this case, four different approaches can be implemented:

- DKCM-P - two parallel RAMs sets and additional multiplexers are used [9], which allows one RAM set to be programmed while another is operating and vice-versa.
- DKCM-D - as described in the point A (output data can be slightly corrupted!), but architectural overheads are considered as the DP-RAM solution is usually less hardware-efficient than the SP-RAM counterpart.
- VCM, which has no coefficient change penalty.

- DKCM- the multiplier for which operational process is stopped when RAM programming.

To qualify the benefits from using the DKCM reconfiguration approach, let define a functional density, *D* [5, 10]:

$$D = \frac{1}{A \cdot T} \quad (4)$$

The functional density for the VCM, DKCM-P or DKCM-D, and DKCM respectively, can be given as follows:

$$D_V = \frac{1}{A_V \cdot T_V} \quad (5)$$

$$D_D = \frac{1}{A_D \cdot T_D \cdot (1 + \frac{r}{n})} \quad (6)$$

where: D_V, A_V, T_V, A_D, T_D – functional density, area and critical delay for the VCM (V) and DKCM (D) respectively; r- number of reconfiguration cycles; n- number of execution cycles between two subsequent reconfigurations.

For the DKCM, a reconfiguration penalty factor, *r/n*, has been introduced. The penalty can be decreased either by the increase of *n* - the number of execution cycles between two subsequent reconfigurations; or by a decrease of the number of reconfiguration cycles, *r*.

5. Conclusions

The multiplication is a very common operation and as it has been shown in this paper, the proper choice of its architecture is a difficult task. For ASICs, two choices: the VCR and KCM with a fixed, easily defined applications to be used in, can be distinguished. However, for FPGAs the boarder between these two solutions cannot be smoothly defined as a FPGA can be quickly reconfigured. Therefore, implementation of the KCM instead of the VCM is strongly recommended as the KCM occupies 17÷23% on average or 29÷41% on maximum, area of the VCM depending on the multiplier width. Furthermore, design's lower area causes usually shorter propagation time and consequently a significant increase in design functionality, *D.* Conversely, a change of the KCM coefficient has a penalty of operation standstill time which decreases design functionality according to eq. 6. Consequently, to decrease the idle, reconfiguration time there is a tendency to use a partial reconfiguration for which only the multiplier circuit is reconfigured.

The reconfiguration time can be further decreased by in-circuit reconfiguration, i.e. by the use of the DKCM. The DKCM offers much quicker reconfiguration but occupies more area in comparison with the KCM, therefore is a middle-way solution between the VCM and KCM. The DKCM requires additional RAM Programming Unit (RPU) which significantly influences the DKCM cost for small multiplier widths. An alternative solution is programming RAMs using a FPGA partial reconfiguration instead of the RPU but this may not be accepted by a given FPGA, requires longer reconfiguration time and pre-calculated RAM contents.

For adaptive signal processing, the DKCM solution is even more attractive as the DKCM-D has no reconfiguration penalty provided that the product can be slightly corrupted and the number of execution cycles is greater than memory size. Furthermore, the process of changing coefficient for the KCM requires not only FPGA reconfiguration but also redesigning and rerouting of the KCM which consumes significant amount of time (about 1min÷1hour) and therefore this solution is usually unacceptable for adaptive systems.

In this paper the multiplication in FPGAs has been thoroughly studies, however general conclusions have not been drawn as the optimal architecture depends strongly on given FPGAs and design specifications. However, possible architectures and their implementation results have been presented, and the Multiplier Automated Tool has been developed, which may significantly speed-up the implementation of a multiplier.

6. References

[1] Brooks F.P., Plaisted D.A., *Computer Arithmetic Systems*, Prentice Hall 1994.

[2] Chapman K., *Constant Coefficient Multipliers for the XC4000E. Xilinx App. Note*, XAPP 054 December 1996.

[3] Petersen R., Hutchings B.L., *An Assessment of the Suitability of FPGA-Based Systems for Use in Digital Signal Processing*, Proc. 5th Int. Workshop on Field Programmable Logic and Applications, Oxford, pp. 293-302, August 1995.

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

[5] Wirthlin M.J., Hutchings B.L., *Improving Functional Density Through Run-Time Constant Propagation*, ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays, pp. 86-92, 1997.

[6] Xilinx Co., *Core Generator*, Foundation 2.1, 1999.

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

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

[9] 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.

[10] Wojko M., ElGindy H., *Self Configuring Binary Multipliers for LUT addressable FPGAs*, 5th Australasian Conf. on Parallel and Real-Time Systems. University of Adelaide, Australia, 28-29th September 1998, pp. 201-212

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

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

[13] 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.

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