Trade-offs and other challenges in CMOS implementation of parallel FIR filters

Katarzyna Kubiak[1,3] and Rafał Długosz[2,3]
[1] Adam Mickiewicz University
Faculty of Mathematics and Computer Science
ul. Umultowska 87, 61-614 Poznan, Poland
[2] UTP University of Science and Technology
Faculty of Telecommunication, Computer Science and Electrical Engineering
ul. Kaliskiego 7, 85-796, Bydgoszcz, Poland
E-mail: rafal.dlugosz@gmail.com
[3] Aptiv Services Poland
ul. Podgórki Tynieckie 2, 30-399, Kraków, Poland

***Abstract*—The paper presents methods of implementing finite impulse response (FIR) filters in hardware. Considering their functioning, FIR filters require only simple arithmetic operations such as multiplication, addition and shifting of signal samples in the delay line. In the case of their implementation as a device in which parallel processing of signals is assumed, one of the main challenges is an efficient implementation of the block of filter coefficients. This applies in particular to high order, N filters and a relatively high resolution (in bits) of the processed signals and filter coefficients. When the objective is to reach a high filter selectivity, understood as the attenuation in the stop band and the steepness in the transient band of the frequency response, the filter coefficients usually also require high resolutions. In the presented work we perform an analysis of trade-offs between such parameters as data rate, occupied chip area (number of transistors), simplicity of the control block as well as energy consumption. We investigate possible approaches to the implementation of such filters, particularly in terms of the complexity of the block of coefficients. One of the key possibilities is the use of at least partially asynchronous signal processing, which has a significant impact on the complexity of the circuit structure and data rates.**

***Keywords*—FIR filters, Parallel Implementation, Asynchronous operation, CMOS Realization, Low power**

I. Introduction

Filtering belongs to one of the most frequent operations performed in various software and electronics systems. Filters may be used to process either continuous or discrete time signals, analog or digital (quantized). Depending on the signal, the filters may be realized in different ways [2]. The realization approach may additionally depend on such parameters as the cut–off frequency, the steepness of the transient band, the attenuation in the stopband, etc. Additionally, selection of the type of the filter and its implementation depends on the target application. In this context it is worth to mention such parameters as the sampling rate, immunity to noise and external conditions (e.g. environment temperature), etc.

In the case of discrete time signals, finite impulse response (FIR) filters are most commonly used. Their main advantages include stability, as well as the possibility to obtain a linear phase response. They offer a relatively simple structure, composed of a delay line (delay elements usually denoted as T), filter coefficients (h_i) and a summing circuit. The filter order N is equal to the number of delay elements.

When an input signal x is passing the filter, its particular samples x_i are being stored in subsequent memory cells of the delay line. The samples after the multiplication by their corresponding filter coefficients h_i are summed in the output block. The output samples y_n of the filter are calculated as follows:

$$y_n = \sum_{i=0}^{N} h_i \cdot x_{n-i}. \tag{1}$$

The implementation method results from various initial premises. Investigations related to the hardware implementation of such filters may focus on simplifying their overall structure (memory, coefficients, summing circuit) or towards reducing the consumed energy, simplifying the structure of the controlling clock circuit, etc. Such filters are usually realized in digital signal processors (DSP), field programmable gate arrays (FPGA) [1] or as components of typical software systems. Software realizations, especially those in which floating point numbers may be used, allow to reach very high attenuations and large selectivities. These parameters depend on the precision of the implementation of the filter coefficients.

In hardware, obtaining the parameters comparable with those that are achievable in software floating point realizations, mentioned above, is not easy. In digital approach, the main problem are trade-offs between the precision in the realization of the filter coefficients, circuit complexity, dissipated power and data rate. The higher the required precision is, i.e. the number of bits in particular coefficients k, the larger memory block in delay lines, more complex multipliers and summing blocks have to be used. These problems become noticeable in fully parallel realizations, in which each coefficient requires its own multiplier.

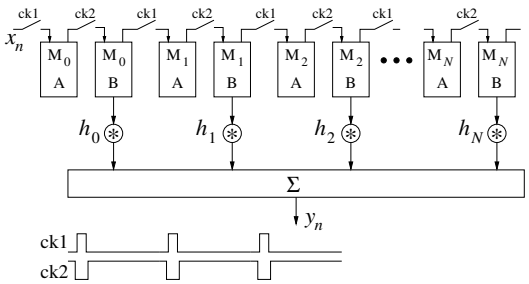

Fig. 1. Conventional approach to the realization of the FIR filter.

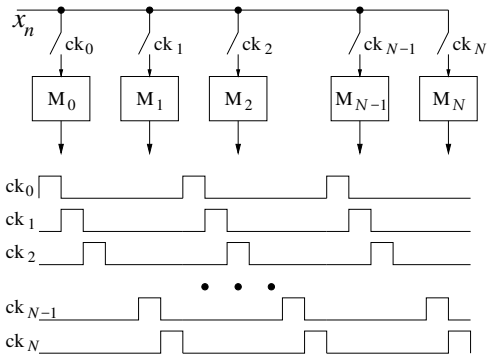

Fig. 2. Delay line realized as a rotational memory, without rewritting signal samples.

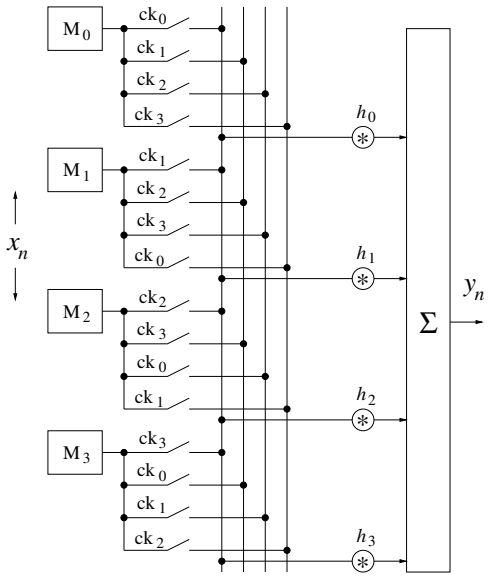

Fig. 3. Rotator FIR filter with rotation switch that couples outputs of the delay line with filter coefficients.

In analog approach, on the other hand, various circuit non-idealities may appear. For example, in switched capacitor (SC) approach, parasitic capacitances of connecting lines impact the precision of the filter coefficients, thus reducing the filter selectivity [3]. In switched current (SI) filters, the gain of current mirrors used in the filter coefficients leads to a similar effect [4]. Additionally, in analog filters more effort is needed to make such circuits immune to the impact of such factors as transistor mismatch, environment temperature, supply voltage, variations in process parameters, leakage, etc.

The focus of this work is transistor level implementation of filters working in parallel [6]. We analyze the circuit complexity and other parameters for two possible architectures of the FIR filters. In the first one, the filter is based on the conventional approach, in which the samples are shifted along the delay line after each new sample is being provided to the filter input. In this approach, the filter coefficients are stored all the time in the same place. In the second approach, the filter coefficients circulate in the memory block, while particular signal samples remain in the same memory cell, until they are overwritten by new samples.

The paper is organized as follows. The next section provides a discussion on selected implementation issues relevant to the realization of FIR filters in hardware, i.e. the complexity of the controlling clock and the multiplication block, which is the basis for the implementation of a single filter coefficient. In the following section, the block of filter coefficients treated as a whole is analyzed in more detail. Optimization of single coefficients requires that the overall filter structure, which results from its frequency, is taken into account. The conclusions are provided in the last section.

II. Implementation issues in parallel FIR filters

Regardless of the approach to the implementation of the FIR filters, the necessity of shifting signal samples along the delay line should be taken into account. It can be achieved by rewriting a sample between memory cells in the delay line or by changing the connections between particular memory cells and the block of the filter coefficients. Two general cases with variations can be considered here.

In the first approach, which we call the classic approach, the memory is realized as a shift register, in which all samples are rewritten between cells for each new signal sample [5]. In this case, the filter coefficients are stored in the corresponding memory blocks, permanently connected to particular memory cells. A block diagram of such a solution is shown in Figure 1. In this way, particular samples are multiplied by successive filter coefficients in subsequent clock cycles, according to 1.

In the second approach, the delay line is implemented as a rotational memory block, shown in Figure 2. In this case, a recorded signal sample remains in a given k-bit memory cell until it is overwritten by one of the subsequent samples after N clock phases. An issue in this approach is the way of implementing the block of filter coefficients and more specifically, the way of connecting particular outputs from the delay line with different coefficients over time. This can be achieved in several ways. One of them is to use a rotation switch which connects particular outputs of the delay line with different coefficients, as shown in Figure 3. Another solution is to permanently connect particular outputs from the delay line with fixed memory blocks containing coefficients and then to rotationally connect the coefficients between these blocks, as shown in Figure 4.

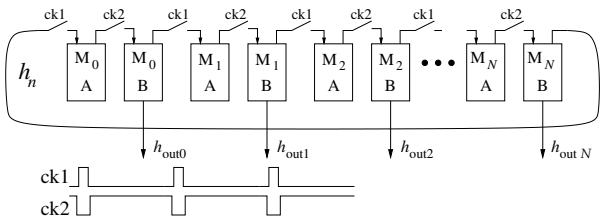

Fig. 4. Circular memory block in which filter coefficients are rewritten between memory cells.

A. Complexity of the clock circuit

One of the important blocks used in filters is the control clock circuit. Its complexity depends on the method of implementing the overall filter, as briefly described above.

In the classic approach mentioned above, a simple, only 2-phase clock generator can be used to control the overall filter. In this case, the complexity of the clock depends largely on the way of implementing the memory cells in the delay line and the filter coefficients. More specifically, an issue is the way of implementing the blocks which multiply the signal by the filter coefficients. They can be realized either as an asynchronous binary tree or as a shift-and-add circuit and an accumulator (ACU). In the first variant, if the output summing circuit of the filter is also realized as a parallel and asynchronous block, then the clock is required only to control the delay line. In the second case, the multiplication operations require an additional k-bit clock generator. We discuss it later in this work.

If each delay element is equipped with two memory cells in the classic approach, then rewriting of the samples can be done fully in parallel and be controlled by a 2-phase clock. It is a two-step process. In the first clock phase, all samples are stored in temporary memory cells, freeing space for further samples. In the second clock phase, the values stored in the temporary cells are shifted to subsequent delay elements. Since the rewritten samples are in practice immediately available at the outputs of the delay line, they can be immediately multiplied by the filter coefficients and summed at the output of the filter in the same clock phase.

In the second approach, with rotational memory, a multi-phase clock circuit with the number of phases equal to $N+1$ is required, as shown earlier in Figure 2. The memory block with filter coefficients has a similar structure to the one presented in the classic approach, with the only difference being the way the samples are fed from the delay line.

B. Filter coefficients – multiplication operation

The way of implementing the coefficients of the FIR filters (the same problem exists in the case of infinite impulse response IIR filters) is one of the basic problems investigated in the literature [7]. As discussed earlier, a lot of work in this area is aimed at optimizing the multiplication operations in order to minimize the number of used transistors and reduce the energy consumed by the filter. The problem becomes relevant in filters with high orders, N, and high resolutions (in bits) of the processed signal samples and the coefficients themselves.

In the vast majority of digital realizations of filters, the multiplication operation is based on the shift-and-add principle, as mentioned above. This principle, however, may be applied in several ways. A commonly used approach is the iterative (serial) one, which involves the ACU and the k-phase clock generator. One of two multiplied numbers is shifted by one bit to the left (multiplication by 2) in subsequent clock cycles. After each operation, the resultant signal is added (or not) to the ACU throughout a set of AND gates, depending on the values of corresponding bits in the second number. The ACU block is realized as the multi-bit full adder (MBFA), composed of 1-bit full adders (1BFA), coupled by their carry in/out terminals. The MBFA is accompanied by a memory block, controlled by a clock.

For the purposes of the considerations presented in this work, we treat the 1BFA and MBFA circuits as standard blocks. Many such systems have been proposed in the literature [8], [9], [10]

In the serial approach, particular bits of the first input number are provided to one of the inputs of corresponding AND gates. The second inputs of all these gates are shorted together and switched between following bits of the second multiplied number in subsequent clock phases. The advantage of this solution is its relatively low complexity, which increases only moderately with the resolution of the multiplied numbers. The cost of this feature is relatively low speed, linearly proportional to the signal resolution.

Another approach involves the implementation of the multiplier as an asynchronous binary tree composed of MBFAs of different lengths. Such circuits are very fast and easy to control, as they do not require a clock block. However, they require a large number of transistors. Their application in fully parallel implementations of filters is in practice limited to relatively small signal resolutions and relatively small values of N.

To assess the complexity of both described approaches let us consider the multiplication of two numbers, k– and l– bit, respectively.

In the case of the serial approach only one MBFA is used. It is able to accommodate the sum of l intermediate signals which are computed from the initial k-bit signal by following shifting operations. In this case the number of 1BFAs, which is equal to the number of the used AND gates and to the resolution of the output signal of the ACU, is given as follows:

$$R^{\mathrm{S}}_{1\mathrm{BFA}} = k + l. \tag{2}$$

The number of transistors used in such ACU and thus in a single filter coefficient is expressed as follows:

$$R^{\mathrm{S}}_{\mathrm{TR}} = (30 + 6 + 8) \cdot R^{\mathrm{S}}_{1\mathrm{BFA}}, \tag{3}$$

where factors 30, 6 and 8 refer to the number of transistors in a single 1BFA, a single AND gate and a single 2-stage memory cell in the ACU block, respectively.

In the parallel approach the number of 1BFAs is expressed in a more complex was, as follows:

$$R^{\mathrm{P}}_{1\mathrm{BFA}} = k \cdot (2^L - 1) + (L - 0.5) \cdot 2^L, \tag{4}$$

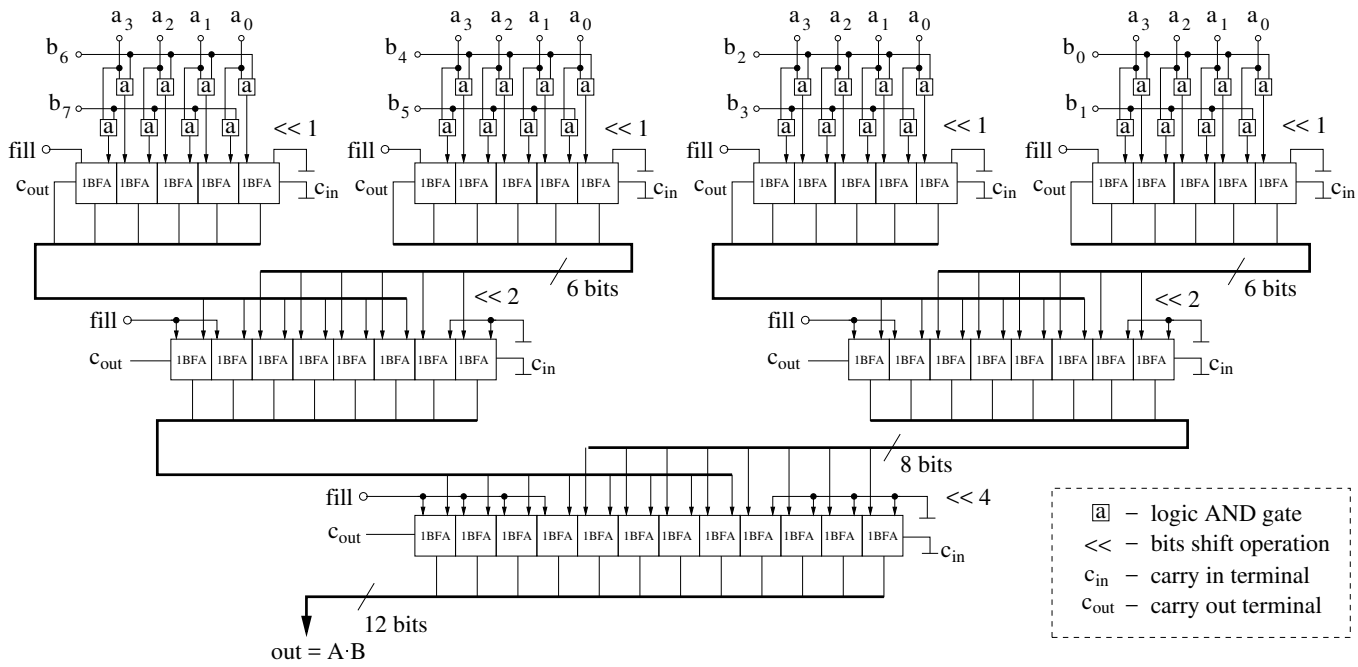

Fig. 5. Binary-tree fully parallel and asynchronous multiplier for and example case of two numbers: 4 and 8 bit numbers (A and B, respectively).

where L is the number of layers in the tree. Assuming that l is one of the powers of 2 (e.g. 2, 4, 8, 16, ...), it is expressed as:

$$L = \log_2 l. \tag{5}$$

In the parallel approach the number of the AND gates at the inputs of the tree is equal to:

$$R^{\mathrm{P}}_{\mathrm{AND}} = l \cdot k. \tag{6}$$

Combining the formulas for the parallel approach, the total number of transistors is expressed as follows:

$$R^{\mathrm{P}}_{\mathrm{TR}} = 30 \cdot R^{\mathrm{P}}_{\mathrm{1BFA}} + 6 \cdot R^{\mathrm{P}}_{\mathrm{AND}}. \tag{7}$$

Formula 4 may require an explanation. Figure 5 shows an exemplary multiplication circuit based on the asynchronous binary tree, working fully in parallel. It is shown for two exemplary numbers A and B with resolutions of 4 and 8 bits, respectively ($k = 4$, $l = 8$). In this configuration, the number of layers in the tree equals 3 according to formula 5. In the first layer four MBFAs are used, each consisting of five 1BFAs. These circuits, however, provide 6-bit numbers on their outputs. It results from the fact that the most significant bits (MSB) of the output signals are in this case carry out signals from most significant 1BFAs. Assuming that A and B have maximum possible values of 15 (1111) and 255 (11111111), respectively, the output of each MBFA in the first layer is equal to 45 (15 + 2· 15), i.e. 101101 binary.

In the second layer of the tree two 8-bit MBFAs are used. The signal is provided with the shift of two bits to the left, i.e. multiplied by 4. As a result, for the maximum values of A and B the output of each 8-bit MBFAs is equal to 225 (45 + 4· 45). It is worth to notice that carry out bits are in this case 0, so they can not be used to extend the resultant number. In the 3$^{\mathrm{rd}}$ (last) layer of the tree a single MBFA is used, with one of its inputs shifted by 4 bits to the left, i.e. multiplied by 16. The resulting output signal is equal to 3825 (225 + 16· 225).

Increasing the resolution of the number B to 16 or 32 bits makes it necessary to use a binary tree with 4 or 5 layers, respectively, with one of its inputs shifted by 8 or 16 positions. It can be shown that excluding the first layer of the tree, in which a shorter chain of 1BFAs can be used, the lengths of the used MBFAs in the following layers increase by 4, 8, 16, ... , in relation to the value of k.

The circuit shown in Fig. 5 can multiply either both positive numbers (for the 'fill' signal equal to 0) or a positive B number and a negative A one (for 'fill' = 1). The input signal x may be shifted by a DC value to avoid negative numbers. However, in the FIR filters the coefficients may be either positive or negative numbers. In this situation the filter coefficients are provided to the A input (thus k becomes the resolution of the filter coefficient), while the signal samples to the B input. In the two's complement code, the MSB is the sign of the number ('1' for negative) and thus can be directly used as the fill signal.

The chip area depends on the technology. We provide an estimate for the CMOS 130 nm technology on the basis of a realized project. A prototype 8-bit ACU in the CMOS 130 nm technology was implemented, as shown in Fig. 6. The circuit was verified by means of laboratory tests. The prototype circuit is composed of eight 1BFAs, accompanied by a memory block realized as a chain of switches and short-term memory cells. The cells are in turn realized as parasitic capacitances at the gate terminals of the NOT logic gates. A single 1BFA is

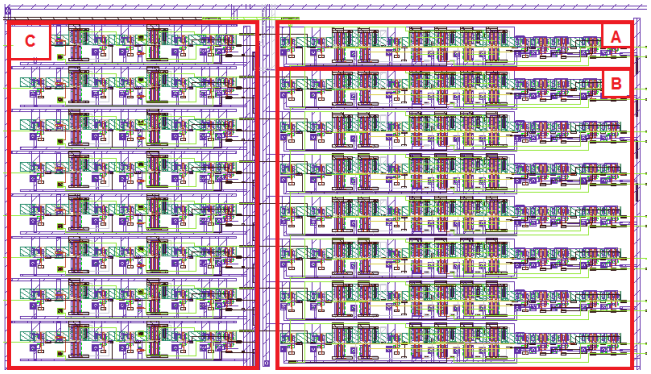

Fig. 6. Accumulator used in the multiplier: (a) 1BFA, (b) 8-bit MBFA, (c) 8-bit memory block composed of switches and memory cells realized as parasitic capacitance of NOT gates.

composed of 30 transistors and occupies an area of 45×5 μm (225 μm^2). This value may be used in an assessment of the overall chip area.

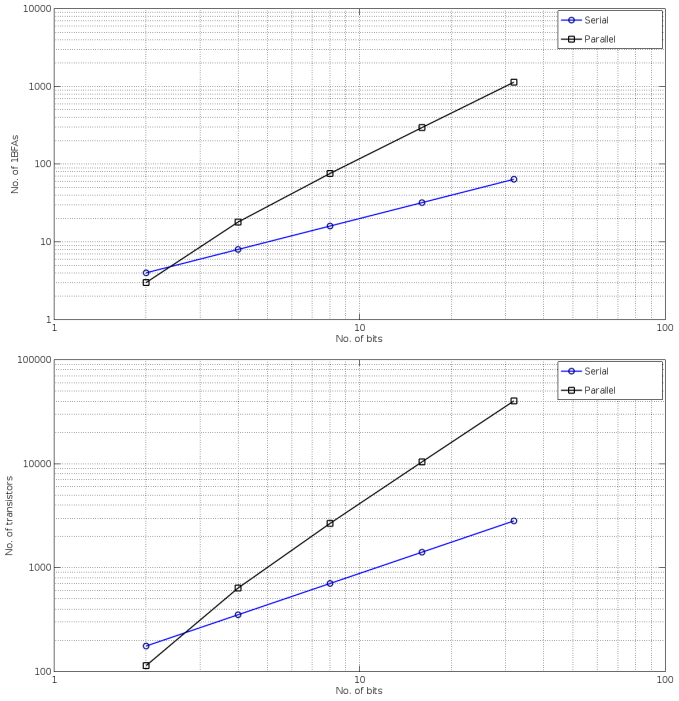

Fig. 7. Complexity of a single filter coefficient for serial and parallel approach: (a) the total number of 1BFAs, (b) the total number of transistors (approximately).

III. Block of filter coefficients – trade offs in transistor-level implementation

Similarly to the multiplication operation, the overall block of the filter coefficients may be implemented either as a serial or a parallel circuit. It is also possible to propose various mixed approaches [1], in which synchronous and asynchronous approaches can be applied at different states of the filtering process depending on the value of N and k. We assessed the complexity of the filter block in different

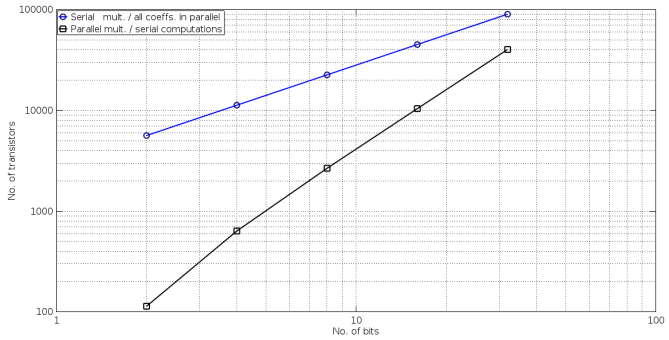

Fig. 8. Complexity of the overall block of filter coefficients for an exemplary case of $N = 31$, different resolutions of signal samples and coefficients. The results were obtained for 32 serial multipliers working in parallel and a single parallel and asynchronous multiplier (32 multiplications in series).

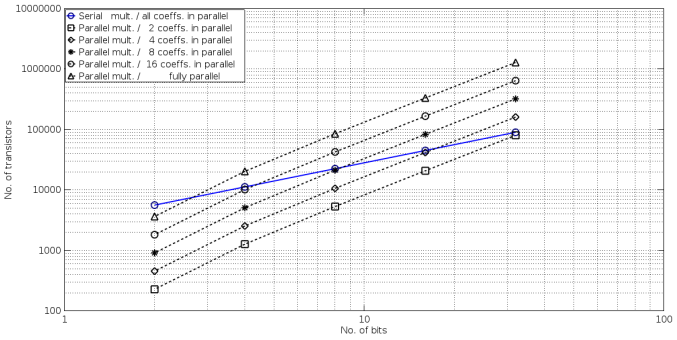

Fig. 9. Similar results as in Fig. 8. In the case of a parallel multiplier the results are presented for different number of multiplication blocks working in parallel.

scenarios. Selected results are shown in Figs. 8 and 9 for an exemplary filter with 32 coefficients.

Initially, we compared two cases, as shown in Fig. 8. The first one involves a single fully parallel (asynchronous) multiplier. This block is switched between following samples stored in the delay line and their corresponding coefficients. In the second case 32 serial multipliers are used. In both cases a similar 32-phases clock is used, however in a different way. The data rate is similar in both cases.

Then, we additionally compared the results for multipliers realized as parallel and asynchronous circuits and several such blocks working in parallel (2, 4, 8, 16, 32 (fully parallel)), as shown in Fig. 9. The results are shown as a function of the resolution of the filter coefficients and the signal samples in all cases.

The obtained results reveal some interesting relations. A single parallel multiplier is more complex than a serial one, although the ratio between the numbers of transistors strongly depends on the signal / coefficient resolution in both cases. The larger the resolution is, the more the ratio (parallel to serial) increases in favor of the serial approach. However, when comparing the complexity of the overall block of the filter coefficients the conclusions are different. The results shown in Fig. 8 suggest that when the moderate data rate is acceptable, a better option is to use a single fully parallel

multiplier and multiplex it, instead of using $N+1$ (filter length) serial multipliers working in parallel.

Fig. 9 shows that for a similar number of used transistors, the filter with fully parallel and asynchronous multipliers offer a better performance, however it depends on the signal resolution. For smaller resolutions (8 bits) a similar circuit complexity is obtained for a filter that is even 8 times faster than the one with serial multipliers. For 16 bits of the resolution a similar number of transistors is obtained for a filter that is 4 times faster. Even for 32 bits of the resolution, the filter with parallel coefficients can be 2 times faster than the one with serial coefficients.

Basing on the obtained results, we make following suggestions for when particular filters fit the best. Let us consider four rough cases, as listed and described below. We consider two parameters, such as filter order N and the resolution of the filter coefficients in terms of the number of bits k.

Low filter order N, small resolution k

In this case the optimal approach is a full parallel one, in which both the multipliers and the overall block of the filter coefficients operate in parallel and asynchronously.

Low filter order N, large resolution k

Here parallel multipliers will offer higher performance because the number of necessary clock phases will be smaller than in the case of the filter with serial multipliers. Let us denote this case as $N < k$.

Large filter order N, large resolution k

If $(N + 1) = k$ or both parameters have similar values, a filter with a single parallel multiplier will offer the best performance. If $N >> k$ several asynchronous multipliers working in parallel can be used, with each of them being switched between a given subset of signal samples and the filter coefficients.

Large filter order N, small resolution k

This case may be considered as a special one, not relevant for investigations presented in this work. Filters with larger N are usually introduced to enlarge the selectivity of the filter. Suppressing the resolution of the coefficients is in opposition to that objective. In several applications (e.g. decimation filters for Σ-Δ modulators) longer filters with equal coefficients are used, however such filters can be realized in a much simpler way, not involving the multiplication operations [11].

IV. Conclusions

The paper presents investigations of hardware implementation of Finite Impulse Response filters in digital specialized integrated circuits. In particular, we focused on the implementation of the filter coefficient block. Depending on the frequency response of the filter, this block is the main source of typically high hardware complexity of the overall filter.

Various possibilities of implementing the block of coefficients were analyzed, in which both sequential and fully parallel and asynchronous multiplication circuits were used. Computations were carried out for different resolutions of the processed signals and the filter coefficients.

The conclusion is as follows: the parallel / asynchronous approach leads to satisfactory results regarding the obtained data rates at a given number of transistors. In the case of a moderate filter length, comparable to the resolution of the processed signals, fully parallel multiplication of particular signal samples by corresponding filter coefficients requires a larger number of transistors than in the case of a single parallel multiplier switched between particular coefficient–sample pairs. For longer filters, several asynchronous multipliers can be used in parallel.

A next step in our investigations will be solving the issue of reducing the number of bits at the output of the filter. This can be achieved by the operation of shifting all bits to the right. Bitwise shifting can be applied either directly at the output of each multiplier or at the output of the overall filter, i.e. after summing up all multiplication products.

References

[1] Britto Pari J., Vaithiyanathan D., "An Efficient Multichannel FIR Filter Architecture for FPGA and ASIC Realizations", *International Journal of Applied Engineering Research*, vol. 12, No. 10, pp. 2209-2220, 2017.

[2] Dąbrowski A., *Multirate and Multiphase Switched-capacitor Circuits*, Chapman & Hall, ISBN-13: 978-0412724909, ISBN-10: 0412724901, London, 1997.

[3] Dąbrowski D., Długosz R., Pawłowski P., "Integrated CMOS GSM Baseband Channel Selecting Filters Realized Using Switched Capacitor Finite Impulse Response Technique", *Microelectronics Reliability Journal*, Elsevier, Vol. 46, No. 5-6, pp. 949-958, Jun 2006.

[4] Długosz R., "Ultra Low Power Switched Current Finite Impulse Response Filter Banks Realized in CMOS 0.18μm technology", *SPIE International Symposium on Microtechnologies for the New Millennium*, Gran Canaria, Spain, May 2007, Proc. SPIE, Vol. 6590, pp.65900H; DOI:10.1117/12.721162, May 2007.

[5] Seok-Jae Lee, Ji-Woong Choi, Seon Wook Kim, Member, and Jongsun Park, "A Reconfigurable FIR Filter Architecture to Trade Off Filter Performance for Dynamic Power Consumption", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 19, No. 12, pp. 2221-2228, Dec 2011.

[6] Gowrikumari Nanireddy, Koteswara Rao Vaddempudi, Dr.M.S.S. Rukmini, "An Area Efficient Fault Tolerant Parallel FIR Filter Design", *Jour of Adv Research in Dynamical & Control Systems*, Vol. 9, No. 3, 2017.

[7] Mehta Shantanu Sheetal, Vigneswaran T., "High Speed and Efficient 4-Tap FIR Filter Design Using Modified ETA and Multipliers", *International Journal of Engineering and Technology (IJET)*, Vol. 6, No. 5, pp. 2159-2170, 2014.

[8] Raushan Kumar, Sahadev Roy, and C.T. Bhunia, "Low-Power High-Speed Double Gate 1-bit Full Adder Cell", International Journal of Electronics and Telecommunications, Vol. 62, No. 4, pp. 329-334, 2016.

[9] S.-M. Kang, Y. Leblebici, "CMOS digital integrated circuits", Tata McGraw-Hill Education, 2003.

[10] R. Shalem, E. John, and E. John, "A novel low power energy recovery full adder cell," *9 th Great Lakes Symposium on VLSI*, Proceedings IEEE, pp. 380-383, 1999.

[11] R. Długosz, T. Talaśka, M. Szulc, P. Śniatała, P. Stadelmann, S. Tanner, P.A. Farine, "A Low Power, Low Chip Area Decimation Filter for Σ-Δ Modulator for Flywheel MEMS Gyro realized in the CMOS 180 nm Technology", *28th International Conference on Microelectronics* (MIEL), Nis, Serbia, pp.411-414, May 2012.