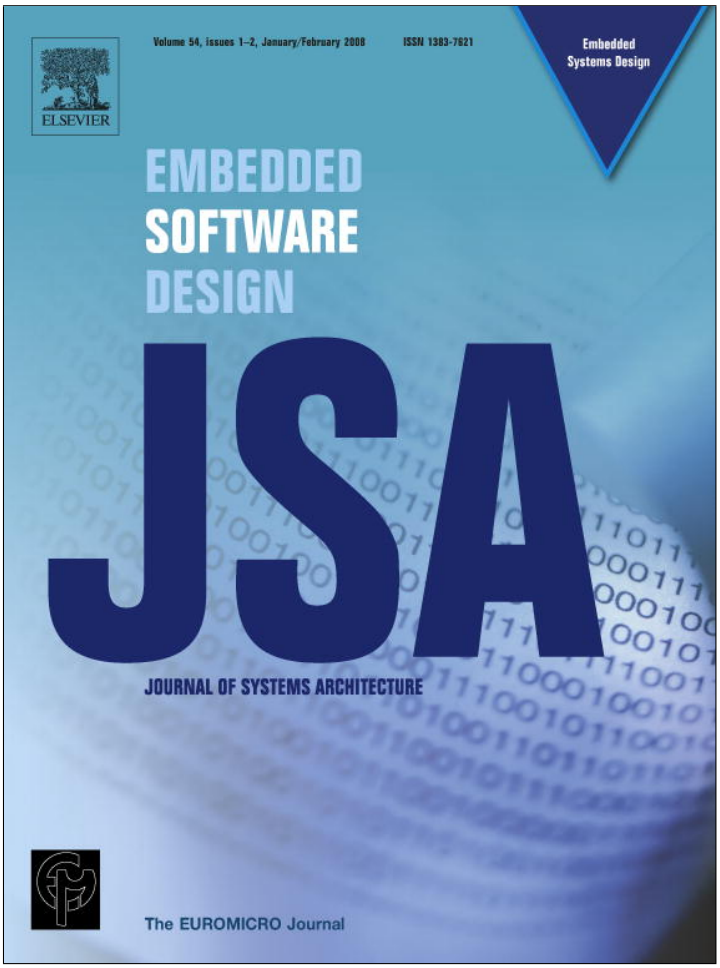

Available online at www.sciencedirect.com
ScienceDirect
Journal of Systems Architecture 54 (2008) 177–196
JOURNAL OF SYSTEMS ARCHITECTURE
www.elsevier.com/locate/sysarc

Configurable folded array for FIR filtering

Vladimir Ciric *, Ivan Milentijevic

Computer Science Department, Faculty of Electronic Engineering, University of Nis, Nis 18000, Serbia

Received 28 July 2005; received in revised form 31 January 2007; accepted 17 May 2007
Available online 5 June 2007

Abstract

The synthesis of configurable bit-plane processing array for FIR filtering is described in this paper. Possibilities for configuration are explored and encompassed by application of folding technique. The proposed folded architecture supports on-the-fly configuration of number of taps and coefficient length. This is achieved by dynamic operations mapping on the different hardware units in array structure. Dynamic operations mapping, involved in application of folding technique, allows recognition of user defined parameters, such as number of coefficients and coefficient length on implemented array size. The architecture provides flexible computations and offers the possibility of increasing the folded system throughput, by reducing the number of operations performed on a single functional unit, at cost of decreasing the coefficient number or length. Effects of folding technique application to architecture configuration capabilities are presented. The configurable folded FIR filter synthesis process is presented in detail. The obtained folded system architecture is described by block diagram, DFG, functional block diagram and the data flow diagram. The method of operation and operations mapping on the processing units are described. The algorithms for data reordering are given. With the aim to illustrate the functionality, configuration capabilities, and ''trade-offs'' relating to occupation of the chip resources and achieved throughputs of synthesized folded architecture, we present results of FPGA prototyping. The proposed configurable folded array is used for H.264/AVC deblocking filter implementation with extremely low-gate count that is achieved at the cost of time, but the design meets the requirement for real-time deblocking in mobile embedded computing platforms.

Keywords: Semi-systolic arrays; FIR filtering; Configurable filtering; Bit-plane; Folding technique; Deblocking filter

1. Introduction

Mobile devices technology is changing rapidly. There is an increasing number of wireless-communications standards, code-division multiple access, and emerging third-generation technologies. However, as wireless technologies mature, service providers differentiate themselves by offering new features, such as multimedia capabilities [1]. Additionally, video coding applications, such as H.264/MPEG-4 Advanced Video Coding, adopt filter called deblocking filter in order to eliminate blocking artifacts and to achieve better coding efficiency [2–4]. The deblocking filter is much more complex than common low-pass FIR filters. The concept of deblocking is first to decide what kind of FIR filter should be involved considering the blurring effects on the currently processed image edges.

* Corresponding author. Tel.: +381 18 529 603; fax: +381 18 588 399.
E-mail address: vciric@elfak.ni.ac.yu (V. Ciric).

1383-7621/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2007.05.001

Then, different types of filters, relating to number of taps, are selected [2–4]. The additional circuitry adds cost, occupies more space, increases power usage in mobile devices, and increases product-design time. This problem can be solved by using configurable architectures. With this approach, computation can be changed on the fly, letting a single architecture perform different computations [1,5]. In configurable FIR filtering, circuitry can be changed on the fly, as software instructions command circuitry control logic to alternate the computation. Reconfiguration can occur within only a few clock cycles [1].

Many different FIR filtering structures exist. Most of them are based on systolic methods [6–10], and provide some trade-off between complexity and throughput. For dedicated applications, the design choice then becomes the minimal complexity structure that can achieve the required throughput rate. Therefore, in order to establish optimal area–time trade-off, a careful choice of circuit design style is necessary. In synthesizing DSP architectures, it is important to minimize the silicon area of the integrated circuits. That is achieved by reducing the number of functional units (such as multipliers and adders), registers, multiplexers, and interconnection wires. The folding transformation is used to systematically determine the control circuits in DSP architectures where multiple algorithm operations are time multiplexed to a single functional unit. By executing multiple algorithm operations on a single functional unit, the number of functional units in the implementation is reduced, resulting in an integrated circuit with low-silicon area [11].

The goal of this paper is the synthesis of folded FIR filter that is capable of on-the-fly configuration of number of taps and coefficient length in fixed array structure. As a starting architecture for the synthesis of the folded bit-plane FIR filter architecture with changeable folding sets, well-known bit-plane architecture (BPA) [5,12] is used [13–15]. The crucial novelty is that folding technique is used to obtain runtime control over functional units of configurable architecture that will enable on-the-fly change of the set of operations performed on a single functional unit. In other words, the different operations can be mapped on the different hardware units in a fixed array structure. The proposed architecture should enable FIR filtering on toroid bit-plane processing array, where the number of taps and coefficient length could be configured. The derived architecture should serve as a platform for implementation of the deblocking filter. It provides flexible computations and offers the possibility of increasing folded system throughput, by reducing the number of operations performed on single functional unit, when the system performs the computation with a reduced number of taps or coefficient length.

This paper deals with the effects of operation assignment to folded architecture configuration capabilities. Data flow of folded FIR filter with changeable folding factor will be presented. The synthesis process will be presented in detail, as well as the solution for the retiming of source BPA DFG that enables the synthesis of configurable fixed size folded array. The obtained architecture, based on new folding set assignment, will be described with a block diagram, DFG, functional block diagram and the data flow diagram, as well as the method of operation and operations mapping on the processing units. The reordering of coefficient bits, inherited from operations mapping, will be extracted from the array and implemented by proposed reordering algorithm in separate module, keeping input and output data order unchanged, as well as the array regularity. The algorithm for coefficient bit reordering will be presented, as well as a hardware implementation for the reordering unit. In order to illustrate the functionality and performances of synthesized folded architecture, we give implementation results of folded FIR filter architecture with changeable number of coefficients and coefficient length. The design ''trade-offs'' relating to the chip resources occupation and achieved throughputs are presented. The configuration abilities are demonstrated on one representative fixed size folded array. The proposed architecture is used for H.264/AVC deblocking filter design in embedded mobile computing devices. Deblocking method with 5 different FIR filter modes [3] is successfully implemented by the proposed architecture.

The paper is organized as follows. Section 2 gives background for the application of folding technique; Section 3 is devoted to bit-plane architecture as a basis for synthesis process; Section 4 presents transposed folded bit-plan FIR filter with folded bit multiplications and changeable folding factor; Section 5 is the main section and contains analysis of dependencies between configuration abilities and folding set assignment, using bit-plane FIR filter with folded bit multiplications, and synthesis of folded architecture with changeable number of coefficients, as well as the description of new assignment

of folding sets and the solution for the retiming procedure. Functional description of the synthesized architecture is presented, too; Section 6 is devoted to the FPGA implementation of the folded FIR filter architecture; Section 7 stands for H.264/AVC deblocking filter implementation based on the proposed configurable folded architecture, while in Section 8 concluding remarks are given.

2. Folding technique

With aim to clarify the application of folding technique to the BPA we give a brief review of folding transformation.

Folding or time-multiplexing is a technique for efficient resource sharing for area-constrained synthesis from a data-flow graph (DFG). Folded architecture is an architecture that executes multiple algorithm operations on single functional unit. The algorithm operations are executed on the reduced number of functional units at the cost of time. Thus, the folding enables a fine tuning of area–time (AT) properties by reducing the number of functional units in applications where throughput can be traded-off for silicon area.

Folding technique is a technique, introduced by Parhi [11], which gives the answer whether the architecture is foldable or not, and allows calculation of folded data path delays and timings, in cases when folding is possible. The folding technique can be well exploited for systematic achieving of throughput requirements on restricted silicon area. The basic terms of folding technique are folding set, folding factor and folding order. Folding set (S) is defined as an ordered set of operations, which contains N operations, time-multiplexed on the same functional unit (FU). The number of operations executed by same FU is called a folding factor (N). Folding order (v) of operation H_v is a time instance in which FU of the folded system executes the operation [11].

The synthesis of folded data path is explained in Fig. 1a and b (taken from [11]). Fig. 1a shows unfolded data path, as an edge $U \rightarrow V$ with $w(e)$ delays, while Fig. 1b depicts the corresponding folded data path. Folding technique states that the data, which begin at the functional unit H_u with P_u pipelining stages, pass through

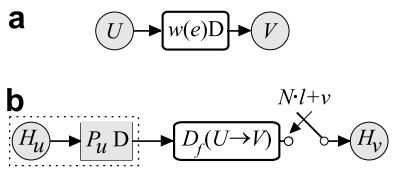

Fig. 1. The synthesis of folded data path. (a) An edge $U \rightarrow V$ with $w(e)$ delays and (b) the corresponding folded data path.

$$D_f(U \rightarrow V) = Nw(e) - P_u + v - u \tag{1}$$

delays, and are switched into the functional unit H_v at the time instances $N \cdot l + v$, where N is a folding factor, while u and v are the folding orders of nodes U and V that satisfy $0 \leqslant u, v \leqslant N-1$.

For a folded system to be realizable, $D_f(U \rightarrow V) \geqslant 0$ must hold for all of the edges in the DFG. Once valid folding sets have been assigned, retiming can be used to satisfy this property or determine that the folding sets are not feasible. Using retiming [11] an edge $U \rightarrow V$ with $w(e)$ delays can be modified, in order to satisfy the condition $D_f(U \rightarrow V) \geqslant 0$, by changing the number of delays as follows:

$$w_r(e) = w(e) + r(V) - r(U), \tag{2}$$

where $w_r(e)$ is the number of delays in edge $U \rightarrow V$ of retimed DFG while $r(X)$ represents the retiming value for node X. In other words if the number of delays of input edge is decreased for value $r(X)$, the number of delays in all output edges should be increased for the same value. Let $D'_f(U \rightarrow V)$ be a number of delays in edge $U \rightarrow V$ in the retimed DFG. The following condition must be satisfied

$$D'_f(U \rightarrow V) \geqslant 0, \quad \text{i.e. } Nw_r(e) - P_u + v - u \geqslant 0. \tag{3}$$

An inequality (3), using Eq. (2) can be rewritten as follows:

$$r(U) - r(V) \leqslant \lfloor D_f(U \rightarrow V)/N \rfloor, \tag{4}$$

where $\lfloor x \rfloor$ is maximal integer less or equal to x. If the solution for system of inequalities exists the DFG can be retimed, and folded.

After this short description of folding technique, let us introduce architecture of bit-plane semi-systolic FIR filter as a source for synthesis of configurable folded FIR filter array.

3. Bit-plane semi-systolic FIR filter

Output words $\{y_i\}$ of FIR filter are computed as

$$y_i = c_0 x_i + c_1 x_{i-1} + \cdots + c_{k-1} x_{i-k+1}, \tag{5}$$

where $c_0, c_1, \ldots c_{k-1}$ are coefficients while $\{x_i\}$ are input words.

Computation (5) can be realized in different manners. When high-performances are required systolic arrays are frequently used. Semi-systolic array share with systolic arrays desirable simplicity and regularity properties in addition to their pipelining and multiprocessing schemes of operation. The only difference is that the broadcasting of data to many PEs in one time step is allowed in semi-systolic arrays, while systolic arrays are restricted to temporal locality of communication. Also, the existence of some additional connections can be allowed for semi-systolic architectures [5,7].

The bit-plane architecture (BPA) is semi-systolic architecture with bit-plane operations that provides regular connections with extensive pipelining and high-computational throughput. The BPA, due to regularity, is taken as a basis for synthesis of folded semi-systolic FIR filter architecture with changeable number of coefficients and coefficient length. In order to explain the BPA following notation is adopted:

m	coefficient word length,
k_C	number of coefficients $(c_0, c_1, \ldots, c_{k_C} - 1)$,
n	input word length,
c_i^j	bit of coefficient c_i (with weight 2^j),

$c_i \equiv c_i^{m-1} c_i^{m-2} \ldots c_i^0$, where $c_i^0 c_i^1 \ldots c_i^{m-1}$ are the bits of coefficient c_i with weights 2^0, $2^1, \ldots, 2^{m-1}$, respectively, $c^j \equiv c_{k-1}^j c_{k-2}^j \ldots c_0^j$, where $c_0^j, c_1^j, \ldots, c_{k-1}^j$, are the bits with weight 2^j of coefficients $c_0, c_1, \ldots, c_{k-1}$, respectively.

The transfer function for the BPA is obtained from (5). Splitting multiplications in respect to coefficient bits, BPA transfer function for $k_C = 3$ and $m = 4$, becomes

$$\begin{aligned} G(z) = z^{-1}&\left(c_0^3 2^3 z^{-9} + z^{-1}\right)\left(c_1^3 2^3 z^{-9} + z^{-1}\right)\left(c_2^3 2^3 z^{-9} + z^{-1}\right) \\ &\left(c_0^2 2^2 z^{-6} + z^{-1}\right)\left(c_1^2 2^2 z^{-6} + z^{-1}\right)\left(c_2^2 2^2 z^{-6} + z^{-1}\right) \\ &\left(c_0^1 2^1 z^{-3} + z^{-1}\right)\left(c_1^1 2^1 z^{-3} + z^{-1}\right)\left(c_2^1 2^1 z^{-3} + z^{-1}\right) \\ &\left(c_0^0 2^0 + z^{-1}\right)\left(c_1^0 2^0 + z^{-1}\right)\left(c_2^0 2^0\right). \end{aligned} \quad (6)$$

The splitted parts of the multiplications become input word times $1 - b$ coefficient multiplications, the partial products. It enables fine-grained pipelining. These are just logical AND function between the input word and coefficient bits. In the first bit-plane the least significant partial products ($c^0 x_0$) of all coefficients are computed and accumulated (Fig. 2, taken from [12]).

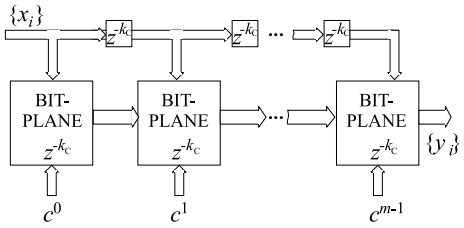

Fig. 2. Architecture of bit-plane FIR filter.

The output of the first bit-plane is shifted by one weight and then the second lowest significant partial products are processed in the second bit-plane and so on [12].

The corresponding DFG for $k_C = 3$ and $m = 4$ is given in Fig. 3, and functional block diagram of bit-plane architecture ($k_C = 3$, $m = 4$, $n = 5$) is shown in Fig. 4 (taken from [12]). The functional block diagram from Fig. 4 is designed to operate with input words $\{x\}$ in 2's complement, and with unsigned coefficients [12]. Simultaneous processing of all LSBs, at the beginning of computation, enables the truncation of one LSB of the intermediate output signal after each bit-plane without any loss of accuracy in more significant weights. Number of coefficients (k_C), coefficient length (m) and input word length (n) are design parameters of the architecture. Architecture consists of $k_C \cdot m$ rows and $m + n + \log_2(k_C)$ columns of basic cells (Fig. 4).

Since architecture of folded system directly depends on method of operations mapping onto the nodes in folded system (folding set assignment), only specific folding set assignment will lead to the solution that enables configuration of number of coefficients. There is no a formal method to find folded set assignment that will lead to an architecture that has preferred feature.

Approach of designing configurable folded architecture with changeable number of coefficients, used in this paper, is to assign folding sets and fold architecture in general form, find dependencies between folding sets and folded architecture parameters,

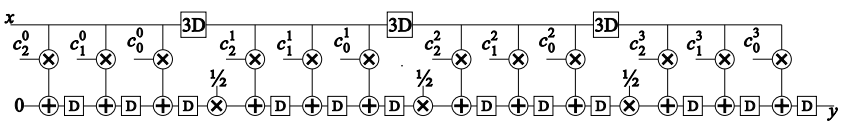

Fig. 3. DFG for the BPA with $k_C = 3$ and $m = 4$.

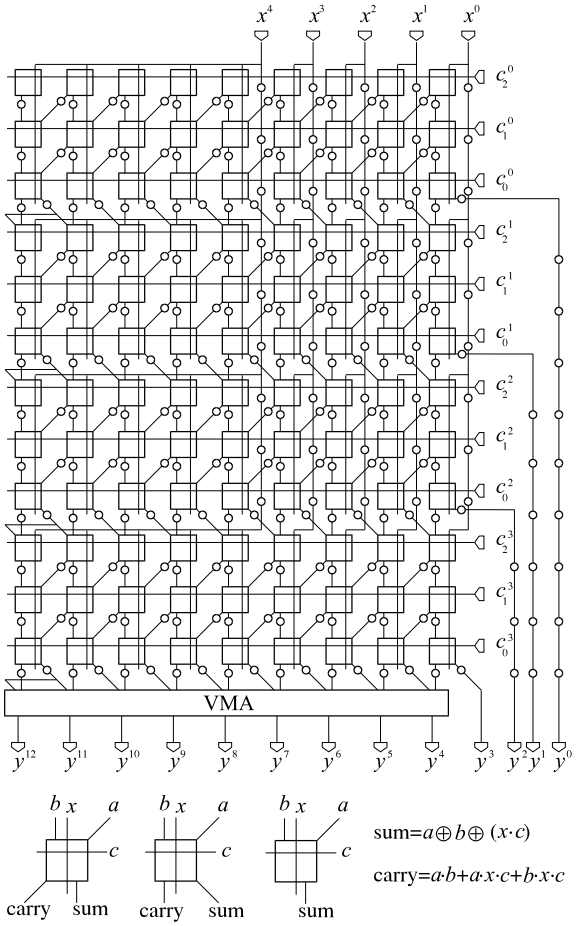

Fig. 4. The BPA for $k_C = 3$ and $m = 4$.

and synthesize control logic that allows configuration of these parameters.

However, different folding set assignments lead to architectures with different possibilities of configuration. Architecture of folded bit-plane FIR filter presented in [13] enables the configuration of folding factor, i.e., coefficient length. We choose this architecture to find dependencies among configuration abilities of folded architecture and folding set assignment, and to adopt folding set assignment with aim to allow the synthesis of folded architecture with changeable number of coefficient.

4. Transposed bit-plane FIR filter with folded bit multiplications

The BPA cannot be transformed into the folded bit-plane architecture by direct application of folding technique. It is obvious, from Figs. 3 and 4, that multiplications of coefficients and input words cannot be recognized as operations, i.e., nodes in the DFG, because the algorithm is based on the resorting of partial products. It implies that it is necessary to apply the folding technique at bit-level. Subsequently, the transfer function (6) should be transformed in order to group bits of one coefficient from all bit-planes (Fig. 5) to a single processing unit (PU) and avoid delays between operations in one plane [13,14].

By avoiding delays between operations in one plane and grouping bits of one coefficient from all bit-planes, each PU from Fig. 5 can be folded to a single processing unit (PU_F) as it is shown in Fig. 6. In Fig. 6 the number of folding sets, denoted as k, is adopted to be equal to number of coefficients (k_C), i.e., $k = k_C$. In the first PU_F the product ($c_{k_C-1} \cdot x_0$) is computed in m clock cycles and accumulated (Fig. 6). The output of the first PU_F is shifted by one weight and then the second partial product, $c_{k_C-2} \cdot x_1$, is processed in the second PU_F, and so on [13].

Bits of one coefficient from all bit-planes can be grouped to a single processing unit by reordering the multiplications within BPA transfer function (6).

If we reorder partial products according to z^{-j}, $G(z)$ is of the form:

$$G(z) = z^{-[(m-1)k_C+1]} \sum_{j=0}^{k_C-1} \sum_{i=0}^{m-1} c_j^i 2^i z^{-j} = z^{-[(m-1)k_C+1]} \sum_{j=0}^{k_C-1} z^{-j} \sum_{i=0}^{m-1} c_j^i 2^i. \quad (7)$$

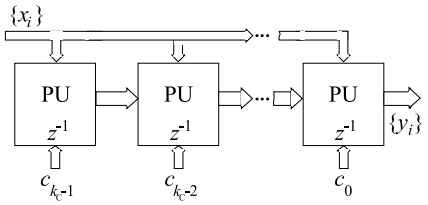

Fig. 5. Architecture of transformed bit-plane FIR filter.

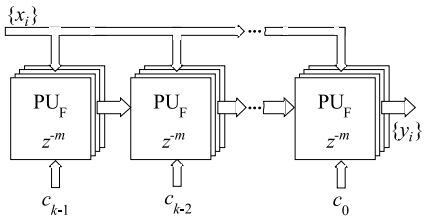

Fig. 6. Architecture of bit-plane FIR filter with folded bit multiplications.

Transformed form of transfer function (6) is

$$G(z) = z^0\left(c_0^{m-1}2^{m-1} + c_0^{m-2}2^{m-2} + \cdots + c_0^0 2^0\right) + z^{-1}\left(c_1^{m-1}2^{m-1} + c_1^{m-2}2^{m-2} + \cdots + c_1^0 2^0\right) + \cdots + z^{-(k_C-1)}\left(c_{k_C-1}^{m-1}2^{m-1} + c_{k_C-1}^{m-2}2^{m-2} + \cdots + c_{k_C-1}^0 2^0\right). \quad (8)$$

Reordering of partial products in (7) affected the architecture to start computation using coefficient c_{k_C-1}, instead of starting the computation with coefficient c_0. Thus, direct form of bit-plane FIR filter is converted to transposed form bit-plane FIR filter. DFG of transposed bit-plane FIR filter that corresponds to Eq. (8) is shown in Fig. 7. Bit-plane FIR filter in transposed form is well prepared for application of folding technique.

Besides the deriving of suitable DFG for folding important issue is the setting of folding sets. In respect to folded architecture shown in Fig. 6, the folding sets are formed as it is shown in Fig. 7.

The folding sets assignment (S_s, r) where s is folding set with timing order r, used for synthesis of transposed bit-plane FIR filter with folded bit multiplications and changeable folding factor (Fig. 7), is done according to the following equations:

$$s = \lfloor (p-1)/m \rfloor \qquad r = (p-1) \mod N. \quad (9)$$

Using folding set assignment (9), as it is shown in Fig. 7, leads to the architecture with folded bit multiplications shown in Fig. 6.

The operations that will be folded are denoted with dashed lines in Fig. 7. One operation from transposed DFG (TDFG) assumes forming of partial products and the addition performed in one "row" of basic cells, (where basic cell contains AND gate and full adder) [13]. There are k folding sets $(k = k_C), S_0, S_1, S_2, \ldots, S_{k-1}$, and the number of folding sets is equal to the number of taps. Each folding set contains m operations, i.e., the folding factor, N, is equal to the coefficient length, $N = m$. Thus, folding equations (1) for determined folded sets, where $P_U = 0$ and U and V are nodes in TDFG from Fig. 7, denoted with p, where $p = 1, 2, \ldots, m, m+1, \ldots, k_C \cdot m$, are:

$$D_f(1 \to 2) = m \cdot 0 - 0 + 1 - 0 = 1$$
$$D_f(2 \to 3) = m \cdot 0 - 0 + 2 - 1 = 1$$
$$\cdots$$
$$D_f(m-1 \to m) = m \cdot 0 - 0 + (m-1) - (m-2) = 1$$
$$D_f(m \to m+1) = m \cdot 1 - 0 + 0 - (m-1) = 1$$
$$D_f(m+1 \to m+2) = m \cdot 0 - 0 + 1 - 0 = 1$$
$$\cdots$$
$$D_f(k_C \cdot m - 1 \to k_C \cdot m) = m \cdot 0 - 0 + (m-1) - (m-2) = 1.$$

The condition $D_f(U \to V) \geqslant 0$ is satisfied for each pair of connected nodes (U, V), and it proves that TDFG from Fig. 7 is well prepared for folding.

The DFG of obtained folded architecture (FA) with k_C taps is given in Fig. 8. Data flow for transposed bit-plane FIR filter with folded bit multiplications is shown in Fig. 9.

Folded architecture from Fig. 8 is derived in the general form for k_C taps and coefficient length m. Proposed application of folding technique allows the simultaneous bit-serial operation of all taps.

The duration of computation in each tap depends on coefficient length m. So, changing of duration of computation cannot change the number of folding sets (number of coefficients), but it can change the number of folded operations within folding sets, i.e., coefficient length (Figs. 6 and 8).

The changeable folding set is the folding set where the number of folded operations can vary. The control units that are able to change the number of operations in folding sets (coefficient length)

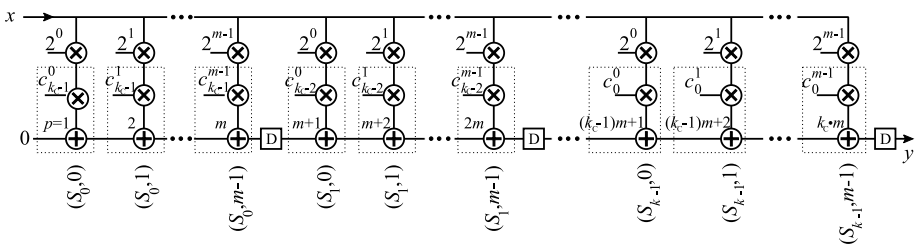

Fig. 7. DFG of transposed bit-plane FIR filter.

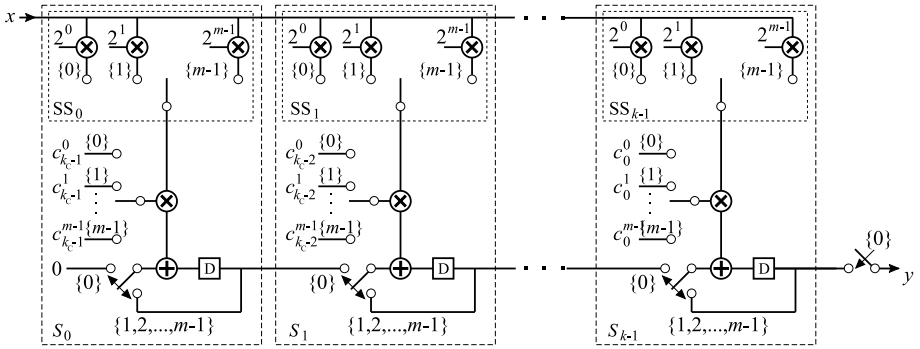

Fig. 8. DFG of transposed bit-plane FIR filter with folded bit multiplications; folding sets are $S_0, S_1, \ldots, S_{k-1}$.

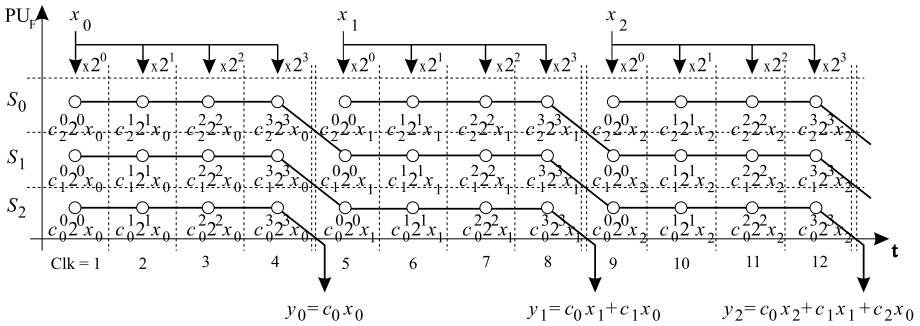

Fig. 9. Data flow of transposed bit-plane FIR filter architecture with folded bit multiplications.

are associated to each folding set and consist of switch with configurable length for coefficients entering, and switch with configurable switching time for partial products accumulation (Fig. 8).

Application of folding technique allows user to configure folding factor onto implemented array size. Configuration facility of the proposed architecture relates to coefficient length. It provides flexible computation and offers the possibility of increasing of folded system throughput at cost of decreasing the coefficient length according to the folding factor. Now, the number of coefficients (k_C) is assumed to be the design parameter.

Functional block diagram of transposed bit-plane FIR filter with folded bit multiplications from Fig. 8 is shown in Fig. 10. The numbering system is inherited from the BPA from Fig. 4 (2's complement).

Entering of input words is controlled by multiplexers in the first row of basic cells, which are driven by control signal $ck1$. The high-level of control signal $ck1$, shown in Fig. 10, enables new input word $\{x\}$ to enter the architecture in every $N \cdot l + 0$th (i.e., $m \cdot l + 0$th) clock cycle, otherwise the path is folded and partial results are accumulated (Fig. 8). The operation with different folding factors is provided by simple changing of control signal $ck1$ (Fig. 10), which period should be equal to m periods of the basic clock signal $ck0$. Each coefficient bit register with its 1-to-m_1 demultiplexer provides rotation within the range 1-to-m_1. The path for m-bit ($0 < m \leqslant m_1$) rotation in the coefficient bit register is closed at mth coefficient bit by attached demultiplexer. Thus, the m-bit rotation is provided.

In comparison with the BPA, the array from Fig. 10 is restricted for the factor $N = m$, i.e., the folding factor is equal to the coefficient length. The proposed transformation of source DFG for the bit-plane architecture enables the synthesis of fully pipelined FIR filter architecture with folded bit multiplications and changeable folding factor, but does not enable the changing of number of coefficients.

Gathering of all bits from one coefficient within one folding set using proposed folding set assignment makes the number of coefficients equal to the number of folding sets (i.e., PU). The number of PUs is architecture parameter that cannot be changed.

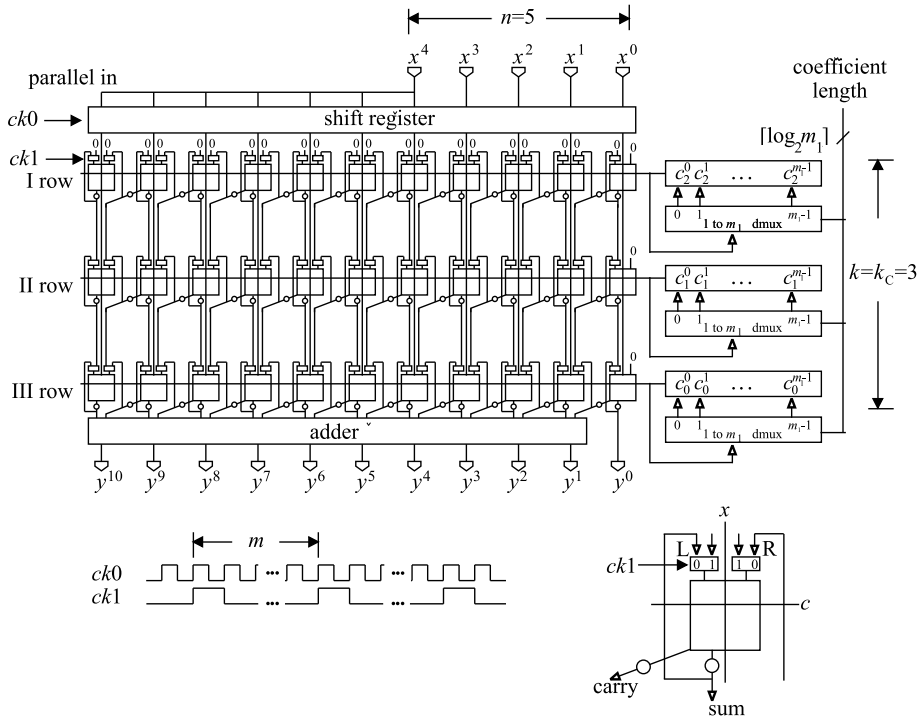

Fig. 10. Functional block diagram of transposed bit-plane FIR filter folded bit multiplications with changeable folding factor.

5. Synthesis of folded architecture with changeable number of coefficients

Architecture shown in Fig. 8 is able to operate with different coefficient lengths, but the number of coefficients is equal to the number of processing units that is invariable. In order to enable the changing of number of coefficients in fixed size array, the number of coefficients should not be equal to the number of PUs. If the number of coefficients is not equal to the number of PUs, then folded architecture has to have input data supply module, as well as coefficient bits supply module to feed the architecture with data. Supply modules manage data flow within the architecture for different number of coefficients. Folded bit-plane FIR filter architecture with changeable number of coefficients is shown in Fig. 11. Furthermore, folded architecture with that property has a data path from output of the last PU to the input of the first. Cyclic data path makes the number of PUs transparent to the number filter taps, i.e., allows accumulation of partial products by encircling data as long as it is necessary to finish computation of output word.

Existence of cyclic data path within folded architecture can be achieved only by assigning new folding set to TDFG from Fig. 7. The folding sets, assigned in the synthesis process of transposed FIR filter with folded bit multiplications and changeable folding factor, did not enable the synthesis of FIR filter with changeable number of coefficients.

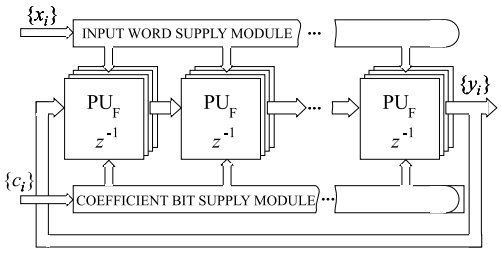

Fig. 11. Architecture of folded bit-plane FIR filter with configurable number of coefficients.

5.1. Folding set assignment

In order to obtain an architecture that enables changing of coefficient number, bit-plane architecture should be folded using new folding set assignment that produces an encircling data path. Also, the number of folding sets should not be obligatory equal to the number of coefficients. Expected data flow, in respect to the architecture from Fig. 11, is sketched in Fig. 12.

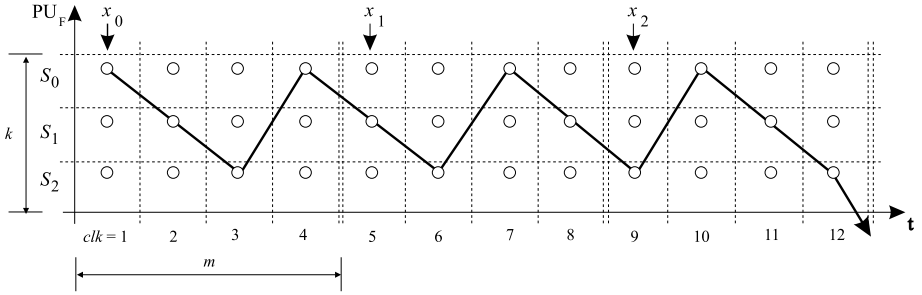

Fig. 12. Sketch of new data flow with data paths from last folding set (S_2) into the first (S_0) for $k = 3$ and $m = 4$.

In order to enable configuration of number of taps in fixed folded FIR filter array we propose a new mapping of operations onto the DFG nodes rather then changing the topology of the DFG by additional transformations. The following notation is involved:

m_C	coefficient length,
k_C	number of coefficients,
k	number of folding sets,
L	total number of operations in the DFG, where one operation assumes forming of partial product and the addition performed by one "row" of basic cells (basic cell contains AND gate and full adder),
p	position of operation within the DFG ($1 \leqslant p \leqslant L$).
(S_s, r)	folding set s with timing order r.

New assignment of folding sets is involved with aim to enable the synthesis of the folded FIR filter architecture that will support the changing of both coefficient number and coefficient length. The folding set assignment, in respect to data path from Fig. 12, is shown in Fig. 13 and described with:

$$\begin{aligned} s &= (p-1) \mod k \\ r &= (p-1) \mod N. \end{aligned} \tag{10}$$

The crucial novelty is that (10) enables the changing of operations in folding sets. In other words the different operations can be mapped onto the different hardware units in fixed array structure. Eq. (10) provide k folding sets where each folding set contains N operations. For the coefficients, k_C, and the coefficient length, m_C, the total number of operations, L, is

$$L = k_C \cdot m_C = k \cdot N. \tag{11}$$

Let us note that the number of folding sets is not obligatory equal to the number of coefficients.

In order to have an answer to the question whether the system is foldable or not, we have to calculate the folding equations (1) and check the condition $D_f(U \to V) \geqslant 0$.

Folding equations (1) for the DFG from Fig. 13 with folding sets described with (10) are:

$$D_f(1 \to 2) = N \cdot 0 - 0 + 1 - 0 = 1$$

$$D_f(2 \to 3) = N \cdot 0 - 0 + 2 - 1 = 1$$

$$\cdots$$

$$D_f(k \to k+1) = N \cdot 0 - 0 + k - (k-1) = 1$$

$$\cdots$$

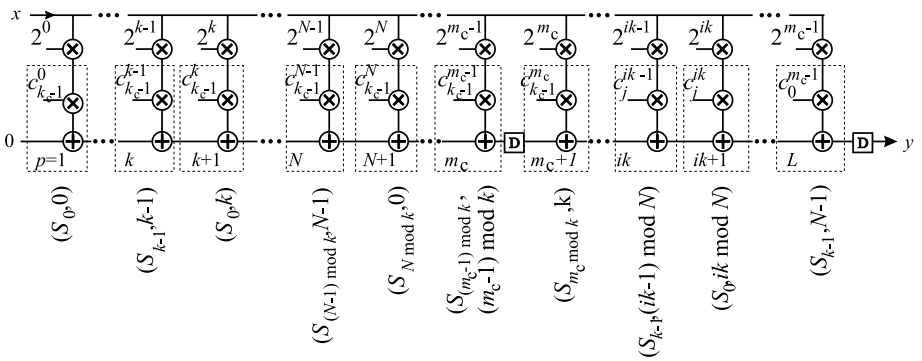

Fig. 13. Transposed DFG with new assignment of folding sets.

$$
\begin{aligned}
&D_f(N \to N+1) = N\cdot 0 - 0 + 0 - (N-1)\\
&\qquad\qquad = -(N-1)\\
&\cdots\\
&D_f(m_C \to m_C+1) = N\cdot 1 - 0 + (m_C \mod k)\\
&\quad - ((m_C-1) \mod k)\\
&\quad = N+1\\
&\cdots\\
&D_f(n\cdot k \to n\cdot k+1) = N\cdot 0 - 0 + (n\cdot k \mod k)\\
&\quad - ((n\cdot k-1) \mod k) = 1\\
&\cdots\\
&D_f(L-1 \to L) = N\cdot 0 - 0 + (N-1) - (N-2) = 1,
\end{aligned}
\tag{12}
$$

where $P_U = 0$ while U and V are nodes of the TDFG denoted as $p = 1,2,3,\ldots,k,\ k+1,\ldots,N,\ N+1,\ldots,m_C,\ m_C+1,\ldots,n\cdot k,\ n\cdot k+1,\ldots,L-1,\ L (1 \leqslant n \leqslant N)$.

Folding equations (12) can be given in the following form:

$$
\begin{aligned}
&D_f(p \to p+1)\\
&\quad = N\cdot w(e) - 0 + [p \mod N] - [(p-1) \mod N]\\
&\quad = \begin{cases} -N+1, & p \mod N = 0,\\ N+1, & p \mod m_C = 01,\ 1 \leqslant p \leqslant L-1,\\ 1, & \text{other.}\end{cases}
\end{aligned}
\tag{13}
$$

From (12) or (13), it can be seen that the condition $D_f(U \to V) \geqslant 0$ is not satisfied for each Nth node, i.e., for nodes U and V on positions where $p_U \bmod N = 0$. The reason why the folding condition is not satisfied is that there are delays between coefficients c_i and c_{i+1}, $i = k_C - 2, k_C - 1, \ldots, 0$, but not between the folding sets in DFG (Fig. 13). Delays causes negative values in folding equations (12). This is the main reason for retiming of the DFG from Fig. 13.

5.2. Retiming

Using the system of inequalities (4) and system of folding equations (13) the following system of inequalities is obtained

$$
r(p) - r(p+1) \leqslant \begin{cases} -1, & p \mod N = 0,\\ 1, & p \mod m_C = 0, 1 \leqslant p \leqslant L-1,\\ 0, & \text{other.}\end{cases}
\tag{14}
$$

The constraint graph (Fig. 14) is formed with aim to provide the solution for inequalities (14). The constraint graph is directed graph where for each $r(p)$, $p = 1,2,\ldots,L-1$, L, from (14) one node is assigned. Each inequality of type $r(U) - r(V) \leqslant \gamma$ is represented by the directed edge from node V to node U with assigned weight γ. An additional node denoted with $L+1$ is connected with all other nodes with zero-weighted edges. The sum of weights on the shortest path, from node $L+1$ to the node that corresponds to $r(p)$, represents the retiming for $r(p)$.

In order to provide the solution for inequalities (14), let us highlight important features of constraint graph. First feature relates on direction of edges (excluding edges that connect node $L+1$ with other nodes). The edge is always directed from the node with higher position to the node with lower position, and connects only neighboring nodes. Second feature concerns edge index. The weight of the edge that has a destination in node p (Fig. 14) can be described as follows:

$$
w_i = \begin{cases} -1, & p \mod N = 0,\\ 1, & p \mod m_C = 0,\ p = 1,2,\ldots,L-1,\\ 0, & \text{other.}\end{cases}
$$

The number of edges with weight $w_i = -1$, as well as with weight $wi = 1$ can be derived using previously described features.

The number of edges with weight $w_i = -1$, counting down from the node L to node $p (1 \leqslant p \leqslant L)$ is

$$\left\lfloor \frac{L-p}{N} \right\rfloor,$$

while the number of edges with weight $w_i = 1$ is

$$\left\lfloor \frac{L-p}{m_C} \right\rfloor.$$

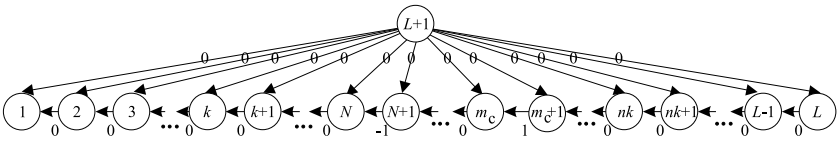

Fig. 14. Constraint graph.

Thus, the general form for retiming for node p, $r(p)$, can be obtained by adding weights of all edges from node L to node p. Therefore, the general form of retiming $r(p)$ for node p is

$$r(p) = \left\lfloor \frac{L-p}{m_C} \right\rfloor - \left\lfloor \frac{L-p}{N} \right\rfloor. \quad (15)$$

For particular case

$$m_C = N, \quad \text{i.e. } k_C = k \Rightarrow r(p) = 0 \quad \text{for all nodes } p\ (p = 1, 2, \ldots, L), \quad (16)$$

and subsequently the retiming is not required.

In order to simplify implementation and reduce chip occupation we force the last PU of folded architecture to be the only PU that will have output lines for $\{y\}$. We involve the constrain that coefficient length (m_C) is divisible with the number of folding sets (k), i.e., $m_C = q \cdot k$, $q = 1, 2, 3, \ldots$ In that manner the computation of output words always ends in last PU. The constrain implies, according to (11), that $m_C \geqslant N$. Bearing in mind that $m_C \geqslant 1$ and $N > 1$, the lower and upper bounds for the retiming (15) are

$$r(1) = \left\lfloor \frac{L-1}{m_C} \right\rfloor - \left\lfloor \frac{L-1}{N} \right\rfloor = (k_C - 1) - (k - 1) = k_C - k,$$

and

$$r(L) = \left\lfloor \frac{0}{m_C} \right\rfloor - \left\lfloor \frac{0}{N} \right\rfloor = 0,$$

respectively. As $m_C \geqslant N$, according to (11), the k_C is less or equal to k ($k_C \leqslant k$), which places the solution for retiming (15) at negative part of $r(p)$ axis. The graphical representation of retiming (15) is sketched in Fig. 15.

Fig. 15 and solution for retiming (15) show that the number of required successive input words $\{x\}$, used for simultaneous processing in architecture from Fig. 15, is $k - k_C + 1$. Actually, input word, x_i, is exploited as an operand in $k - k_C + 1$ successive computations on different nodes. For the chosen folding factor N ($N \geqslant 1$) input data word x_i is active in the folded system for $T = (k - k_C + 1) \cdot N$ clock cycles.

The minimal number of registers, R, corresponds to the maximal number of active variables that is obtained for $k_C = 1$. Thus, the minimal number of registers is $R = k - 1 + 1 = k$. Since the folding factor is N, a new input data word will be entered into the folded system every N clock cycles. Graphical

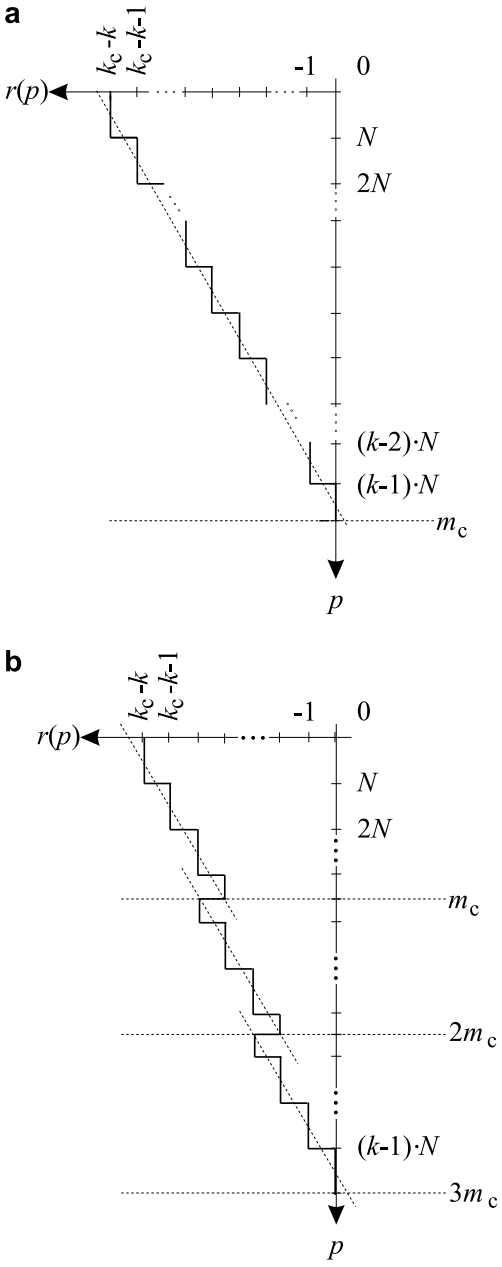

Fig. 15. Graphical representations of retiming: (a) for case $k_C = 1$, $m_C = L$; (b) for case $k_C = 3$, $m_C = L/3$.

representations from Fig. 15 shows that entering is required at $n \cdot m_C$ ($n = 1, 2, \ldots, k_C - 1$) time instances, also.

According to the previous discussion, graphical representation of input data life cycle is generated (Fig. 16). Horizontal dashed lines in Fig. 16 represent the time instances, while vertical lines represent activity of input data words. The input data word x_i is "live" since it enters the architecture (beginning of

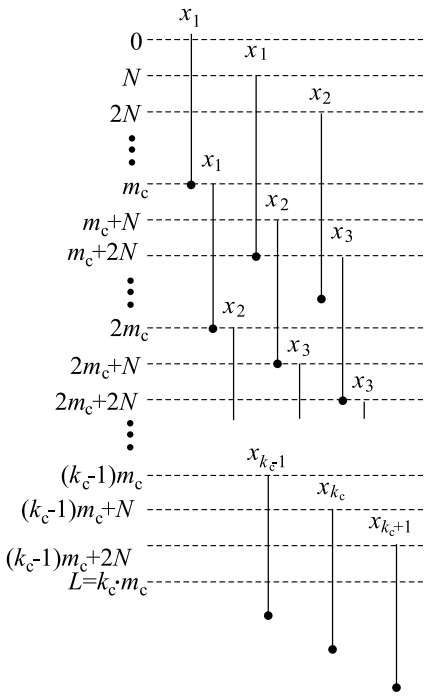

Fig. 16. Graphical representation of life cycle in general form.

vertical line), and becomes "dead" as soon as all computations that use x_i are completed (dot at the end of vertical line), as it is shown in Fig. 16.

The next step is the forming of allocation table. Allocation table shows the allocation of input words in registers. Allocation table is obtained using input data life cycle form Fig. 16, and it is shown in Fig. 17.

5.3. Architecture of folded FIR filter with changeable number of coefficients

Hardware module for input data entering (Fig. 18) is designed according to the allocation table from Fig. 17.

The general form of folded FIR filter architecture with changeable number of coefficients and coefficient length is given in Fig. 19. Folding sets S_0, $S_1,\ldots,S_{k-1}$ are shown in dashed boxes (Fig. 19). Each folding set contains N operations. Initially, the computation starts in folding set S_0 where the product $2^0\cdot c^0_{k_C-1}\cdot x_0$ is obtained in the first clock cycle. In the next clock cycle folding set S_1 generates the partial product $2^1\cdot c^1_{k_C-1}\cdot x_0$ adding previously computed partial product from folding set S_0. Thus, the value$(2^0\cdot c^0_{k_c-1}\cdot x_0)+(2^1\cdot c^1_{k_c-1}\cdot x_0)$ is entered into the next section, which performs the operations from S_2, in the third clock cycle.

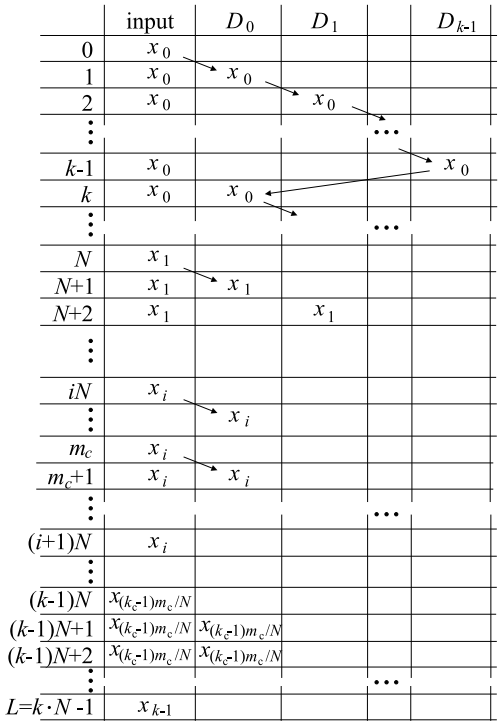

Fig. 17. General form of allocation table.

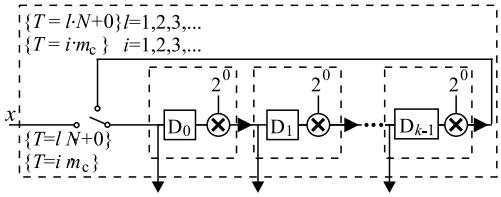

Fig. 18. Hardware module for input data entering.

The next important time instance is $(k+1)$st clock cycle. In that clock cycle both input data path and summation path are folded from section S_{k-1} to S_0. In input data path product $2^k\cdot x_0$ is present at input of the section S_0, while in the summation path $(2^0\cdot c^0_{k_C-1}\cdot x_0)+(2^1\cdot c^1_{k_C-1}\cdot x^0)+\cdots+(2^{k-1}\cdot c^{k-1}_{k_C-1}\cdot x_0)$ enters the same section. S_0 adds $2^k\cdot c^k_{k_C-1}\cdot x_0$ to the entered sum. However, the computation for the coefficient c_{k_C-1} is not finished yet.

The complete product $c_{k_C-1}x_0$ is obtained in the section $S_{(m_C-1)\bmod k}$ during clock cycle m_C. The computation of $c_{k_C-2}x_1$ starts in (m_C+1)st clock cycle. The section $S_{m_C \bmod k}$ computes

$$\begin{aligned}&\{(2^0\cdot c^0_{k_C-1}x_0)+(2^1\cdot c^1_{k_C-1}\cdot x_0)+\cdots\\&\quad+(2^{m_C-1}\cdot c^{m_C-1}_{k_c-1}\cdot x_0)\}+2^0\cdot c^0_{k_C-2}\cdot x_1\\&\quad=(c_{k_C-1}x_0)+(2^{m_C-1}\cdot c^0_{k_C-2}\cdot x_1).\end{aligned}$$

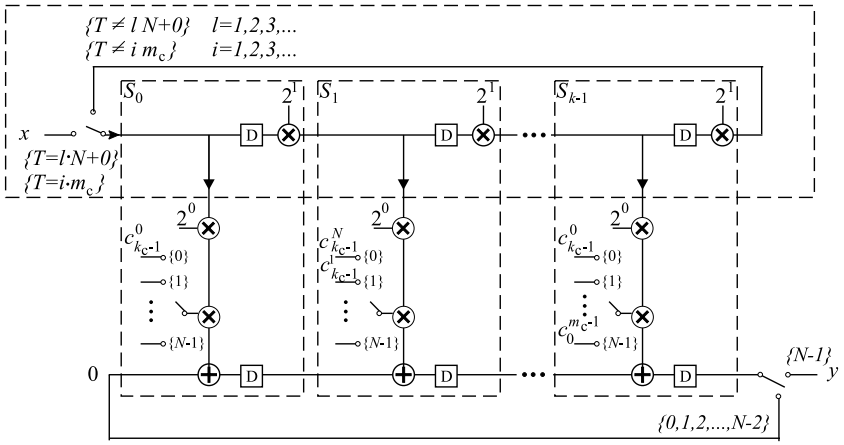

Fig. 19. Folded FIR filter architecture with changeable number of coefficients and coefficient length.

In order to explain method of operation we have started the explanation with computation of y_{k_C-1} that is the first result where all coefficients are included. Let us note that in the simultaneous process the architecture starts with computation of all results $y_{k_C-1}, y_{k_C-2}, \ldots, y_0$ in reverse order. New output word $\{y\}$ is generated every N clock cycles. The first generated result is y_0, which is obtained using only one coefficient ($y_0 = c_0 \cdot x_0$) requiring m_C plane operations. As output words are present in output lines each Nth clock cycle, initial latency if $m_C > N$ is $q \cdot N$ clock cycles, where q is the smallest integer for which the condition $q \cdot N - m_C > 0$ is satisfied. For cases when $m_C < N$ initial latency is equal to m_C.

The data flow through the folded architecture for case $k = 3$, $N = 4$, $k_C = 2$ and $m_C = 6$ is given in Fig. 20, and the functional block diagram of the derived folded architecture from Fig. 19 is shown for case $k = 3$ and $N = 4$ in Fig. 21. The numbering

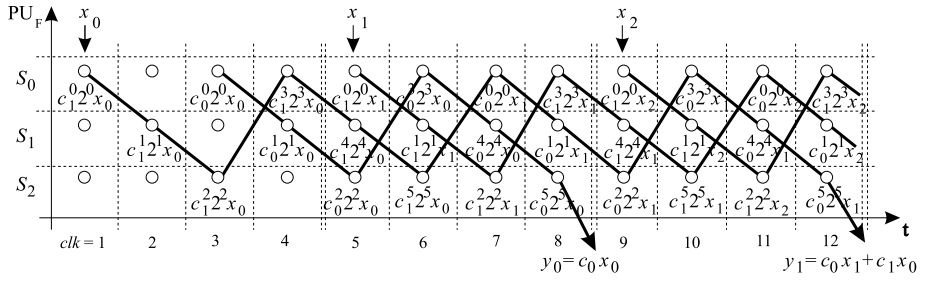

Fig. 20. Data flow for folded architecture ($k = 3$, $N = 4$, $k_C = 2$ and $m_C = 6$).

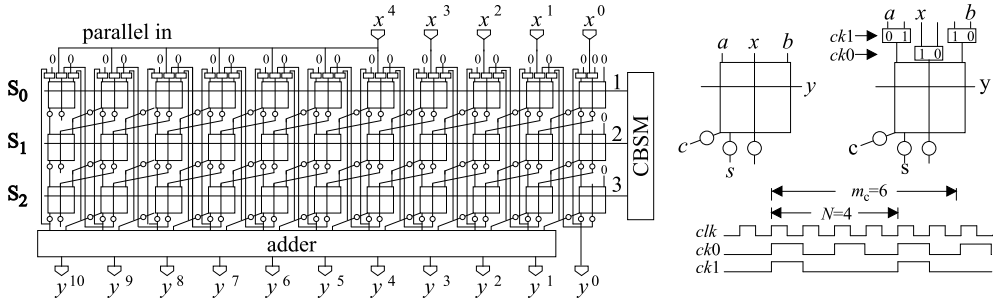

Fig. 21. Functional block diagram of folded bit-plane FIR filter with changeable number of coefficients.

system of the architecture from Fig. 21 is inherited from the BPA from Fig. 4 (2's complement for $\{x\}$, and unsigned coefficients).

The proposed architecture supports the operation with changeable number of coefficients and coefficient length. The mechanism for throughput increasing can be easily exploited on proposed architecture [14,15]. Let us note that the ordering of coefficient bits depends on number of coefficients k_C and coefficient length m_C. The folded array from Fig. 21 requires the involvement of specialized hardware module for feeding the architecture with coefficient bits. The coefficient bit supply module (CBSM), shown in Fig. 21, provides the proper ordering of coefficient bits. The module is based on mathematical dependencies that are inherent to the folding dependencies between operations and functional units, which are described with folding set assignment (10).

5.4. Synthesis of coefficient bit supply module for folded architecture

Assume that operation p $(1 \leqslant p \leqslant L)$, shown in Fig. 13, performs multiplication of input data words by coefficient bit c_i^j. According to (10), operation p belongs to folding set $s = (p-1) \bmod k$, with folding order $r = (p-1) \bmod N$. In other words, folded architecture multiplies input data word (Figs. 19 and 20) by coefficient c_i^j on folding set $S_s (0 \leqslant s \leqslant k-1)$ in time instance $\delta \cdot N + r$ $(0 \leqslant r \leqslant N-1;\ \delta = 0,1,2,\ldots)$. The operation that has position in DFG equal to p (Fig. 13), according to folding set assignment (10), can be described as

$$p = m_C(k_C - (i+1)) + j + 1. \tag{17}$$

The dependency between folding set s $(0 \leqslant s \leqslant k-1)$ and folding order r of coefficient bit c_i with weight 2^j, using (10) and (17), is obtained as

$$\begin{aligned} s &= (m_C \cdot (k_C - (i+1)) + j) \mod k \\ r &= (m_C \cdot (k_C - (i+1)) + j) \mod N. \end{aligned} \tag{18}$$

Expression (18) describes the folding set s that performs multiplication by coefficient c_i^j in time instances $\delta \cdot N + r (0 \leqslant r \leqslant N-1;\ \delta = 0,1,2,\ldots)$.

Inverse dependencies, denoted as $i = f(s,r)$ and $j = g(s,r)$, can be obtained by mapping position of operation p to matrix $A_{k\times N}$ in accordance with folding set assignment (10). Each column in matrix $A_{k\times N}$ represents one folding set S_s $(0 \leqslant s \leqslant k-1)$ and each row stands for time instances r $(0 \leqslant r \leqslant N-1)$ where the folding set, S_s, performs operation p. Matrix $A_{k\times N}$ for the case when $k = 3$ and $N = 4$ is

$$A_{3\times 4} = \begin{vmatrix} 1 & 5 & 9 \\ 10 & 2 & 6 \\ 7 & 11 & 3 \\ 4 & 8 & 12 \end{vmatrix}. \tag{19}$$

General form of matrix $A_{k\times N}$ is

$$A_{k\times N} = \begin{vmatrix} 1 & & & N+1 & & & \\ & 2 & & & \ddots & & \\ & & 3 & & & & \\ & & & 4 & & & \\ & & & & \ddots & & \\ & & & & & k & \\ k+1 & & & & & & \\ & \ddots & & & \ddots & & \\ & & N & & & k\cdot N & \end{vmatrix}. \tag{20}$$

With aim to simplify the solution, and to emphasize dependence between operation p and its position in matrix $A_{k\times N}$, modulo dependencies from (19) are removed and new matrix $A'_{2k\times N}$ is created. Matrix $A'_{2k\times N}$ for the case when $k = 3$ and $N = 4$ have the following form:

$$A'_{6\times 4} = \left|\begin{array}{ccc|ccc} 1 & 5 & 9 & & & \\ & 2 & 6 & 10 & & \\ & & 3 & 7 & 11 & \\ & & & 4 & 8 & 12 \end{array}\right|. \tag{21}$$

The value on position (s,r) in matrix (21) can be described as

$$p = ((s + a\cdot k) - r)\cdot N + (r+1), a = \begin{cases} 0, & s \geqslant r \\ 1, & s < r \end{cases}. \tag{22}$$

Coefficient bits in DFG from Fig. 14 are assigned to operations according to the following:

$$\begin{aligned} i &= k_C - 1 - \left\lfloor \frac{p-1}{m_C} \right\rfloor, \\ j &= (p-1) \mod m_C. \end{aligned} \tag{23}$$

Using (22) and (23), dependencies $i = f(s,r)$ and $j = g(s,r)$, can be developed as

$$
\begin{aligned}
i &= k_C - 1 - \left\lfloor \left(\frac{((s + a \cdot k) - r) \cdot N + r}{m_C} \right) \right\rfloor, \\
a &= \begin{cases} 0, s \geqslant r \\ 1, s < r \end{cases}, \\
j &= (((s + a \cdot k) - r) \cdot N + r) \mod m_C, \\
a &= \begin{cases} 0, s \geqslant r \\ 1, s < r \end{cases}.
\end{aligned} \tag{24}
$$

According to (18) and (24) CBSM can be developed as two dimensional array of latches with k rows and N columns, where each latch stores one coefficient bit. Parameters k_C and m_C determine the reordering of coefficient bits. In order to synthesize the reordering array, which will reorder the coefficient bits from input coefficient bit-stream entered in regular form, let us determine the final position of coefficient bit $c^0_{k_C-1}$ within reordering array. From (18), coefficient bit $c^0_{k_C-1}$ will be placed in the first folding set ($s = 0$, the first row in reordering array), with timing order $r = 0$ (the first column). The coefficient bit $c^1_{k_C-1}$ will be placed in the second folding set ($s = 1$, the second row), with folding order $r = 1$ (the second column), etc. The coefficient bit $c^{m_C-1}_0$ will be placed in the last column of the last row.

In respect to the previous analysis, CBSM has to provide initialization (reordering) mode, where coefficient bits are entered in bit-serial manner. The least significant bit of the last coefficient ($c^0_{k_C-1}$) should be entered first, followed by coefficient bit $c^1_{k_C-1}$, etc. The coefficient bits should flow through the reordering array following described reordering path. CBSM array is shown in Fig. 22a, where reordering, i.e., initialization path, is shown with dashed lines.

CBSM has two operational modes. The first, initialization mode, when coefficient bits are entered into the CBSM, and the second, run mode, when CBSM feeds the array with coefficient bits. As the initialization mode was analyzed by (18), the run mode can be analyzed in the same manner using (24). Coefficient bits flow during the run-mode is shown with solid lines in Fig. 22. Rows are implemented as shift registers, so during the run mode coefficients rotate through the rows from right to left, feeding each folding set of folded array with coefficient bits in correct order (Fig. 22a). Initial positions of bits within CBSM for $k = 3$, $N = 4$, $k_C = 2$ and $m_C = 6$, after initialization, is shown in Fig. 22b.

The number of clock cycles, required for initializing the structure, is $k \cdot N$. After $k \cdot N$ clock cycles,

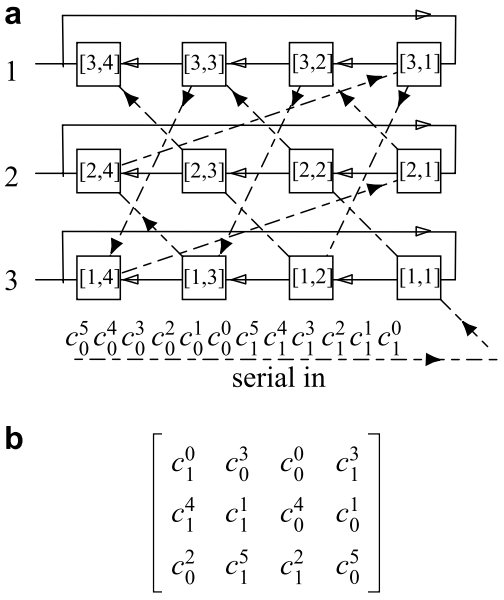

Fig. 22. (a) CBSM for $k = 3$, $N = 4$, $k_C = 2$ and $m_C = 6$ and (b) layout of coefficient bits after initialization.

coefficient bits are loaded for the beginning of computation. In the processing mode coefficients rotate through the rows from right to left.

Programmability of CBSM assumes changing of both number of coefficients and coefficient length with aim to provide flexibility as it is shown in Fig. 11. The synthesized module is able to handle feeding of folded array that performs computation with deferent number of coefficients and coefficient length.

Product of number of coefficients and coefficient length ($k_C \cdot m_C$) is invariable for particular folded array in respect to (11), and it is equal to the product of number of folding sets and folding factor ($k \cdot N$). Decreasing the coefficient length (m_C), number of coefficients (k_C) must be increased. In most applications there is no need for increasing the number of coefficients each time when coefficient length is decreased, and in the same time, there is no need for increasing the coefficient length each time when number of coefficients is decreased.

If number of coefficients (k_C) in these cases remains the same, fewer operations are required to complete the computation. Therefore, if folded architecture is able to meet the requirement of reducible number of folding sets (k) or reducible folding factor (N), then architecture is able to increase throughput each time when coefficient number (k_C) or coefficient length (m_C) is reduced,

in respect to equation $k \cdot N = k_C \cdot m_C$. The architecture throughput in that case could be increased $k \cdot N/k_C \cdot m_C$ times.

5.5. Reducible folding factor

Let the coefficient length m_C be reduced to m_C^R, while the number of coefficients k_C, in the implemented architecture, remains constant. Straightforward way to reduce m_C is to replace the most significant bits of coefficients with zeros. Initial state of the CBSM from Fig. 22b, for $k = 3$, $N = 4$, $k_C = 2$ and $m_C = 6$, where the coefficient length is reduced to $m_C^R = 3$, is

$$\begin{bmatrix} c_1^0 & 0 & c_0^0 & 0 \\ 0 & c_1^1 & 0 & c_0^1 \\ c_0^2 & 0 & c_1^2 & 0 \end{bmatrix}.$$

In respect to coefficient bits ordering during the initialization of CBSM (Fig. 22a), for the case $k = 3$, $N = 4$, $k_C = 2$ and $m_C^R = 3$, coefficient bits should enter the architecture in following order:

$$000c_0^2c_0^1c_0^0000c_1^2c_1^1c_1^0.$$

Replacing the most significant coefficient bits with zeros the coefficient length is reduced, but number of computations remains the same as well as the throughput of architecture.

In order to reduce the coefficient number (k_C) or coefficient length (m_C), at least one of parameters k and N has to be changed. Let us consider the following scenario, with aim to find out which parameter is more suitable for changing.

The number of rows in folded array from Fig. 21 is equal to the number of folding sets (k). Thus, the number of folding sets cannot be changed in a straightforward manner. The number of columns in folded array is $m_C + n + \log_2 k_C$. By reducing coefficient length m_C fewer number of columns is required for computation. However, performing computation on wider bit-planes does not affect nor computation time or accuracy.

The number of rows in CBSM is equal to the number of rows in folded array (k), while the number of columns is equal to folding factor N. Number of columns in CBSM can be reduced using shift registers with changeable length, as it is shown in Fig. 23.

Using CBSM with changeable length, CBSM+, (Fig. 23), layout of coefficient bits after initialization for $k_C = 2$ and $m_C = 3$ is

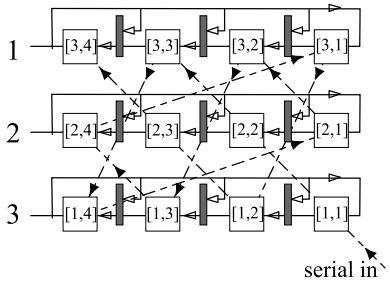

Fig. 23. CBSM+ for $k = 3$, $N = 4$.

$$\begin{bmatrix} c_1^0 & c_0^0 & 0 & 0 \\ c_0^1 & c_1^1 & 0 & 0 \\ c_1^2 & c_0^2 & 0 & 0 \end{bmatrix},$$

and coefficient bits should be entered in following order:

$$00c_0^0c_1^200c_0^2c_0^100c_1^1c_1^0.$$

Ordering of coefficient bits in initialization array relays on modulo dependencies, and it can be easily implemented by algorithm within software that reconfigures folded array, in respect to (19) and (20).

The computation time linearly depends on coefficient length for constant number of coefficients. Instead of extending the coefficient to the full coefficient length when the operation with coefficients with smaller length is required, the proposed CBSM+ reduces the folding factor according to the coefficient length and increases the throughput $k \cdot N/k_C \cdot m_C^R$ times.

6. Implementation

The folded FIR filter architecture was described in VHDL as a parameterized FIR filter core. The implementation was done onto the VirtexE FPGA with aim to illustrate what filtering can be carried out onto the configurable folded architecture. The implementation provides the results that relate on occupation and throughput. Implementation results for 14 different programmable folded architectures are obtained by Xilinx Webpack ISE 6.3, without any specific user constraints, using default optimization effort. The results are presented in Table 1. One row stands for one implemented architecture (with input word length n, number of implemented rows of mul/add array k, max folding factor that can be achieved N_{MAX}, and implemented length of output word y). Ohe number of used CLBs in target FPGA, equivalent gate count, clock period and

Table 1
Implementation results for different array sizes

n	k	N_{MAX}	y	Area		Clock (ns)	Freq. (MHz)
				CLB	Gate count (KG)		
8	**3**	**7**	**13**	**37**	**1.78**	**7.02**	**143**
8	4	8	18	72	3.38	9.95	101
8	4	16	26	108	5.17	8.00	125
8	4	32	42	183	8.76	11.20	89
8	8	4	15	95	4.30	9.80	102
8	8	7	19	129	5.89	11.50	87
8	8	8	20	136	6.31	9.40	106
8	8	16	27	201	9.40	11.50	87
8	**16**	**4**	**16**	**157**	**6.86**	**9.40**	**106**
8	16	8	20	223	10.16	11.80	85
8	16	16	28	359	16.78	11.00	91
8	32	4	17	263	10.56	12.50	80
8	32	8	21	366	15.55	11.30	88
8	32	16	29	604	26.89	11.50	87

max operating frequency are given for each implemented architecture. Graphical representations in Figs. 24 and 25 are generated from Table 1.

Fig. 24 illustrates the gate count, for architectures implemented with k rows in mul/add array, as a function of maximal folding factor (N_{MAX}). Gate count linearly depends on N_{MAX}. The increasing of N_{MAX} requires wider registers for coefficient storage (Figs. 22 and 23), wider rows of basic cells in mul/add array, as well as wider (vector merging) adder (the length of output data word is $y = n + N_{MAX} + \log_2 k$). Let us note that no truncation is involved.

Fig. 25 shows the gate count for architectures with different maximal folding factors, as a function of number of rows in mul/add array (k).

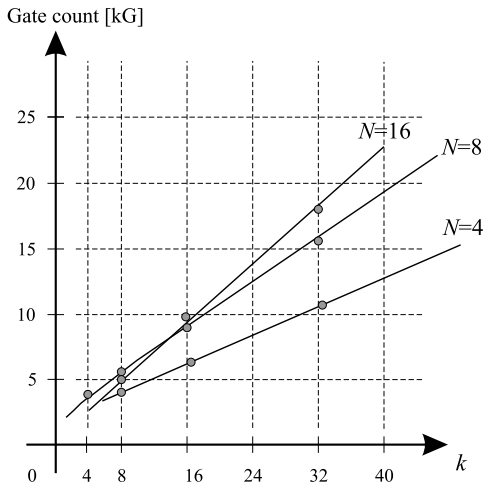

Fig. 25. Gate count as a function of number of rows in mul/add array.

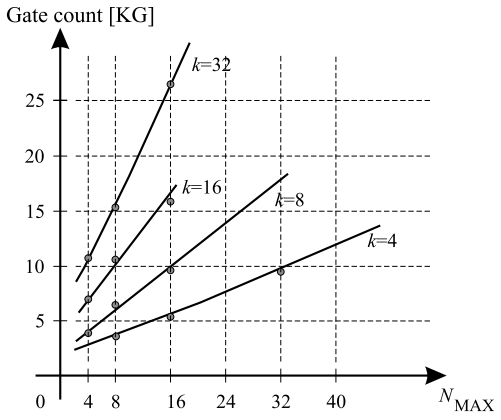

Fig. 24. Gate count as a function of maximal folding factor N_{MAX}.

The previous discussion concerns the implementation of different architectures. However, the most important feature of proposed architecture is the configuration of number of coefficients, coefficient length and folding factor.

Table 2 illustrates the configuration abilities of two implemented architectures, $n = 8$, $k = 3$, $N_{MAX} = 7$, $y = 13$ and $n = 8$, $k = 16$, $N_{MAX} = 4$, $y = 16$ (values indicated in bold in Table 1). The table shows the implementation results for clock period, reconfiguration period, initial latency, and throughput for possible programmed values of k_C and m_C, taking into account chosen folding factor N ($N_{MAX} \geqslant N \geqslant 1$). Reconfiguration time is equal to $k \cdot N$ clock cycles.

Table 2
Throughput in function of folding factor (N)

k	N	k_C	m_C	Clock (ns)	Reconf. (clk)	In.lat (clk)	Throughput $1/(N \cdot \text{clock})$ (MHz)
3	7	7	3	7.02	21	3	20.41
3	5	5	3	7.02	21	3	28.57
3	4	4	3	7.02	21	3	35.61
16	4	8	8	9.4	64	16	26.60
16	3	6	8	9.4	64	18	35.46
16	2	4	8	9.4	64	16	53.19
16	1	2	8	9.4	64	16	106.38

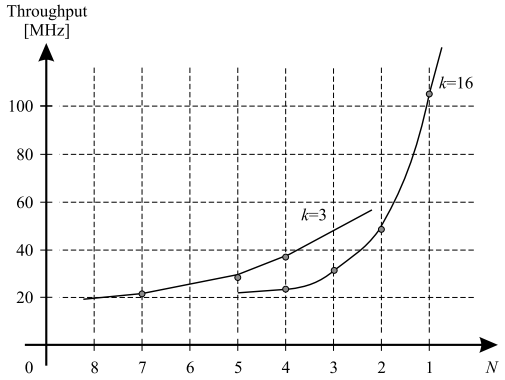

Fig. 26. Throughput as a function of chosen folding factor.

Using the data from Table 2, the graphical representation in Fig. 26 that describes the throughput as a function of chosen folding factor, is generated.

The increasing of throughput is achieved by decreasing of folding factor.

7. Configurable folded array as H.264/AVC deblocking filter (implementation and comparison)

Traditionally, block-based coding techniques partition the image into non-overlapping square blocks. To reduce the blocking artifact, the deblocking filter is adopted in H.264/AVC to improve both objective and subjective video quality, especially in the low-bit rate or highly compressed environment. The choice of filtering outcome depends on the boundary strength and the gradient of image samples across the block boundary. Many solutions for deblocking filters have been proposed [2,3,16,17]. In order to demonstrate the design of the deblocking filter, which is based on the proposed configurable folded architecture from Fig. 21 and embedded in H.264/AVC encoder/decoder [16,17], we use deblocking method with 5 different FIR filter modes, given in [3]. Filter modes are: mode 4, strongest filter (7-tap FIR filter); mode 3, strong filter (7-tap FIR filter); mode 2 (5-tap FIR filter); mode 1 (5-tap FIR filter); mode 0, weak filter (3 or 5-tap FIR filter). Coefficients for all filtering modes are within the range 1–4 ($m_C = 3$).

According to filtering modes and coefficient length, we have chosen the folded array with $k = 3$ and $y = 13$, and maximal folding factor $N_{MAX} = 7$ (bold values indicated in first row of Table 1) that can cover all previously mentioned filtering modes. Configuration abilities for the chosen folded array are given in Table 2. Due to (16), 3 tap filter is realized as 4 tap filter with $c_3 = 0$.

To evaluate the accuracy and efficiency of the proposed architecture we described the proposed design in VHDL, at RTL level, which is synthesizable. We have evaluated mentioned folded array for two cases. The first is with CBSM (Fig. 22), while the other is with CBSM+ (Fig. 23). It should be noted that the reducible folding factor is employed in the second case. According to the JVT verification model [18], a C-program model of deblocking filter was also developed to generate input simulation vectors. For the sake of comparison, we give Table 3 that contains implementation results from [16] and [17], as well as our results.

Folded architecture with CBSM, does not provide folding factor reduction, and filters all edges using $N_{MAX} = 7$ filter regardless to the filtering mode. The number of cycles per macro block (MB) required for this architecture is constant (Table 3). The architecture with CBSM+ provides the folding factor reduction, thus the number of filtering cycles depends on video content. Table 3 gives the best, the worst and the number of cycles per MB obtained for well known Foreman video sequence. The best and the worst cycles per MB values are obtained for hypothetical video sequences where all edges are filtered using 4-tap and 7-tap filters, respectively.

Table 3 shows that proposed architectures have more than 10 times smaller gate count than architec-

Table 3
Comparison of H.264/AVC deblocking filters

	Yuwen Huang's Arch.	Bin Sheng's Arch.	Our arch. with CBSM	Our arch. with CBSM+
Tech.	0.25 μm	0.25 μm	FPGA VirtexE	FPGA VirtexE
Freq. (MHz)	100	100	100	100
Gate count (in K)	20.66	24.00	1.61[b]	1.78[b]
Cycles/MB	614	446	7552	Best – 4480; Worst – 7552; Foreman – 5572
AT[a]	12685	10704	12158	9918
QCIF	–	–	161.8	181.3
CIF	–	–	40.9	44.8
4CIF	–	–	9.7	11.4
HDTV	45.2	62.3	4.3	5.1

*** For QCIF (176×144), CIF (352×288), 4CIF (704×576), HDTV (1280×720) given values are in (fps). **** Verilog code for Huang's and Shengs architectures was validated using 0.25 μm CMOS cells library [16,17].

[a] For area–time (AT) measure we take (Gate count) × (Cycl./MB).

[b] The gate count is obtained as "equivalent gate count" from Xilinx WebPack implementation report.

tures proposed in [16] and [17]. Gate count reduction is achieved at cost of time. Our architectures have approximately 10 times greater number of cycles per MB. Proposed architecture with CBSM has nearly the same AT value as the architecture proposed in [16], but 12% worse AT value than [17]. Folded array that exploits reducible folding factor (with CBSM+) has 21% better AT value than [16], and 7.3% better than [17]. It should be noted that configurable folded array with $k = 3$, $y = 13$, $N_{MAX} = 7$ and CBSM can meet the requirement for real-time deblocking of video sequences in CIF (352×288, 30 fps) at 74 MHz, while the requirement can be met at 67 MHz with reducible folding factor employment (CBSM+).

8. Concluding remarks

We considered the synthesis of configurable folded FIR filter architecture. In order to enable the successful application of folding technique, we transformed traditional bit-plane architecture, and involved dynamic operation assignment. The idea of mapping different operations on the different hardware units in the processing array structure was successfully implemented on the folded bit-plane processing array. Thus, the synthesis of folded bit-plane processing array for FIR filtering was carried out. Configuration possibilities were well explored and encompassed by the application of folding technique. The proposed folded architecture supports on-the-fly configuration of the number of taps and the coefficient length. The architecture provides flexible computation and offers the possibility of increasing the folded system throughput, by reducing the number of operations performed on single functional unit, when system performs the computation with reduced number of taps or coefficient length. FPGA implementation results were presented to illustrate configuration abilities, as well as to show design trade-offs related to the occupation of chip resources and achieved throughputs. A wider application area and finding of suitable area–time trade-offs are provided for the bit-plane architecture involving the possibility of filter configuration through the application of folding technique. The area–time trade-offs for configurable folded array were exploited in design of H.264/AVC deblocking filter for embedded mobile computing devices. It resulted in deblocking filter design with low-gate count which meets the requirement of real-time deblocking in target applications. The array is restricted for the folded factor at cost of time. In deblocking application, more than 10 times smaller gate count, in respect to designs in [16] and [17], is achieved by nearly 10 time increased number of cycles per MB. The overall product of gate count and number of cycles per MB for the proposed architecture with reducible folding factor is slightly better than [16] and [17]. It makes our platform more efficient for embedded mobile computing applications where computational time, based on image format, is not of primary importance.

References

[1] L. Paulson, L. Garber, Reconfiguring wireless phones with adaptive chips, IEEE Computer 36 (9) (2003) 9–11.

[2] P. List, A. Joch, J. Lainema, G. Bjontegard, M. Karczewicz, Adaptive deblocking filter, IEEE Transactions on Circuits and Systems for Video Technology 13 (7) (2003) 614–619.

[3] Z. Yu, J. Zhang, Video deblocking with fine-grained scalable complexity for embedded mobile computing, International Conference on Signal Processing 2 (September) (2004) 1173–1178.
[4] I. Richardson, H.264 and MPEG-4 Video Compression – Video Coding for Next Generation Multimedia, John Wiley and Sons, Inc., New York, 2003.
[5] D. Reuver, H. Klar, A configurable convolution chip with programmable coefficients, IEEE Journal of Solid State Circuits 27 (7) (1992) 1121–1123.
[6] Y.-C. Lin, F.-C. Lin, Classes of systolic arrays for digital filtering, International Journal of Electronics 70 (4) (1991) 729–737.
[7] I. Milentijevic, M.S. Stojcev, D. Maksimovic, Configurable digit – Serial convolver of type F, Microelectronics Journal 27 (6) (1996) 559–566.
[8] P. Corsonello, S. Perri, G. Cocorullo, Area–time–power tradeoff in cellular arrays VLSI implementations, IEEE Transaction on Very Large Scale Integration (VLSI) Systems 8 (5) (2000) 614–624.
[9] R. Lin, Reconfigurable parallel inner product processor architectures, IEEE Transactions on VLSI Systems 9 (2) (2001) 261–272.
[10] Robert Hawley, Bennett Wong, Thu-ji Lin, Joe Laskowski, Henry Samueli, Design techniques for silicon compiler implementations of high-speed FIR digital filters, IEEE Journal of Solid-State Circuits 31 (5) (1996).
[11] K.K. Parhi, VLSI Digital Signal Processing Systems (Design and Implementation), John Wiley and Sons, Inc., New York, 2000.
[12] T. Noll, Semi-systolic Maximum Rate Transversal Filters with Programmable coefficients, in: Proceedings of Workshop on Systolic Architectures, Oxford, 1986, pp. 103–112.
[13] I. Milentijevic, V. Ciric, O. Vojinovic, T. Tokic, Folded Semi-Systolic FIR Filter Architecture with Changeable Folding Factor, Neural, Parallel and Scientific Computations 10 (2) (2002) 235–247.
[14] I. Milentijevic, V. Ciric, T. Tokic and O. Vojinovic, FPGA Implementation of Folded FIR Filter Architecture with Changeable Folding Factor, Facta Universitatis, Ser. Electronics and Energetics 15(3), pp. 451–464. University of Niš, Yugoslavia.
[15] I.Z. Milentijevic, V. Ciric, T. Tokic and O. Vojinovic, Folded Bit-Plane FIR Filter Architecture with Changeable Folding Factor, DSD 2002, in: Proceedings of EUROMICRO – Digital System Design, Dortmund, Germany, September 2002, pp. 45–52.
[16] Y. Huang, T. Chen, Architecture Design for Deblocking Filter in H.264/AVC, in: Proceedings of ICME, Baltimore, Maryland, USA, July 2003, pp. 692–696.
[17] B. Sheng, W. Gao, D. Wu, An Implemented Architecture of Deblocking Filter for H.264/AVC, in: Proceedings of International Conference on Image Processing (ICIP), 2004, pp. 665–668.
[18] JVT software JM10.2, January 2006.

Vladimir M. Ciric is a teaching and research assistant in the Faculty of Electronic Engineering at the University of Nis, Serbia. Both B.S. and M.Sc. degrees he received from Faculty of Electronic Engineering, University of Nis 2001 and 2005, respectively, where he currently attends the Ph.D. studies. His research interests include computer architectures, fast arithmetic, digital image processing and video coding.

Ivan Z. Milentijevic is an Associated Professor at Faculty of Electronic Engineering, University of Nis, Serbia. He received B.S. in Electrical engineering and M.Sc. and Ph.D. degree in Computer Science from the Faculty of Electronic Engineering in 1989, 1994 and 1998, respectively. His research interests include computer architecture, parallel processing, fast and fault tolerant arithmetic, digital signal processing and computer science education. He has published 15 journal papers and coordinated and managed 2 international projects. Currently he is the head of Computer Science Department at University of Nis.