FACTA UNIVERSITATIS (NIŠ)
Series: Electronics and Energetics vol. 8, No.2 (1995), 191–210

DIGIT-SERIAL SEMI–SYSTOLIC CONVOLVER

Ivan Milentijević, Mile Stojčev and Dejan Maksimović

Abstract. We focus our attention on convolution of time discrete and digital signals, as one of many compute–bound problems that can benifit from systolic approach. H.T. Kung in [1] presents a family of semi–systolic (B1, B2 and F) and pure–systolic designs for the convolution problem. The semi–systolic architecture of type F is used as a basic structure for synthesis of the digit–serial convolver described in this paper. With a goal to transform this architecture into a digit–serial one, it was necessary to modify it. In essence, the proposeded modification primary relates to involving digit–serial processing elements (PEs) for multiplication and addition, instead of bit–parallel, as basic constituents of the convolver. The modified structure is characterized by the digit–serial processing in pipeline fashion, the reduced size of hardware in respect to the bit parallel version, and the feeding of input data without dummy values, i.e. PEs are fully utilized. Digit–serial PEs are implemented in LSD-frst (Least Significant Digit) integer arithmetic with the two's complement number representation.

1. Introduction

Very large–scale and wafer–scale integration technologies let us fabricate large, complex system on silicon. Systolic arrays as the typical representatives of VLSI/WSI ICs are widely used nowdays in order to achieve real–time signal and image processing (FIR, IIR filtering, 1-D, 2-D convolution etc.), matrix arithmetic (matrix–vector multiplication, matirix–matrix multiplication etc.) and non–numeric applications (pattern matching, searching, sorting etc.). Systolic array processors generally consist of a regular array with simple and nearly identical processing elements (PEs) where the data are locally communicated and rhythmically operated in a pipeline fashion. Such structures are usually realized as dedicated (algorithm specific) architectures, and are good candidates for implementation of parallel processing algorithms on a VLSI chip [2].

Manuscript received June 21, 1995.

The authors are with University of Niš, Faculty of Electronic Engineering, Beogradska 14, 18000 Niš, Yugoslavia.

We focus our attention on convolution of time discrete and digital signals, like one of many compute–bound computations that can benifit a lot from the systolic approach. The convolution problem has been the object of extensive theoretical research. H.T. Kung in [1] presents a family of semi–systolic (B1, B2 and F) and pure–systolic (R1, R2, W1 and W2) designs for the convolution problem. All solutions proposed by H.T. Kung are based on the implementation of bit–parallel arithmetic. This implies that a high hardware complexity is needed for their realization. Danielsson in [3] presents serial/parallel convolvers which utilize systolic arrays where the basic cell is a full adder and a basic structure is the serial/parallel multiplier. In bit–serial approach, contrary to the previous, at a time one input bit is processed. Consequently, these structures are convinient for low speed applications, and they are characterized with the smallest required area, interconnections and pin–out. Recently in [4, 5] Hartley and Corbett have proposed a digit–serial processing technique for realization of systolic arrays. In [5] convolution array of type R1 and W1, as proposed by Kung [1], are described. In [4] Hartley and Corbett present a functional level design for the digit–serial implementation of several arithmetic operators including multiplier, magnitude comparator and FIR filter as a complex one. In digit–serial computation data words are divided into digits and transmitted serialy between the operators one digit at a time. Hardware complexity of VLSI digit–serial circuits lies between bit–parallel and bit–serial approach. These systems are ideal for moderate speeds. Several bit–parallel semi–systolic arrays intended for convolution have been described in [6,7,8,9]. Common feature of these architectures is programmability of coefficients. This is very important for applications in adaptive filter systems. Parallel–In/Serial–Out structure is described in [6]. The main shortcomings of this sturcture are: adders to accumulate the outputs of the tap multipliers are needed; increasing wordlenght of the accumulated signal; long signal paths, and complicated wiring. Accumulation free structure is presented in [7]. Instead of separate adders, free inputs in the top row of array multipliers are used for accumulation. The drawbacks concernig wordlenght and wiring complexity remain. In bit–plane structure [8] the partial products of the multipliers are resolved. This results in highly regularity and simple interconnections. Modified bit–plane structure is described in [6]. In contrast to [8] two bits of each coefficient are processed per modified bit–plane. The modified bit–plane structure has been used as a concept for realization of the configurable convolution chip [9]. The systolic field supports a configuration during operation in a number of taps and coefficient wordlenght.

The paper is organized as follows. Section 2 deals with convolution. In Section 3 we briefly discuss the basic principle of digit–serial processing.

In Section 4 the hardware structure of the digit–serial multiplier, as a basic building block of the convlolver, is described. Synthesis and principle of operation of the digit–serial convolver of type F are explained in Section 5. Details concerning data–flow and precise timing are given also. Results of the logic simulation are presented in Section 6. Section 7 gives the conclusisons.

2. Convolution

The convolution of two sequences of numbers is a central problem in signal processing. Digital filtering is indeed the convolution of a fixed sequence – the discretized impulse response of the filter – and the signal sequence. For specificity we define the convolution as follows: for given sequence of weights $\{A_1, A_2, \dots, A_k\}$ and input sequence $\{X_1, X_2, \dots, X_n\}$, the output sequence $\{Y_1, Y_2, \dots, Y_{n-k+1}\}$ is defined as

$$Y_i = A_1 X_i + A_2 X_{i+1} + \cdots + A_k X_{i+k-1}\,. \tag{1}$$

The convolution problem is compute–bound, since each input X has to be multiplied by each of the k weights. An interesting solution proposed by Kung [1] solves this problem using a semi–systolic array (SSA) known as design F (Fig.1). Design F operates in a bit parallel manner. Having in mind the specific operative prerequisities, in the text that follows, we will proceed with a description of the design F implemented in a digit–serial technique. Let us note that designs R1 and W1, proposed by H.T. Kung [1], are already redesigned for digit–serial arithmetic and are described by Corbet and Hartley in [5].

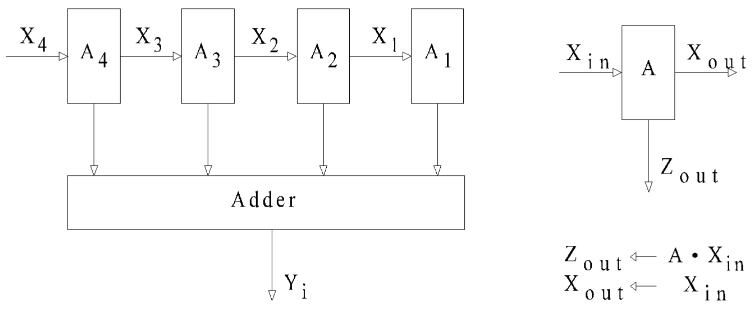


Fig. 1. Semi–systolic design of type F

3. Digit–serial processing

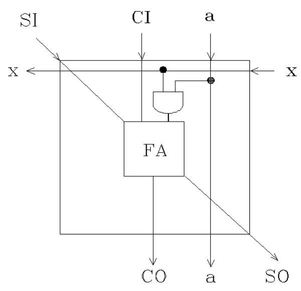
In digit–serial arithmetic each data word is divided into several digits. The following two types of arithmetic can be distinguished: most significant digit (MSD)–first arithmetic (on–line arithmetic) [10] and least significant digit (LSD)–first arithmetic [4,5]. MSD arithmetic may be used in preference to LSD–first arithmetic when floating point computation is required, where division or square root operations are used or where tight–feedback loops are involved, such as in IIR filters [5]. The disadvantage of on–line arithmetic is increased size of the computational elements and the overhead of data conversion. This paper addresses the convolution problem. In convolution, only multiplication and addition are used as operations, and tight–feedback loops are not involved. In this kind of application, bearing in mind primary the hardware complexity of SA structure, and design feasibility of PEs for multiplication and addition, LSD–first arithmetic is more efficient. Keeping in mind these general remarks, we will use digit–serial processing technique, with two's complement number representation and LSD–first integer arithmetic. Denote with W the number of bits per word (i.e. word size), and with D the number of bits per digit (i.e. digit size). Each word consists of $W/D = \alpha$ digits, where α is an integer, and the following condition $1 \leq D \leq W$ is valid. This means that for each data word processing α cycles are required. The time period of α cycles will be called a sample period (SP). Data words are transfered between the PEs over D data lines in digit–serial fashion. During digit serial processing, there is no gap between the last digit of a previos word and the first digit of a current word. This implies that some control mechanism has to be involved in order to point at the end of one word and to the begining of the next one. Therfore, we introduce periodical control signals Ci $(i = 1, \ldots, \alpha)$. The control signal Ci is active (high logic level) only during the i–th clock period of SP. For more details about digit–serial processing see [4,5].

4. Digit–Serial Multiplier

In this paper we use a fully parallel carry–save multiplier as the basic structure for the proposed digit–serial multiplier. With the aim to adapt a fully parallel multiplier for digit–serial processing we involve some modifications in its structure. Primary, modifications relate to the fact that instead of an array of $W \times W$ we use $W \times D$ array of basic cells, two stage pipelinig and folding tecnique. Multiplier design with similar structure has been already described by Hartley and Corbett in [4].

The structure of the basic cell of the digit–serial array multiplier is given in Fig. 2.

Different algorithams for multiplication (shift–and–add, Guild's, carry–save and other algorithms) can be implemented using such cells. We have adopted carry–save algorithm, as a convinient solution, because by its nature it allows us to break the multiplication process after processing of one digit and to continue with the multiplication process using carries and sums generated in the previos step. The structure of the 4–bit parallel carry save multiplier (CSM) implemented as a rectangular array of basic cells is given in Fig. 3a.



$$SO = x \cdot a \oplus SI \oplus CI$$

$$CO = SI \cdot CI + x \cdot a \cdot SI + x \cdot a \cdot CI$$

Fig. 2. Basic cell
Legend: FA–stands for full adder, SI–sum input, CI–carry input SO–sum output, and CO–carry output

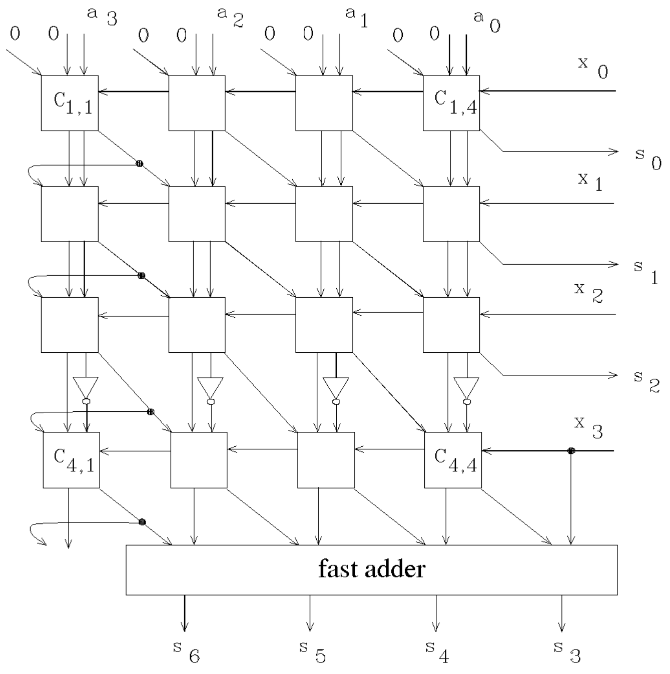


Fig. 3a. The 4–bit parallel carry–save multiplier

The array pictured in Fig. 3a multiplies two parallel four bit numbers in two's complement representation according to the algorithm given on Fig. 3b.

$$\begin{array}{cccccccc}
 & & & \multicolumn{4}{c}{a_3a_2a_1a_0 \times x_3x_2x_1x_0} & \\
\cline{4-7}
 & & a_3x_0 & a_3x_0 & a_2x_0 & a_1x_0 & (a_0x_0) & \to S_0 \\
 & & a_3x_1 & a_2x_1 & a_1x_1 & a_0x_1 & & \\
\cline{3-6}
 & PS_3^1 & PS_3^1 & PS_2^1 & PS_1^1 & (PS_0^1) & \to S_1 & \\
 & a_3x_2 & a_2x_2 & a_1x_2 & a_0x_2 & & & \\
\cline{2-5}
PS_3^2 & PS_3^2 & PS_2^2 & PS_1^2 & (PS_0^2) \to & S_2 & & \\
\bar{a}_3x_3 & \bar{a}_2x_3 & \bar{a}_1x_3 & \bar{a}_0x_3 & & & & \\
 & & & x_3 & & & & \\
\cline{1-4}
S_6 & S_5 & S_4 & S_3 & & & &
\end{array}$$

Fig. 3b: Multiplication algorithm implemented in array from Fig. 3a

The bits of operand $A = a_3a_2a_1a_0$ are passed vertically by columns, while bits of operand $X = x_3x_2x_1x_0$ are passed across the rows from right to the left. In the cell $C_{i,j}$ the bitwise product is formed and added to the carry–out from cell $C_{i-1,j}$ and the sum–out of cell $C_{i-1,j-1}$. The sum–output of the addition is passed to the cell $C_{i+1,j+1}$ whereas the carry–output, having double weight, is passed to the cell $C_{i+1,j}$. The least significant parts of the product terms s_0, s_1, s_2, are obtained on the right side of the array. The most significant parts of the product terms s_3, s_4, s_5, s_6 are obtained after summing of carries and sums from the last row of the array. Two's complement arithmetic operations are implemented into array according to the following rules:

(i) sign extension for each row of cells is performed before the sums being shifted right and passed to the row below,

(ii) bearing in mind that high order bit of the operand X has negative weight, the final row of the arrray must subtract rather than add the partial product Ax_3. This is accomplished by inverting operand A,

(iii) the sign bit of the operand X is fed–in simultaneously both into the array and to the fast adder as the least significant carry–in bit.

It is obvious that parallel multiplication of two operands of size W can be implemented on an array of $W \times W$ cells to which fast adder is attached.

If data processing in the fast adder is not taken into account, carries and sums ripple down to the W cells.

Using folding technique it is possible to modify the parallel multiplier into a digit– serial one. Multiplication of the word of size W with the digit of size D can be organized as an array of $W \times D$ cells. Repeating this activity α, times multiplication of $W \times W$ bits can be done. Carries and sums of the D–th row, obtained in i–th clock cycle, are latched. In the $(i+1)$–th cycle, latched carries and sums are used as inputs of the array.

Let us explain the principle of operation of the digit–serial multiplier for case $W = 16$ and $D = 4$ ($\alpha = 4$) pictured in Fig 4. We assume that 16–bit operand A is preloaded into register RA and is available in a parallel form. At inputs $x_0, \dots, x_3$ during each clock cycle one digit of the operand X_k is fed–in (LSD–first). We will consider multiplication of operand A, with a sequence of words $\{X_i\}$ $i = 1, \dots, k$, starting with time interval T_1 (see Fig 5). In T_1 at inputs $x_0, \dots, x_3$ the LSD of word X_1, denoted as x_{11}, is present. During T_1 we reset latch L_R. Trough this, all carry and sum inputs of the array are set to logical zero, and new multiplication process can start. In T_1 the word A is multiplied with digit x_{11}. LSD of the product, product term p_{11}, is generated on the right side of the array and is stored into the latch L_L. During T_2, p_{11} is available at its output pins $s_0, \dots, s_3$. At the end of T_1 carry and sum outputs are stored into L_R. In order to continue the multipliction of the word A with digit x_{12}, during T_2, carry and sum bits are used as inputs. In T_2 and T_3 multiplication is done in the same manner. To the $W \times D$ array of cells an additional basic cell, denoted as C_A, is attached. Its outputs are defined as:

$$SO_{CA} = C4 \cdot x_3 \oplus SO_{C4,16} \,,$$
$$CO_{CA} = C4 \cdot x_3 \cdot SO_{C4,16} \,.$$

During time intervals T_1, T_2 and T_3 the control signal $C4 = \{0\}$ and $SO_{CA} = SO_{C4,16}$. In T_4 the control signal $C4$ is active ($C4 = \{1\}$), and the array performs multiplication of the word A with x_{14}. According to the described algorithm the following activities are done:

a) for cells $C_{4,i}$ ($i = 1, \dots, 16$) bits of the operand A are complemented by XOR gates.

b) the most significant bit of digit x_{14}, available on input x_3, is added in cell C_A.

After this, in $T4$ the product term p_{14} is calculated and is latched into L_L. During T_5, p_{14} is present at the outputs $s_0, \dots, s_3$. In this manner generation of the low order word of at output of the latch L_L is completed.

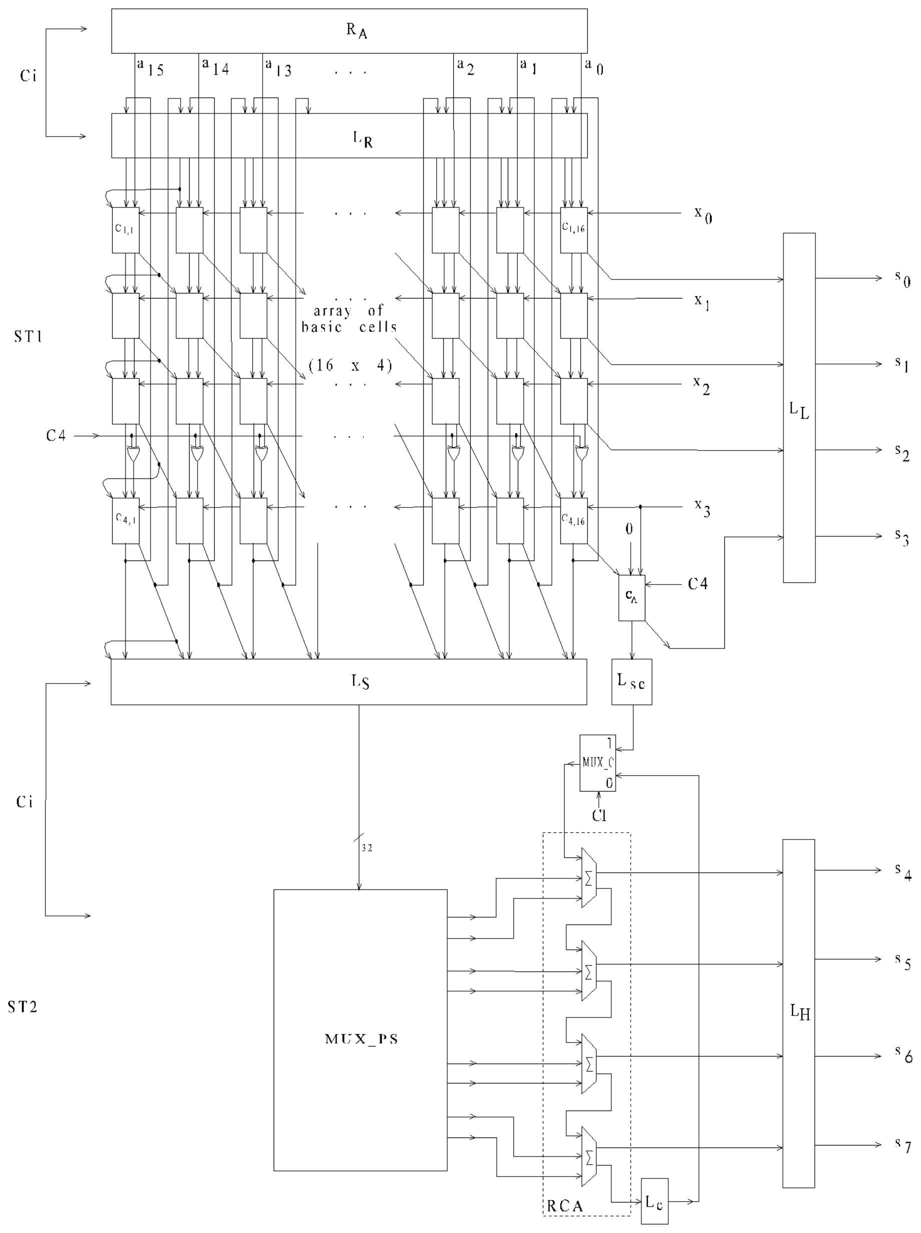


Fig. 4. DSM for $W = 16$ and $D = 4$

Sums and carries generated in T_4 at the outputs of the last row of cells are latched into L_S. At the same time the carry bit generated in the cell C_A

is latched into L_{SC}. In order to obtain a high order word of the product it is necessary to add carries and sums already latched into L_S and L_{SC}. To do this in digit–serial manner a 4–bit ripple carry adder (RCA), multiplexer block 32 to 8 (mux_PS), multiplexer 2 to 1 (MUX_C) and latch L_C are used. According to the described algorithm and timing diagrams given in Fig. 5, the multiplication will be completed for the next four clock cycles.

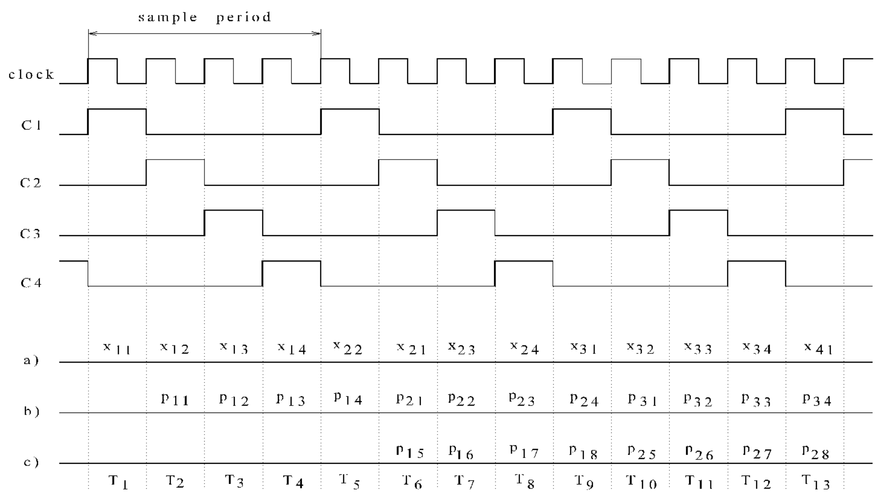

sample period
clock
C1
C2
C3
C4
a)
x11 x12 x13 x14 x22 x21 x23 x24 x31 x32 x33 x34 x41
b)
p11 p12 p13 p14 p21 p22 p23 p24 p31 p32 p33 p34
c)
p15 p16 p17 p18 p25 p26 p27 p28
T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13

Fig. 5. Timming diagrams of operands and result availability and control signals $C1$, $C2$, $C3$ and $C4$ for $\alpha = 4$:
(a) digit–availability of operands X_i at inputs $x_0, \dots, x_3$;
(b) digit–availability of the low order parts of the products at outputs $s_0, \dots, s_3$;
(c) digit–availability of the high order parts of the products at outputs $s_4, \dots, s_7$.

During T_5, T_6, T_7 and T_8 digits p_{15}, p_{16}, p_{17} and p_{18}, which are constituents of the high order word of the product, are calculated respectively. At the end of each clock cycle a coresponding digit is latched into L_H. This means that digits $p_{15}, \dots, p_{18}$ become available at outputs $s_4, \dots, s_7$ in T_6, T_7, T_8 and T_9, respectively.

As it was mentioned, the proposed digit–serial multiplier is implemented as two–stage (ST1, ST2) pipeline system. This fact allows us to to start new multiplication process in clock cycle T_5, meaning that the array (stage ST1) in clock cycles T_5, T_6, T_7 and T_8 multiplies the operand A with digits x_{21}, x_{22}, x_{23} and x_{24} which belong to operand X_2. In the same time in the stage ST2 the high order word of the product of the previos multiplication is calculated. Using two–stage pipelining, as it is proposed in the given scheme, overlapping of two successive multiplication process can be achieved. Block scheme for DSM with word size W and digit size D is given in Fig. 6.

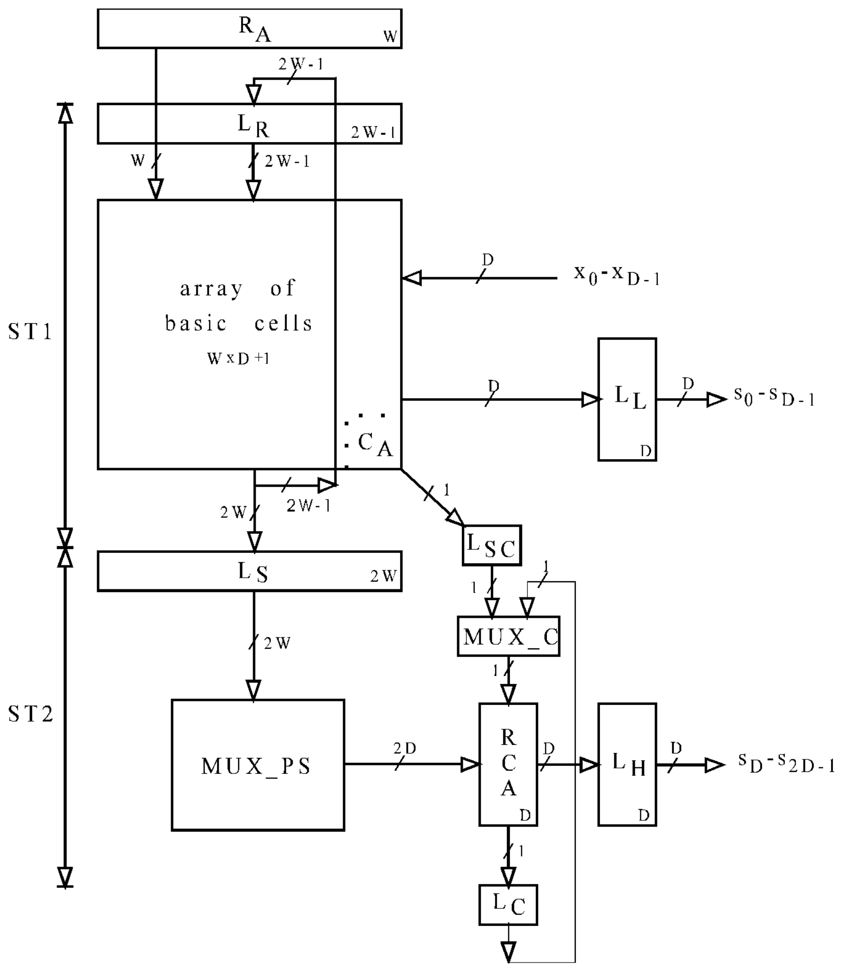


Fig. 6. Block scheme for DSM with word size W and digit size D

5. Digit–serial Convolver

The semi–systolic architecture of type F [1], given in Fig. 1, is used as the base structure for synthesis of digit–serial convolver described in this paper. As can be seen from Fig. 1, coefficients, as elements of weight vector, are preloaded and, are static during computation. Input values X_i are shifted for one cell right in each cycle, so the convolution problem is one that computes, according to (1), the inner product of the weight vector and section of input vector it overlaps. In order to generate a new Y_i, all product terms are collected and summed. The structure F, proposed in [1], is based on a bit–parallel arithmetic. With a goal to implement this architecture into a digit–serial one, it's modification was inevitable. In essence, the modification primary relates to involving the digit–serial processing elements for multiplication and addition, as the basic constituents of the convolver. This allows us to reduce the hardware significantlly, with respect to a bit–parallel version.

The modified structure is characterized by a digit–serial processing in a pipeline fashion and data inputing without dummy values, i.e. the processing elements are fully utilized. The structure of the proposed digit–serial convolver DSC is shown in Fig. 7. It consists of k multiplier cells (MC) and a pipeline adder tree. For data transfer between MC cells D data lines are needed. The DSC structure with k MC cells ($k = 4$, $W = 16$ and $D = 4$) is pictured in Fig. 8. As can be seen from Fig. 8, the pipeline adder tree is composed of three adders, denoted as $A11$, $A12$ and $A21$. Principle of operation is based on LSD–first integer arithmetic, i.e. at inputs $x_{i0} - x_{i3}$ the least significant digit of input data is fed–in first.

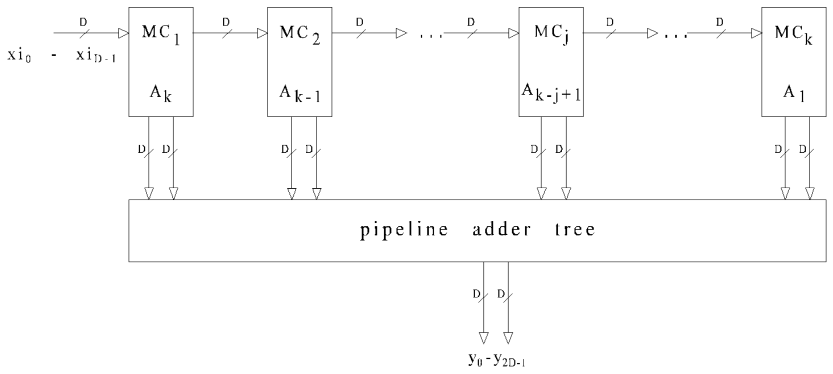


Fig. 7. Structure of DSC with k taps for word size W and digit size D

The structure of MC_j $(j = 1, \dots, k)$ is given in Fig. 9. MC_j consists of two building blocks, denoted as DSM_j and S_j. Novelty, in respect to previously described DSM_j, is feeding of the programmable coefficients A_j $(j = 1, \dots, k)$. The coefficient A_j is accepted serially into a shift register RA_j. RA_j accepts data from RA_{j-1} and transfers them to RA_{j+1}. All RA_js form a meandring shift register. After initialization the coefficient A_{k-j+1} is

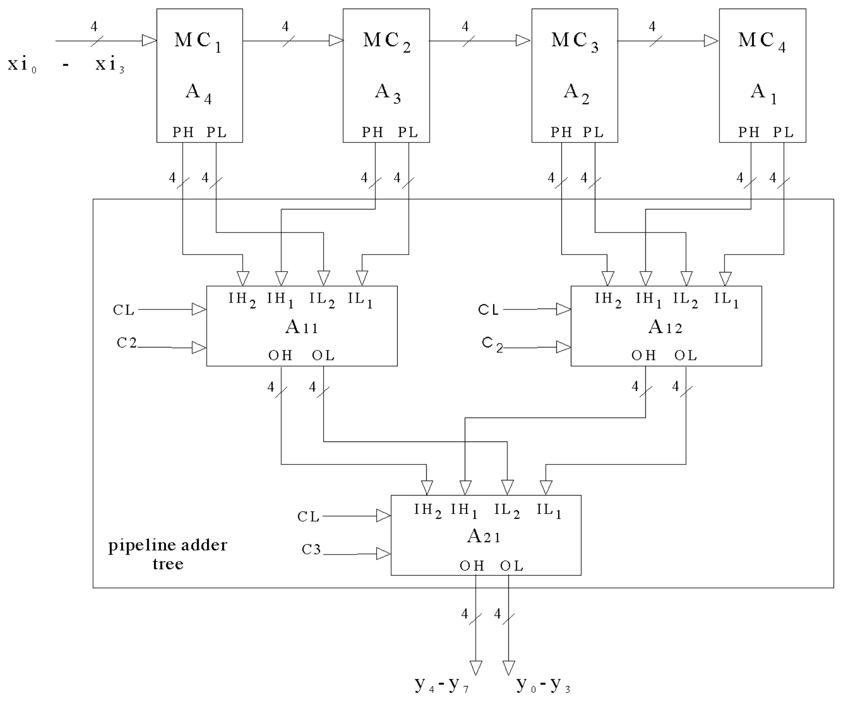
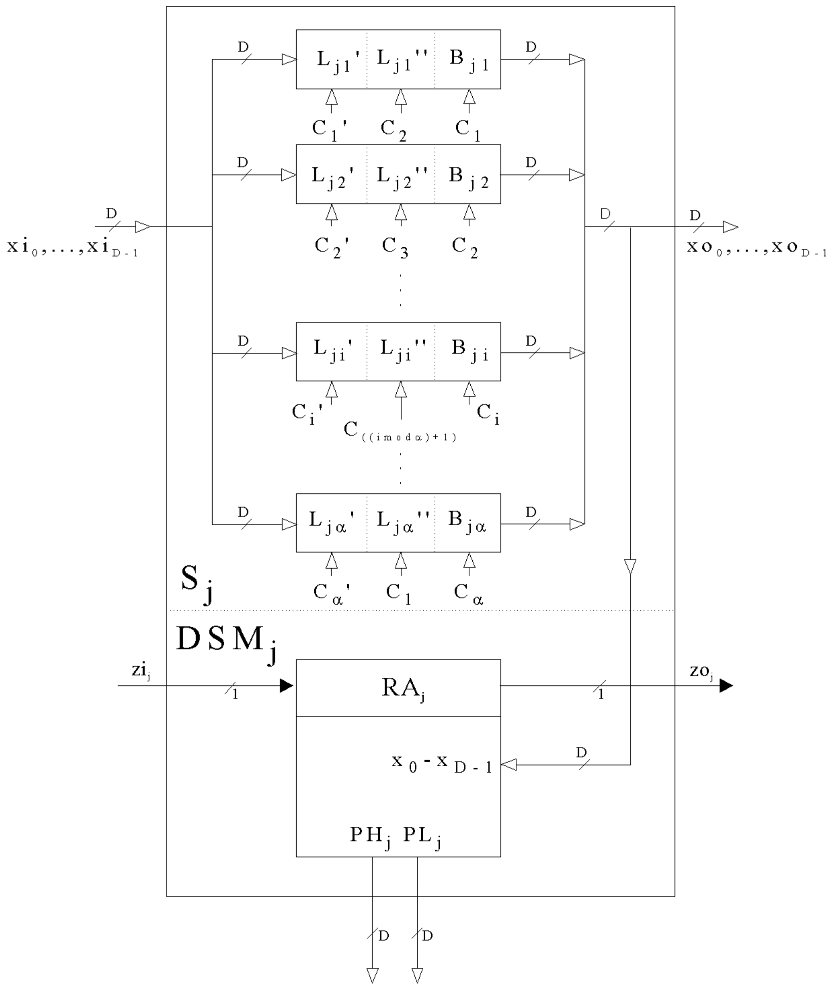


Fig. 8. DSC for $k = 4$, $W = 16$ *and* $D = 4$

stored in RA_j. During computation the coefficients are static and available in a parallel form. The function of S_j is twofold. At first, as an element for temporary storage, and secondly as a supplier of DSM_j with the corresponding digit. S_j consists of α subblocks L_{ji} $(i = 1, \dots, \alpha)$. The subblock L_{ji} is shown in Fig. 10. Sections L'_{ji}, L''_{ji} and B_{ji} are the constituents of L_{ji}. Sections L'_{ji} and L''_{ji} are identical, and they are comprised of α positive edge triggered D flip–flops. Section B_{ji} is implemented with α three–state buffers.

Let in an arbitrary clock cycle, T_p, when $Ci = \{1\}$, at inputs $x_{i0} - x_{iD-1}$ of cell MC_j the i–th digit $(i = 1, \dots, \alpha)$ of the m–th input word $(m = 1, \dots, n)$, x_{im}, is present. The subblock L_{ji}, as a constituent of the block S_j, is used as an element for temporal storing of x_{im}. x_{im} is latched into section L'_{ji} of the subblock L_{ji} by the rising edge of the control signal Ci'. With the rising edge of the control signal $C_{((i \bmod a)+1)}$ the digit stored in the section L'_{ji} is transfered into the section L''_{ji}, where it is stored temporary for next α clock cycles. By reactivating the control signal Ci $(Ci = \{1\})$, during a clock cycle T_q $(\alpha - 1$ clock cycles later in respect to $T_p)$, section B_{ji}



L_{j1}' L_{j1}'' B_{j1}
C_1' C_2 C_1
L_{j2}' L_{j2}'' B_{j2}
C_2' C_3 C_2
L_{ji}' L_{ji}'' B_{ji}
C_i' $C_{((i \bmod \alpha)+1)}$ C_i
$L_{j\alpha}'$ $L_{j\alpha}''$ $B_{j\alpha}$
C_α' C_1 C_α
S_j
DSM_j
$xi_0,\ldots,xi_{D-1}$
$xo_0,\ldots,xo_{D-1}$
D
zi_j
zo_j
1
RA_j
$x_0 - x_{D-1}$
PH_j PL_j

Fig. 9. Structure of MC_j

simultaneously sends the digit x_{im}, both to the multiplier DSM_j and to the neighbouring cell MC_{j+1}. Double buffering in the subblock L_{ji} (Fig. 10) provides L_{ji} both to accept and to send digits simultaneously. During the clock cycle T_q at inputs $DI_0 - DI_{D-1}$ of the subblock L_{ji} the i–th digit of the $(m+1)$–st input word $X_{i,m+1}$ is present. The digit is latched into the section L'_{ji} at the rising edge of the control signal Ci'. Further, all activities previously described are repeated. In Fig. 11 relevant control signals and data flow at a digit–level through blocks S_j, for the convolver with $k = 8$, $W = 16$ and $D = 4$, are pictured (shaded areas denote the intervals when the digits pass to the corresponding DSM$_j$ and to the MC$_{j+1}$, while * denotes digits present at inputs xi_0–xi_3).

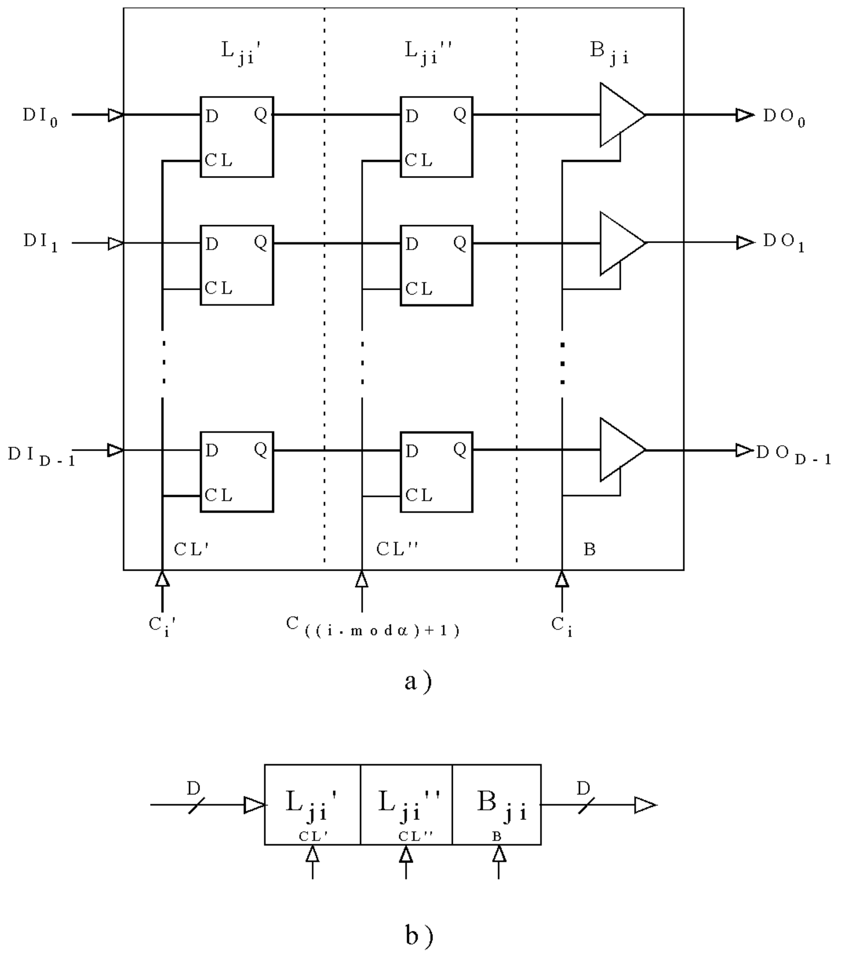


Fig. 10. Subblock L_{ji}: (a) structure of subblock L_{ji}
(b) Global scheme of subblock L_{ji}

In each cell MC_j a collection of α subblocks L_{ji} forms the block S_j. In this way, all multipliers are simultaneously fed–in with first, then with the second,..., and finally with the α–th digit of the corresponding data word. Thus, the operation of all multipliers in the convolver is synchronized, and data processing is organized in a pipeline fashion at a digit level without conflicts. The time needed to load the structure, tload, is equal to the number of clock cycles starting from the instant when the LSD of the input word X_1 is present at the inputs $xi_0 - xi_{D-1}$ of the cell MC_1, till the begining of Y_1 computation. For loading of all synchronization blocks $\alpha \cdot k$ clock cycles are nedeed. After that at the outputs PL_j $(j = 1, \ldots, k)$ the LSD of a low order word of the product is available. The LSD of a high order word is obtained at the outputs PH_j $(j = 1, \ldots, k)$ after α clock cycles.

The multilevel pipeline adder tree performs summation of the product terms obtained at outputs of the MC_j cells. The structure of a two–level adder tree, for convolver with $k = 4$, $W = 16$ and $D = 4$, is given in Fig. 8. Cells of type A are the constituents of the adder tree. As can be seen from Fig. 8, A cells at the first level are denoted with $A11$ and $A12$, while at the second level with $A21$. Detailed structure of cell A is shown in Fig. 12. The cell A can operate in two modes, ADD or L mode. The mode of operation is defined by the state of the control signal at the input ADD/L^*. For $ADD/L^* = \{1\}$, $B1$ and $B4$ act as non–inverting buffers, while the outputs of $B2$, $B3$, $B5$ and $B6$ are in a high impedance state, and thus ADD mode is selected. On the contrary, L mode is chosen. In ADD mode A cell operates as a double digit–serial adder, while in L mode it acts as a latch. Mode selection is performed during convolver's initialization. When ADD mode is selected the cell A is structured as pictured in Fig. 13. In this mode, both digit–serial adders $\mathrm{RCA_L}$ and $\mathrm{RCA_H}$ are active. In ADD mode, the principle of operation of A cells is adapted to a manner of product generation at the outputs of the MC cells. Addition of the digits of a low order (high order) word of the product is performed by $\mathrm{RCA_L}$ ($\mathrm{RCA_H}$) for α clock cycles. Operation of $\mathrm{RCA_L}$ and $\mathrm{RCA_H}$ is time overlapped. During the first clock cycle the $\mathrm{RCA_L}$ and $\mathrm{RCA_H}$ accept zero as a carry_in. For the rest, $(\alpha - 1)$ clock cycles, $\mathrm{RCA_L}$ performs addition for the rest of $(\alpha - 1)$ digits. During the last, α–th clock cycle, the $\mathrm{RCA_L}$'s carry_out is latched into the LCL. In the $(\alpha + 1)$–st clock cycle the multiplexer MUX_H selects LCL's output as a carry_in for $\mathrm{RCA_H}$. For the next $(\alpha - 1)$ clock cycles MUX_H selects LCH's output as a carry_in. In the adder tree structure, Fig. 8, all cells of type A are set in ADD mode. In general, in the first level all cells of type A are driven by the control signal $C2$. The appearance of $C2$ provides a condition for starting summation of the new products. If the cells at some arbitrary level are driven by the control signal Ci, then the cells which belong to the following level have to be driven by the control signal $C_{((i \bmod a)+1)}$. When the number of installed MC cells is equal to k, where $k = 2^r$ $(r = 1, 2, \ldots)$, then the adder tree is composed of A cells set in ADD mode, only (in Fig. 8 $k = 2^2$). In other cases tree balancing is done by setting some cells of type A into L mode. In this way a delay function is assigned to these cells. Digits present at inputs of A cells, set into L mode (Fig. 14), are transfered to cell's

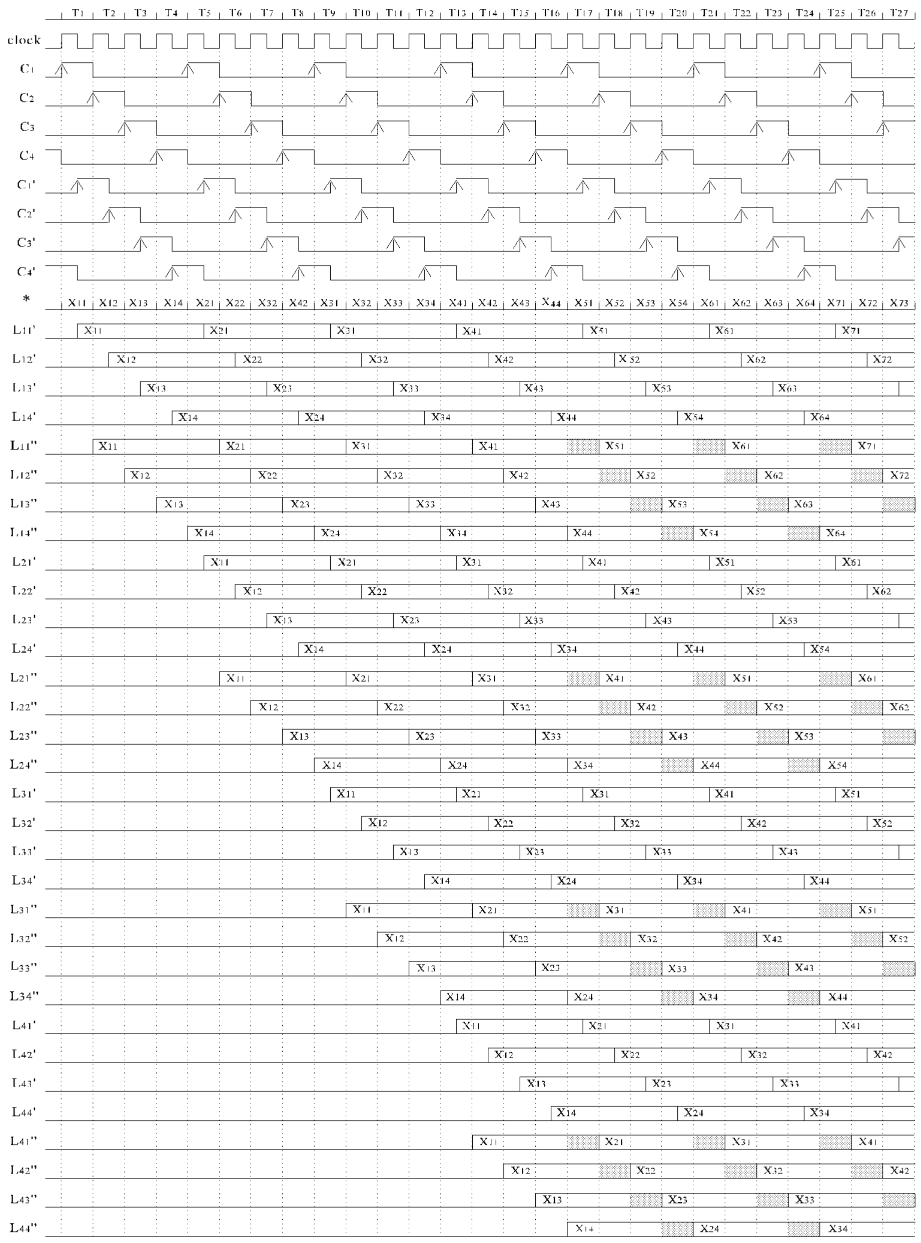


Fig. 11. Relevant control signals and data flow at a digit–level through blocks S_j for the DSC with $k = 8$, $W = 16$ and $D = 4$

outputs one clock cycle later. The convolver structure with three MC cells

($k = 3$) is sketched in Fig. 15. The cell of type A, marked with asterix, is set in mode L.

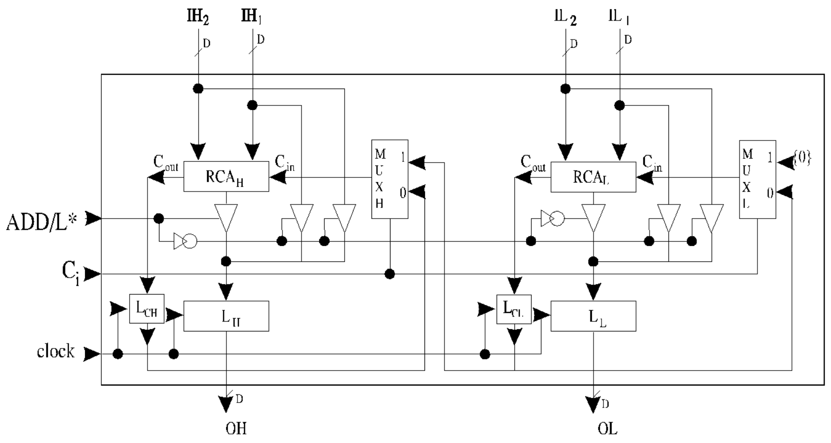


Fig. 12. Structure of cell A

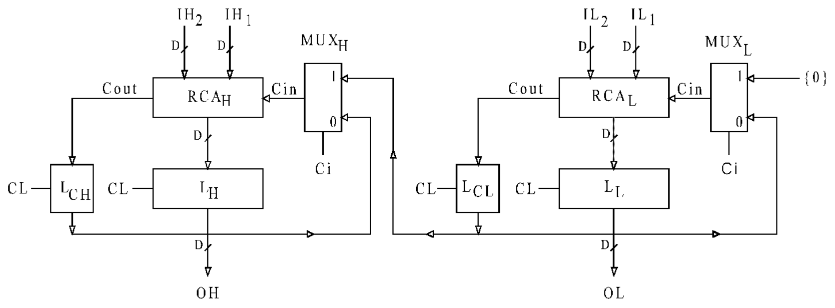


Fig. 13. Cell A in ADD mode

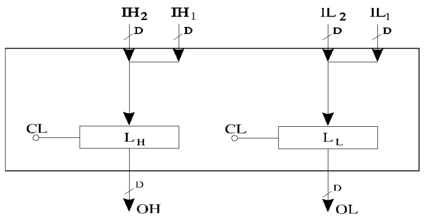


Fig. 14. Cell A in L mode

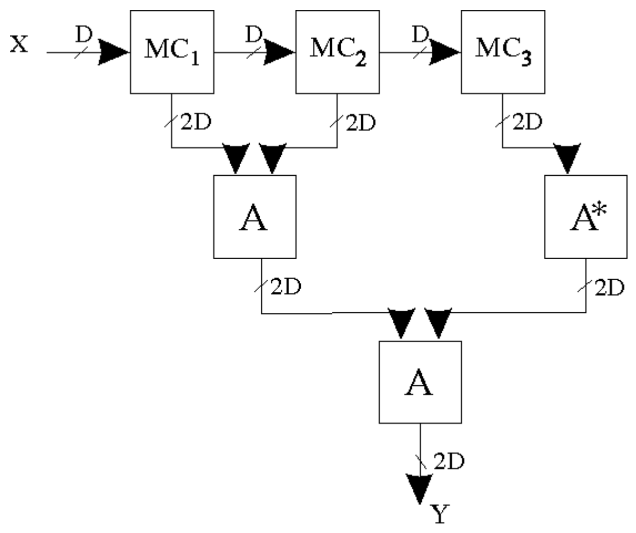


Fig. 15. DSC for $k = 3$

In general, the processing time of one digit in a previosly described pipeline adder tree is equal to $t_{add} = \lfloor \log_2(k-1) \rfloor + 1$ clock cycles.

Initial latency of the whole structure is

$$Z = t_{load} + t_{mul} + t_{add} = \alpha k + \lfloor \log_2(k-1) \rfloor + 2\,,$$

where: t_{load} – time needed to load the structure; t_{mul} – latency of the digit–serial multiplier; t_{add} – latency of the pipeline adder tree. Initial latency is a time interval, in clock cycles, starting from the instant of feeding LSD of the input word X_1, till LSD of the output word Y_1 is generated.

6. Logic Simulation

The DSC is designed in the standard cell methodology using Intel's CHMOS III Cell Library [11]. Design verification is done by the use of logic simulator HILO–3. Logic simulation is performed for several DSC configurations. Configurations were based on MCs with different word size W and different digit size D. The number of MCs (taps) is denoted with k. Simulation results for five different DSC configurations are given in Table 1. For each configuration maximal operating clock frequency is determined. Convolver's troughput rate ($F_{SP} = 1/\alpha \cdot T_{cl}$) and initial latency ($Z$[number of cycles]) are also given. Column Amax points to maximal coefficient length for given configuration of DSC, where $A\max = W - (\lfloor \log_2(k-1) \rfloor + 1)$.

Table 1

$W[b]$	$D[b]$	$W/D = \alpha$	k	$A\max$	Z	$f_{CL}\,[MHz]$	$F_{SP}\,[MHz]$
8	4	2	8	5	20	22.6	11.3
12	3	4	6	9	28	26.4	6.6
16	4	4	4	14	19	22.6	5.7
24	6	4	3	22	15	17.6	4.4
32	8	4	2	31	10	14.4	3.6

Legend: W–word size, D–digit size, α–number of digits in one data word, k number of taps, $A\max$–maximal coefficient length, Z–initial latency, f_{CL}–clock frequency, F_{SP}–sample frequency.

7. Conclusions

The bit–parallel semi–systolic architecture of type F is transformed into a digit serial one. In essence, the proposeded modification primary relates to involving digit serial processing elements (PEs) for multiplication and addition, instead of bit–parallel, as basic constituents of the convolver. The DSC is characterized by digit–serial processing in pipeline fashion, and feeding input data without dummy values, i.e. PEs are fully utilized. The choice of digit–size allows the designer to match throughput requirements to specifications. Digit–serial PEs are implemented in LSD–frst integer arithmetic with two's complement number representation. The DSC consists of two types of PE, multiplier and adder. Implementation of a digit–serial technique allows us to reduce considerably the DSC hardware structure with respect to design F realized in a bit–parallel fashion. The DSC is designed using standard cells from INTEL's CHMOS III Library. Programmibility of coefficients is provided too. Coefficients are loaded into the DSC during chip initialization in a bit–serial manner.

REFERENCES

1. H.T. Kung: *Why systolic architectures.* IEEE Computer, Vol. 15, January 1982, pp. 37–46
2. Y.S. Kung: *VLSI array processors.* Prentice Hall, Englewood Clifs, New Jersey, 1988
3. P.E. Danielsson: *Serial/Parallel convolvers.* IEEE Trans. on Computers, Vol. C–33, No. 7, July 1984, pp. 652–667
4. R. Hartley, P. Corbett: *A digit–serial processing techniques.* IEEE Trans. on Circuits and Systems, Vol. 37, No. 6, June 1990, pp. 709–719
5. P. Corbett, R. Hartley: *Designing systolic arrays using digit–serial arithmetic.* IEEE Trans. on Circuits and Systems–II: Analog and Digital Signal Processing, Vol. 39, No. 1, January 1992, pp. 62–65

6. T. Noll: *Semi–systolic maximum rate transversal filters with programmable coefficients.* Workshop of Systolic Architectures, Oxford, England, U.K., 1986, pp. 103–112

7. P.R. Cappello, K. Steiglitz: *A note on 'Free Accumulation' in VLSI filter architectures.* IEEE Transactions on Circuits and Systems, Vol. CAS–32, No. 3, March 1985, pp. 291–296

8. P.B. Denyer, D.J. Myers: *Carrry–save array for VLSI signal processing.* Proc. First Int. Conf. on VLSI, Edinburgh, Scotland, U.K., August 1981, pp. 151–160

9. D. Reurer, H. Klar: *A configurable convolution chip with programmable coefficients.* IEEE Journal of Solid–State Circuits, Vol. 27, No. 7, July 1992, pp. 1121–1123

10. M.J. Irwin, R.M. Owen: *Digit–pipelined arithmetic as illustrated by the paste up system: A tutorial.* IEEE Computer, April 1987, pp. 61–73

11. Introduction to INTEL cell–based design: *Intel Corporation.* Santa Clara, California, USA, 1986.