A Unified Approach to Implementation of Stack Filters

J. Astola[1], D. Akopian[1], S. Agaian[2], and O. Vainio[1]

[1]Signal Processing Laboratory
Tampere University of Technology
P. O. Box 553, 33101 Tampere, Finland

[2]Electrical Engineering Department
Tufts University
Medford, MA 02155

Abstract

New stack filter architectures are proposed. Known architectures are considered from new positions and combined to obtain new structures with suitable complexities and throughputs. It is shown that all stack filters can be used without changes when the input data is represented in one of the binary lexicographic codes. Stack filters are also proposed for the cases where the data are represented in multiple-value lexicographic codes. The class of pipeline-parallel structures for common stack filters is simple and modular in structure, and suitable for VLSI implementation. The time-area complexity of the proposed filters is $O(k)$, where k is the number of bits in the input signals.

1 Introduction

Many implementation architectures for stack filters have been proposed recently with different complexities and throughputs. In the threshold decomposition architecture [5], multilevel signals which belong to the set $\{0, 1, \ldots, L-1\}$ from the input window of size n are decomposed to L unit-weighted signals. A positive Boolean function (PBF) is applied to each of the unit-weighted binary data as shown in Fig. 1. The set of L PBF's creates L unit-weighted data and the output is obtained by converting the unit-weighted signals to a binary-weighted signal using arithmetic summation. A realization based on a bit-serial approach [2] performs processing on binary-weighted signals and has simple structure, but the output is obtained after $log_2 L$ steps of PBF calculations, while in the threshold decomposition architecture only one stage of PBF calculation is used. In order to reduce PBF calculation stages, input compression before the stack filter has been proposed [3]. Sample points appearing in the filter window are compressed to the integers 1 through n, where n is the size of the input window, without changing the positions and relative ranks of the sample points. However, the complexity is increased because of the comparison, addition, and subtraction circuits. Another way to increase the throughput is the representation of input data in Fibonacci p-codes [1]. Each stack filter may be determined by a PBF function or by a set of min/max operators in the real domain. For this reason, sorting structures can be also used for stack filter realization [4].

In this paper, new stack filter architectures are proposed. Known approaches are combined in order to obtain new structures with suitable complexities and throughputs. It is shown that the bit-serial approach to stack filters for binary-weighted data may be generalized for all lexicographic code representations of the input data. Binary unit-weighted, binary binary-weighted, and Fibonacci p-codes are binary lexicographic codes. Input compression can be considered as the representation of input data in a monotone code.

The bit-serial approach is generalized for the representation of input data in multiple-value lexicographic codes, and is combined with threshold decomposition in order to obtain a set of intermediate structures between the threshold decomposition architecture and architectures based on the bit-serial approach for binary-weighted data. It can be combined also with real domain domain stack filters.

2 Stack filters

Suppose that $X(t)$ is a discrete-time stochastic process, $X(t) \in Q = \{0, 1, \ldots, M-1\}$. Let us consider the set of data $\vec{X} = (X_1, X_2, \ldots, X_n)$. in the window of width n at the time t_0.

Definition 2.1. The thresholding functions $T_m(X_i)$ and $T_m(\vec{X})$ are defined as:

$$T_m(X_i) = \begin{cases} 1, & \text{if } X_i \geq m; \\ 0, & \text{if } X_i < m \end{cases}, \qquad (2.1)$$

and $T_m(\vec{X}) = [T_m(X_1), T_m(X_2), \ldots, T_m(X_n)]$.

1058-6393/95 $4.00

The binary-weighted code $(b_{ik},\ldots,b_{i1})$ of the number X_i is obtained from the usual representation:

$$X_i=\sum_{j=1}^{k} b_{ij}2^{k-j},\ b_{ij}\in\{0,1\}. \tag{2.2}$$

The unary-weighted code $(d_{i,2^k-1},\ldots,d_{i,1})$ of the number X_i is obtained from the representation:

$$X_i=\sum_{j=1}^{2^k-1} d_{ij}=\sum_{j=1}^{2^k-1} T_j(X_i). \tag{2.3}$$

We consider also other one-to-one code representations $(a_{i1},\ldots,a_{ik})_R$, obtained by applying the given rule R to the numbers X_i to determine the set of bits $(a_{i1},\ldots,a_{ik})$, $a_{ij}\in\{0,1\}$ for binary codes and the set of digits $(a_{i1},\ldots,a_{ik})$, $a_{ij}\in P$ for nonbinary codes, where P is the given set of numbers.

A Positive Boolean function (PBF) is a Boolean function whose minimum sum-of-product form is free of complemented variables. Thus a PBF is of the following form (in DNF representation):

$$f=K_1+K_2+\cdots+K_s \tag{2.4}$$

$$K_i=x_{i_1}\cdot x_{i_2}\cdot\ldots\cdot x_{i_{r(i)}} \tag{2.5}$$

where "+" denotes OR and "·" denotes AND operation.

Definition 2.2. A window width n stack filter $S_f(\cdot)\ \{0,1\}^n\to\{0,1\}$ is based on an n-variable positive Boolean function $f(\cdot)$:

$$S_f(\vec{X}(t))=S_f(\sum_{l=1}^{M}T_l(\vec{X}(t)))=\sum_{l=1}^{M}S_f(T_l(\vec{X}(t)))$$
$$=\sum_{l=1}^{M}f(T_l(\vec{X}(t))).$$

3 Lexicographic (monotone) codes

Let us consider the representation of the data values $0,1,\ldots,M-1$ in the form $\underline{c}(i)=(c_1(i),\ldots,c_p(i))$, $i=0,1,\ldots,M-1$, where $c_j(i)\in\{L_1,\ldots,L_r\}$, $j=1,\ldots,p$ and $L_1<L_2<\cdots<L_r$.

Definition 3.1. The representation $(a_i^1,\ldots,a_i^p)=\underline{c}(i)$, $i=0,1,\ldots,M-1$ is a **lexicographic** code if for any $i,j:\ 0\le j<i\le M-1$ there is an index t, $1\le t\le p$ such that $a_i^k=a_j^k$, $k=1,\ldots,t-1$ and $a_i^t>a_j^t$.

This means that the linear (magnitude) ordering of the data corresponds to the lexicographic ordering of the codewords $(a_i^1,\ldots,a_i^p)$.

Let $x_1,\ldots,x_n$ belong to any linearly ordered set $\{L_1,\ldots,L_r\}$. Then the stack filter $S_f(\cdot)$ can be directly extended to operate on these values $x_1,\ldots,x_n$ by

$$S_f(x_1,\ldots,x_n)=K_1\vee K_2\vee\cdots\vee K_s, \tag{3.1}$$

where

$$K_i=x_{i_1}\wedge x_{i_2}\wedge\cdots\wedge x_{i_{r(i)}} \tag{3.2}$$

and $\vee$ denotes the maximum and $\wedge$ denotes the minimum operation.

We also consider the following freezing rule $F_h(x)$:

$$F_h(x)=\begin{cases}L_r, & \text{if } h<x;\\ L_1, & \text{if } h>x.\end{cases} \tag{3.3}$$

Let $\vec{X}=(X_1,X_2,\ldots,X_n)$ be the current set of input data in the stack filter window and $X_i=(a_i^1\ldots a_i^k)$ is a lexicographic code representation of the samples. We consider the following mapping, depending on a function S_f :

$$\begin{pmatrix}a_1^1\\ \vdots\\ a_1^k\end{pmatrix}\begin{pmatrix}a_2^1\\ \vdots\\ a_2^k\end{pmatrix}\cdots\begin{pmatrix}a_n^1\\ \vdots\\ a_n^k\end{pmatrix}\Rightarrow\begin{pmatrix}d_1^1\\ \vdots\\ d_1^k\end{pmatrix}\begin{pmatrix}d_2^1\\ \vdots\\ d_2^k\end{pmatrix}\cdots\begin{pmatrix}d_n^1\\ \vdots\\ d_n^k\end{pmatrix} \tag{3.4}$$

where

$$d_i^1=a_i^1,\ i=1,\ldots,n,\quad h^1=S_f(d_1^1,\ldots,d_n^1), \tag{3.5}$$

and d_i^j are defined recursively:

$$d_i^j=\begin{cases}a_i^j & \text{if } a_i^m=h^m \text{ for } m=1,2,\ldots,j-1;\\ L_1 & \text{if } \exists l<j: a_i^m=h^m \text{ for } m=1,2,\ldots,l-1\\ & \text{and } a_i^l<h^l;\\ L_r & \text{if } \exists l<j: a_i^m=h^m \text{ for } m=1,2,\ldots,l-1\\ & \text{and } a_i^l>h^l;\end{cases} \tag{3.6}$$

$$h^m=S_f(d_1^m,\ldots,d_n^m), \tag{3.7}$$

that is if $\exists l: a_i^m=h^m$ for $m=1,2,\ldots,l-1$ and $a_i^l\ne h^l$, then

$$d_i^j=F_{h^l}(a_i^l) \text{ for all } j\in\{l+1,l+2,\ldots,k\},$$

where $F_h(x)$ is the freezing rule (3.3). The following recursive sequence is formed:

$$\{d_i^j\}_{i=1,\ldots,n}\to h^j\to\{d_i^{j+1}\}_{i=1,\ldots,n}\to h^{j+1}\to. \tag{3.8}$$

Theorem 2.1. If input numbers X_i are represented in a monotone code as $(a_i^1\ldots a_i^k)$, then the output of the stack filter will be:

$$S_f(X_1,\dots X_n)=\begin{bmatrix}h^1\\ \vdots\\ h^k\end{bmatrix}=$$

$$S_f\left[\begin{pmatrix}a_1^1\\ \vdots\\ a_1^k\end{pmatrix}\quad\begin{pmatrix}a_2^1\\ \vdots\\ a_2^k\end{pmatrix}\quad\dots\quad\begin{pmatrix}a_n^1\\ \vdots\\ a_n^k\end{pmatrix}\right]=$$

$$S_f\left[\begin{pmatrix}d_1^1\\ \vdots\\ d_1^k\end{pmatrix}\quad\dots\quad\begin{pmatrix}d_n^1\\ \vdots\\ d_n^k\end{pmatrix}\right]=\begin{bmatrix}S_f(d_1^1\dots d_n^1)\\ \vdots\\ S_f(d_1^k\dots d_n^k)\end{bmatrix}\tag{3.9}$$

where d_i^j is defined in (3.5)-(3.7).

The expression (3.9) means that stack filtering of the data from an arbitrary value range can be replaced by sequential stack filtering of the numbers from the chosen fixed range of values. By this reason, if there exists a stack filter implementation for the given value range of input data, it can be used also with input data from an arbitrarily larger range. Thus we can combine the bit-serial (number-serial) approach with an arbitrary type of stack filter implementation. The appropriate algorithm can be represented in the following way:

Algorithm 3.1.
For $i=1,\dots,n$ **do in parallel**
set $t_i=1$
For $j=1,\dots,k$ **do in sequential**
For $i=1,\dots,n$ **do in parallel**
Begin
If $t_i=1$ **then** $d_{ij}=a_{ij}$
else $d_{ij}=d_{ij}$;
$h_j=S_f(d_{1j},d_{2j},\dots,d_{nj})$
If $t_i=1$ **and** $h_j>a_{ij}$
then set $d_{ij}=L_1, t_i=0$
If $t_i=1$ **and** $h_j<a_{ij}$
then set $d_{ij}=L_r, t_i=0$
END

Here $S_f(d_{1j},\dots,d_{nj})$ is the function of type (3.1)-(3.2) or a real domain stack filter for a fixed range of data $\{L_r\}$.

The processor structure for realizing Algorithm 3.1 operates with the bits of type $\{L_r\}$ and uses the fixation rule (3.3), and the output is obtained in the given lexicographic representation (See Fig. 2).

4 Bit-serial approach for binary lexicographic codes

It can be shown that the unary-weighted, binary-weighted, and Fibonacci p-codes are binary lexicographic codes. Input compression in [3] uses the transformation:

$$(X_1,\dots,X_n)\quad\rightarrow\quad(r_1,\dots,r_n),$$

$r_1,\dots,r_n\in\{1,2,\dots,n\}$.
By definition, input compression is the representation in lexicographic codes only for the set of numbers from the input window, because $X_i>X_j$ if and only if $r_i>r_j$. When r_i are considered as binary-weighted data, they are expressed in binary lexicographic codes. For binary monotone codes, Algorithm 3.1 processes data in an analogous way as the bit-serial approach for binary-weighted data [2]. In this case, the freezing rule is:

$$F_h(x)=\begin{cases}1, & \text{if } h<x;\\ 0, & \text{if } h>x.\end{cases}\tag{4.1}$$

For a given window, Algorithm 3.1 with the rule (4.1) for different binary monotone representations of input data is the same. Only the code representation of the input samples and the output is changed. The structure in Fig. 2 realizes the stack filter with the given PBF for arbitrary binary monotone codes, if S_f is understood as a PBF function.

Algorithm 3.1, based on arbitrary Boolean functions and the freezing rule (4.1), chooses one data from n others, when inputs are represented in an arbitrary binary code.

Theorem 4.1. We consider the input set of data: $\{X_1,\dots,X_n\}$. Suppose that each X_i is represented in an arbitrary binary code R as $X_i=(a_i^1,\dots,a_i^k)_R$, $a_i^j\in\{0,1\}$. Also suppose that f is an arbitrary Boolean function (not only PBF) of n variables, for which $f(0,0,\dots,0)=0$ and $f(1,1,\dots,1)=1$. Let $y=(h^1,h^2,\dots,h^k)$ be the output of Algorithm 3.1 for the given window and rule (4.1). Then $\exists t:\ y=X_t$, where $X_t\in\{X_1,\dots,X_n\}$.

5 Stack filters for multiple-value and monotone codes

If we represent a natural number A as $A=\sum_{i=0}^{k}a_ir^i$, where $a_i\in\{0,1,2,\dots,r-1\}$, then expression $(a_k,a_{k-1},\dots,a_0)$ is called the multiple-value code for A. Multiple-value code is a monotone representation.

Suppose that each X_i in window $\{X_1,\dots,X_n\}$ is represented in multiple-value code as $X_i=(a_i^1,\dots,a_i^k)$ $a_i^j\in\{0,1,2,\dots,r-1\}$ For multiple-value codes with the radix r we set $\{L_1,\dots,L_r\}=\{0,\dots,r-1\}$ and the freezing rule for Algorithm 3.1 will be:

$$F_h(x) = \begin{cases} r-1, & \text{if } h < x; \\ 0, & \text{if } h > x. \end{cases} \qquad (5.1)$$

The processor for realizing Algorithm 3.1 is the structure of Fig. 2, where logic switches operate in accordance with the freezing rule (5.1) and instead of a PBF, the function S_f (3.1)-(3.2) is used.

In accordance with Theorem 2.1 for multiple-value representation, stack filtering of the given input set of data with the values from $\{0,\ldots,r^p-1\}$ can be replaced by sequential stack filtering of the data set from the fixed range $\{0,\ldots,r-1\}$. If, in addition, we filter the data from this fixed range using the threshold decomposition structure, then the pipeline-parallel structure with $(r-1)$ PBF units will be obtained, which performs the stack filtering in p PBF calculation stages. The adder transforms the binary signals to multiple-value bits of the output and the result is formed also in multiple-value code.

By choosing the value of r, a pipeline-parallel class of processors can be obtained. When $r=2$, then the approach leads to the circuit proposed in [2]. For $r=2^k$, we can obtain the completely parallel case, followed from the threshold decomposition structure [5].

For the values of $r: r=2^t, 1 \le t \le k$, the representation in multiple-value code is only the grouping of bits in usual binary-weighted codes, the freezing rule is the fixation in higher level $(1,\ldots,1)$ and in lower level $(0,\ldots,0)$, and this case has the practical meaning for the possibility of a simple realization. On one hand, the quantity of PBF units leads to higher performance, on the other hand it leads to increased complexity. For concrete applications, the suitable structure can be chosen. The implementations of stack filters for arbitrary r is presented in Fig. 3 and Fig. 5.

The combination of the threshold decomposition structure with the bit-serial approach has also the following interpretation.

The output is formed in the following way. Let the current stage be $j: 1 \le j < k$, and the output bit set before the current time is $h^1 \ldots h^{j-1}$. The $r-1$ threshold levels are considered:

$$\begin{aligned} l^j_{r-1} &= h^1 \ldots h^{j-1} r-1\ 0 \ldots 0 \\ &\cdots \\ l^j_2 &= h^1 \ldots h^{j-1} 2\ 0 \ldots 0 \\ l^j_1 &= h^1 \ldots h^{j-1} 1\ 0 \ldots 0 \end{aligned} \qquad (5.2)$$

l^j_m is the lower level in levels zone

$$\underbrace{(h^1 \ldots h^{j-1} m * \ldots *)}_{\text{k bits}}, \quad m = 1,2,\ldots,r-1. \qquad (5.3)$$

The thresholds of input data $X_i = (a^1_i,\ldots,a^k_i)$ values on the levels (5.2) are coincident with the threshold values of d^j_i on the levels $1,2,\ldots,r-1$:

$$T_{l^j_m}(X_i) = T_m(d^j_i)$$

On these $r-1$ threshold levels one can calculate the values of PBF f: $(f^1,\ldots,f^{r-1})$

$$\begin{pmatrix} f^{r-1}=0 \\ \ldots\ldots \\ \ldots\ldots \\ f^{m+1}=0 \\ f^m=1 \\ \ldots\ldots \\ \ldots\ldots \\ f^1=1 \end{pmatrix}$$

The current output bit is the following sum:

$$h^j = \sum_{i=1}^{r-1} f^i.$$

In an architecture for multiple-value representation based on binary logic and PBF, the following algorithm is realized (see Fig. 3).

Algorithm 5.1

For $i=1,\ldots,n$ **do in parallel**
set $t_i = 1$
For $j=1,\ldots,k$ **do in sequential**
For $i=1,\ldots,n$ **do in parallel**
Begin
If $t_i = 1$ **then** $d_{ij} = a_{ij}$
else $d_{ij} = d_{ij}$;
A. **For** $l=1,\ldots,r-1$ **do in parallel**
if $d_{ij} \ge l$
then set $s^l_{ij} = 1$
else $s^l_{ij} = 0$
$f^l_j = PBF(s^l_{1j}, s^l_{2j}, \ldots, s^l_{nj})$
B. $h_j = f_{mv}(d_{1j}, d_{2j}, \ldots, d_{nj}) = f^{r-1}_j + \cdots + f^1_j$
If $t_i = 1$ **and** $h_j > a_{ij}$
C. **then set** $d_{ij} = 0, t_i = 0$
If $t_i = 1$ **and** $h_j < a_{ij}$
D. **then set** $d_{ij} = r-1, t_i = 0$
END

The combining of the bit-serial approach for words represented in lexicographic code with the parallel threshold decomposition algorithm (see Fig. 1) for multiple-value digits can be obtained by Algorithm 5.1, where positions A,B,C and D are replaced by the following expressions:

A. **For all $l \in \{L_r\}$ do in parallel**

B. $h_j = f_{mv}(d_{1j}, d_{2j}, \ldots, d_{nj}) = \sum_{l=1}^{r} f_j^l = L_1 f_j^{L_1} + \sum_{k=2}^{r}(L_k - L_{k-1}) f_j^{L_k}$,

C. **then set** $d_{ij} = L_1, t_i = 0$

D. **then set** $d_{ij} = L_r, t_i = 0$

Processors for arbitrary lexicographic codes on the base of PBF can be realized analogous to the processors for multiple-value codes. In this case they contain r PBF blocks, corresponding to the numbers $(L_1, \ldots, L_r)$. Output of each PBF block corresponding to the number L_k enters the $(L_k - L_{k-1})$ inputs of the postaddition circuit (Sum), which has L_r inputs (see Fig. 4).

For example, if $\{L_r\} = \{L_1, L_2, L_3\} = \{1, 3, 5\}$, then the processor has three PBF blocks, and outputs of the PBF blocks corresponding to the numbers 3 and 5 are repeated two times, and the postaddition circuit has 5 inputs.

6 Pipeline-parallel structures for stack filters

In this chapter, we propose a new class of processors, where the number of PBF blocks and the throughput may be altered (See Fig. 5 when $r = 2^t$, or Fig. 3). We use the number of PBF blocks and the number of PBF-calculation stages as parameters to characterize the properties of proposed structures.

For this purpose, we divide each k-bit set of an input sample into t-bit sets. In each stage, n t-bit words enter in parallel the stack filter based on threshold decomposition. Then a t-bit output is obtained. After comparing in logic switches, the following n t-bit words enter the stack filter, and after $\frac{k}{t}$ stages, the output for the current window is formed completely. The structure contains $2^t - 1$ PBF-blocks, where t may be changed. Every structure consists of the following parts:

1. The freezing or comparing line contains n logic switches (LS).

2. The threshold line contains $2^t - 1$ groups of switches, and each group consists of n identical switches. These logic switches (SW) are threshold switches and have one input I, state j, and output Out. If the input $I \geq j$, then $Out = 1$, if $I < j$, then $Out = 0$.

3. The PBF line contains $2^t - 1$ PBF blocks, and each of them forms a binary output on one of the threshold lines. All PBF blocks run in parallel.

4. An adder transforms the binary set of $2^t - 1$ PBF-block outputs to the binary-weighted t-bit parallel output of the stage.

Input compression can be used before the pipeline-parallel processor. Bits or digits for the chosen monotone code enter the structures successively, but connection lines transfer each bit or digit in parallel before the threshold lines and after the adders.

Because of feedback, all structures have asymptotical Area-Time complexity $O(k)$, where k is the quantity of bits of the input data, because increasing the number of bits leads to the increase of calculation steps only, but not to the increase of hardware.

7 Conclusion

The bit-serial approach for stack filter design has been generalized for the cases when input data are represented in arbitrary binary or multiple-value lexicographic codes. It is shown that many of formerly proposed structures in fact are using particular lexicographic codes.

Pipelined-parallel structures are obtained as the consequence of this approach. The relations between the existing structures are shown. Every stack filter with an arbitrary structure for the given range of data values can be supplemented by the feedback for filtering the data from a larger value range. Moreover, feedback significantly improves the asymptotic behavior of the structures.

References

[1] S. Agaian, J. Astola, K. Egiazarian, and P. Kuosmanen, "Decompositional methods for stack filtering using Fibonacci p-codes," *Signal Processing,* in press.

[2] K. Chen, "Bit-serial realizations of a class of nonlinear filters based on positive Boolean functions", *IEEE Trans. Circuits Syst.*, vol. 36, pp. 785-794, June 1989.

[3] L. Lin, G. B. Adams, and E. J. Coyle, "Input compression and efficient algorithms and architectures for stack filters," *Proc. IEEE Winter Workshop on Nonlinear Digital Signal Processing,* Tampere, Finland, Jan. 1993.

[4] L. E. Lucke and K. K. Parhi, "VLSI structures for weighted order statistics filters," *Proc. IEEE Winter Workshop on Nonlinear Digital Signal Processing,* Tampere, Finland, Jan. 1993.

[5] P. D. Wendt, E. J. Coyle, and N. C. Gallagher, "Stack filters," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. ASSP-34, pp. 898-911, Aug. 1986.

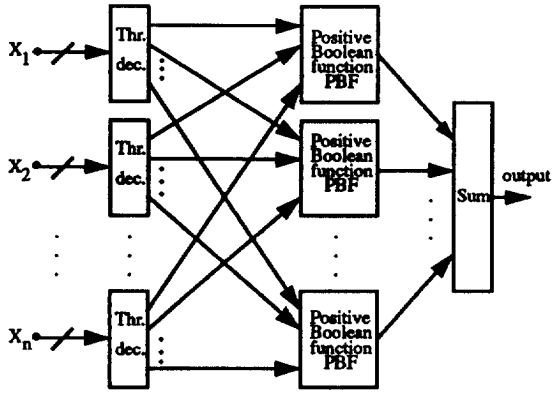

Fig. 1. Parallel threshold decomposition structure

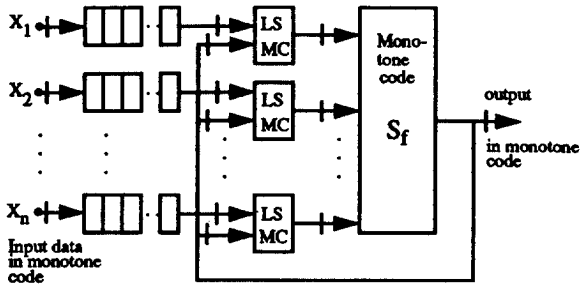

Fig. 2. Stack filter when data are represented in monotone code

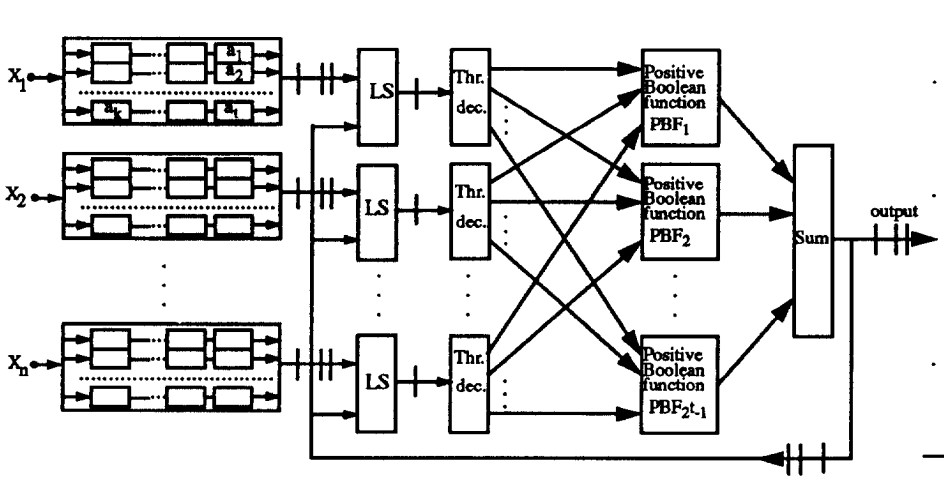

Fig. 3. Pipeline-parallel structure for stack filters

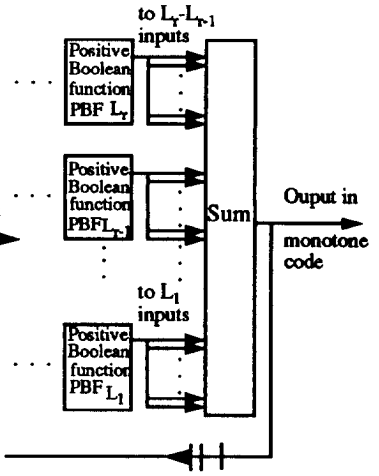

Fig .4 . Output forming of the processor for monotone code on the base of PBF.

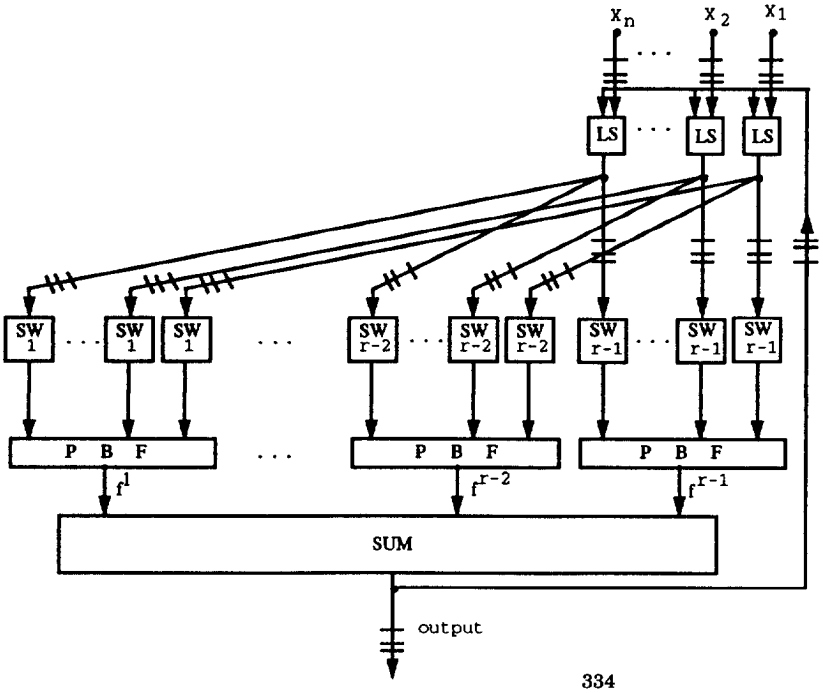

Fig.5. Stack filter based on PBF for input data, represented in multiple value code