SBNR Processor for Stack Filters

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

The signed binary number representation (SBNR) is used to reduce the number of positive Boolean function (PBF) calculation stages in stack filters. The approach is based on the possibility of minimization of signed power-of-two terms in SBNR representation of input data. A coder and a decoder for the mutual transformation of binary-weighted code and the minimal SBNR are proposed. An algorithm and a processor structure for stack filtering based on the minimal SBNR are presented. The time-area complexity of the proposed filter is $O(k)$, where k is the number of bits in the input signals.

1 Introduction

Stack filters are nonlinear digital filters, used for suppression of noise, when linear methods are not effective for signal smoothing and noise reduction. This class of filters has been introduced by Wendt et al. [8] and includes the well-known median, order statistic, and weighted order statistic filters, among others. Stack filters use a sliding window to observe successively fragments of the given signals or images, and produce the output for every window position.

A well known realization based on a bit-serial approach performs processing on binary-weighted signals and has a simple structure, but the output is obtained after $log_2 L$ stages of calculation of PBF, where $\{0, \ldots, L-1\}$ are the possible values of input data [3].

In order to reduce PBF calculation stages, input compression before the stack filter has been proposed [9]. Sample values 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 samples. Another way to increase the throughput is the representation of input data in Fibonacci p-codes [1].

Signed binary number representation (SBNR) is an efficient tool in digital filter implementations [2],[5],[6],[7], which is applied particularly for simplifying interconnections in VLSI design and for reducing multiplication operations.

In this paper we show that stack filtering with a bit-serial algorithm is possible for SBNR, and derive the processor structure for this task. This possibility is used to reduce the average number of PBF calculation stages in a stack filter. The approach is based on the property of the minimal SBNR that every signed unit bit must be followed by a zero and so, after every unit bit on the output, one stage can be omitted. The coder and decoder for mutual transformation of the binary-weighted code and the minimal SBNR are constructed in a systolic way.

2 Definitions and properties

2.1 Stack filters

Suppose that $X(t)$ is a discrete-time stochastic process, $X(t) \in Q = \{0, 1, \ldots, M-1\}$.

We denote the current window of input data as:

$$\vec{X} = (X_1, X_2, \ldots, X_n). \tag{2.1}$$

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

$$x_{i,m} = T_m(X_i) = \begin{cases} 1, & \text{if } X_i \geq m; \\ 0, & \text{if } X_i < m \end{cases} \tag{2.2}$$

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

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):

$$S_f = K_1 \vee K_2 \vee \cdots \vee K_s \tag{2.3}$$

$$K_i = x_{i_1} \wedge x_{i_2} \wedge \ldots \wedge x_{i_{r(i)}}. \tag{2.4}$$

We also consider the functions (2.3)-(2.4), where $x_i \in \{L_1, \ldots, L_r\}$, $L_1 < \ldots < L_r$, $\vee$ is understood as maximum operator, $\wedge$ denotes the minimum operator. In this case when x_i represents the values of input data, S_f determines the stack filter, which can be also defined in traditional way:

Definition 2.2. [8]. A window width n stack filter $S_f(\cdot) : Q^n \rightarrow Q$ is based on an n-variable positive Boolean function $f(\cdot)$: $\{0,1\}^n \rightarrow \{0,1\}$, operating on the binary threshold signals. The output of the stack filter is obtained by adding all the binary outputs:

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

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

2.2 Signed digit number representations, GNAF and minimal SBNR

The modified r-ary form of the number N is

$$N = e_n r^n + \cdots + e_1 r + e_0 \qquad (2.5)$$

where $e_i \in Z, |e_i| < r$ for all $i = 0, 1, \ldots, n$. This representation is not unique. However, the following property is known [4]:

Property 2.1. If the coefficients e_i $(i = 0, \ldots, n-1)$ in (2.5) satisfy the following conditions:

$$|e_i + e_{i+1}| < r \qquad (2.6)$$

$$|e_i| < |e_{i+1}| \quad if \quad e_i e_{i+1} < 0, \qquad (2.7)$$

then representation (2.5) is unique and minimal, i.e., the number of nonzero coefficients is minimal. Representation (2.5) when (2.6),(2.7) are satisfied is called the generalized non-adjacent form (GNAF). In this paper we are interested in the case $r = 2$, when Property 2.1 is transformed to:

Property 2.1.a. For every natural number L there exists one and only one representation:

$$L = \sum_{i=0}^{k} a_i 2^i, \quad a_i \in \{-1, 0, 1\} \qquad (2.8)$$

where $a_j a_{j+1} = 0$ for every $j = 0, 1, \ldots, k-1$.

We call the representation, defined in Property 2.1.a, as the minimal signed binary number representation (SBNR).

2.3 Coder from binary-weighted to minimal SBNR representation

Let us consider the binary-weighted code of the number A:

$$(a_0, a_1, \ldots, a_{k-2}, a_{k-1}) : \; A = \sum_{t=0}^{k-1} a_t 2^t \qquad (2.9)$$

We consider the $(k+1)$ bit representation with the added "0" bit from the right.

Property 2.2. Every bit sequence $\{1, 1, \ldots, 1, 0\}$ in the binary-weighted code representation of an arbitrary number A can be replaced by the sequence $\{-1, 0, \ldots, 0, 1\}$ of the same length.

Moving from the less significant bit in 0 position to the most significant bit in $(k-1)$ position, in our case from the left to the right, we can obtain the minimal SBNR.

The following algorithm can be used for the realization of this procedure:

Given $(a_0, \ldots, a_{k-1})$
1. $x_1 = 0$
2. **for** $t = 0, \ldots, k-1$ **do in sequential**
3. $x_2 = a_t, x_3 = a_{t+1}$
4. $y_1 = x_2 x_3 + x_1 x_3 + x_1 x_2$
5. $y_2 = x_1 \bar{x}_2 + \bar{x}_1 x_2$
6. $y_s = x_3(x_1 \bar{x}_2 + \bar{x}_1 x_2)$
7. $b_t = y_2, sign(b_t) = y_s$
8. $x_1 = y_1$
9. $b_k = x_1$
10. End

In steps 4-6, multiplications and additions mean the logical AND and OR operations, respectively. This algorithm can be realized systolically, as shown in Fig. 1.

2.4 Decoder from minimal SBNR to binary-weighted code

For the inverse transformation from arithmetic code $(b_0, \ldots, b_k)$ to binary-weighted code $(a_0, \ldots, a_k)$, we can move from the most significant bit b_k to the less significant b_0, replacing in sequential each fragment $(-1, 0, \ldots, 0, 1)$ by the set $(1, 1, \ldots, 1, 0)$. This procedure can be realized also systolically (see Fig. 2):

Given $(b_0, \ldots, b_k)$
1. $y_1 = 0$
2. **for** $t = 0, \ldots, k$ **do in sequential**
3. $y_2 = b_t, y_s = sign(b_t)$
4. $x_1 = y_2 y_s + y_1 \bar{y}_2 \bar{y}_s$
5. $x_2 = \bar{y}_1 y_2 + y_1 \bar{y}_2 \bar{y}_s$
6. $y_1 = x_1$
7. $a_t = x_2$
8. End

2.5 Lexicographic (monotone) codes

Let us consider the representation of the data values $0, 1, \ldots, M-1$ in the following form

$$\underline{c}(i) = (c_1(i), \ldots, c_p(i)), \quad i = 0, 1, \ldots, M-1$$

where $c_j(i) \in \{L_1, \ldots, L_r\}$, $j = 1, \ldots, p$

and $L_1 < L_2 < \cdots < L_r$. We usually write $(a_i^1, \ldots, a_i^p)$ instead of $(c_1(i), \ldots, c_p(i))$ for the coding function. Note also that the *digits* of the code need not be integers. We only require that they have a linear order $L_1 < L_2 < \cdots < L_r$.

Definition 2.4. 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 $a_i^t > a_j^t$.

Theorem 2.1. GNAF and minimal SBNR are lexicographic representations of the numbers.

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 the expressions (2.3)-(2.4) where $\vee$ denotes the maximum and $\wedge$ denotes the minimum operations. We consider the following freezing rule $F_h(x)$:

$$F_h(x) = \begin{cases} L_r, & \text{if } h < x; \\ L_1, & \text{if } h > x. \end{cases} \tag{2.10}$$

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 monotone code representation of the samples.

We consider the following mapping, depending on a function S_f :

$$d_i^1 = a_i^1, \ i = 1, \ldots, n,$$

$$h^1 = S_f(d_1^1, \ldots, d_n^1), \tag{2.11}$$

and d_i^j are defined recursively:

$$d_i^j = \begin{cases} a_i^j & \text{if } a_i^m = h^m \text{ for } m = 1, 2, \ldots, j-1; \\ L_1 & \text{if } \exists l < j : a_i^m = h^m \text{ for } m = 1, 2, \ldots, l-1 \\ & \text{and } a_i^l < h^l; \\ L_r & \text{if } \exists l < j : a_i^m = h^m \text{ for } m = 1, 2, \ldots, l-1 \\ & \text{and } a_i^l > h^l; \end{cases} \tag{2.12}$$

$$h^m = S_f(d_1^m, \ldots, d_n^m), \tag{2.13}$$

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 (2.10). The following recursive sequence is formed:

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

Lemma 2.1. If input numbers X_i are represented in lexicographic code as $(a_i^1 \ldots a_i^k)$, then

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

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

$$S_f \left[\begin{pmatrix} d_1^1 \\ \vdots \\ d_1^k \end{pmatrix} \quad \cdots \quad \begin{pmatrix} d_n^1 \\ \vdots \\ d_n^k \end{pmatrix} \right] = \begin{bmatrix} S_f(d_1^1 \ldots d_n^1) \\ \vdots \\ S_f(d_1^k \ldots d_1^k) \end{bmatrix} \tag{2.14}$$

where d_i^j is defined in (2.11)-(2.13).

3 SBNR processor for stack filters

The following algorithm based on Lemma 2.1 for monotone codes transforms stack filtering of samples from the range $\{0, \ldots, M-1\}$ to sequential stack filtering of data from the fixed set $\{-1, 0, 1\}$. The freezing rule is:

$$F_h(j) = \begin{cases} -1, & \text{if } h > j; \\ 1, & \text{if } h < j; \end{cases} \tag{3.1}$$

Algorithm 3.1.

Input window $\{X_1, \ldots, X_n\}, i = 1, \ldots, n$, $X_i = (a_i^1, \ldots, a_i^k)$, $a_i^j \in \{-1, 0, 1\}$, $\{L_1, \ldots, L_r\} = \{-1, 0, 1\}$, and so

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** ($\leftarrow$)
Begin
If $t_i = 1$ **then** $d_{ij} = a_{ij}$
else $d_{ij} = d_{ij}$;
$h_j = S_f(d_{1j}, d_{2j}, \ldots, d_{nj})$
If $t_i = 1$ **and** $h_j > a_{ij}$
then set $d_{ij} = -1, t_i = 0$
If $t_i = 1$ **and** $h_j < a_{ij}$
then set $d_{ij} = 1, t_i = 0$
If $|h_j| = 1$ *after each 1 and -1 bit 0 bit follows,*
then set $h_{j+1} = 0, j = j+2$ **and GO TO** ($\leftarrow$)
END

Here S_f is the function of type (2.3),(2.4). After each signed 1, the algorithm automatically sets the

following output as 0, allowing the PBF calculation stage to be bypassed. The architecture for stack filtering using arithmetic code representation of input data but binary logic and PBF f is shown in Fig. 3. The following algorithm is realized in this processor:

Algorithm 3.2. (for SBNR stack filter)
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** ($\leftarrow$)
Begin
If $t_i = 1$ **then** $d_{ij} = a_{ij}$
else $d_{ij} = d_{ij}$;
If $d_{ij} = -1$, **then set** $r_{ij} = 0, s_{ij} = 0$
If $d_{ij} = 0$, **then set** $r_{ij} = 0, s_{ij} = 1$
If $d_{ij} = 1$, **then set** $r_{ij} = 1, s_{ij} = 1$
$f_j^+ = PBF(r_{1j}, r_{2j}, \ldots, r_{nj})$
$f_j^- = PBF(s_{1j}, s_{2j}, \ldots, s_{nj})$
$h_j = f_j^- + f_j^+ - 1$
If $t_i = 1$ **and** $h_j > a_{ij}$
then set $d_{ij} = -1, t_i = 0$
If $t_i = 1$ **and** $h_j < a_{ij}$
then set $d_{ij} = 1, t_i = 0$
If $|h_j| = 1$, *after each -1 and 1 bits 0 bit follows*
then set $h_{j+1} = 0, j = j + 2$ **and GO TO** ($\leftarrow$)
END

The logic switches (LS) in the processor serve for the realizing of freezing rule (3.1), while the logic switches SW are threshold switches.

Because of feedback, the structure has asymptotical Area-Time complexity $O(k)$, where k is the quantity of bits of the input data, because increasing of bits leads to the increasing of calculation steps only, but not to the increasing of hardware. Processor contains two PBF units. The quantity of PBF-calculation stages p can be a value from $\frac{k}{2} \leq p \leq (k+1)$ and is equal to the number of 0 bits in the signed representation of output for every window position. On the average, it can be evaluated by the value $\hat{p} = \frac{3}{4}k + \frac{1}{2}$.

For comparison, the threshold decomposition structure [8] contains $2^k - 1$ PBF units and processes data for one PBF calculation stage, and the bit-serial structure in [3] has one PBF unit, processing the input set of data for k stages. If $k = 16$, then the bit-serial processor in [3] forms the output after 16 PBF stages, while the proposed processor forms the output after only 12 stages.

4 Conclusion

In this paper, the bit-serial approach for stack filter design is generalized for the case when input data are in minimal signed binary representation. This leads to (on the average) reduction in the number of PBF-calculation stages, taking into account the possibility to ignore the following step of processing when a unit bit occurs as the current output. The processing of signed bits is performed using two parallel channels. A simple coder and decoder for mutual transformation of binary-weighted code and SBNR work in parallel with the stack filter and do not increase the processing time. The proposed approach also makes it possible to use stack filters in systems based on signed arithmetic.

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] M. Andrews, "Systolic SBNR adaptive signal processor," *IEEE J. Solid-State Circuits*, vol. SC-21, pp. 120-128, 1986.

[3] 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.

[4] W. E. Clark and J. J. Liang, "On arithmetic weight for a general radix representation of integers," *IEEE Trans. Inform. Theory* IT - 19 (1973), pp. 823-826.

[5] J. B. Evans and B. Liu, "A CMOS implementation of a variable step size digital adaptive filter," *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, Glasgow, Scotland, May 1989, pp. 2489-2492.

[6] S. L. Hurst, "Multiple-valued logic, its status and its future," *IEEE Trans. Comput.*, vol. C-33, pp. 1160-1179, Dec. 1984.

[7] Y. C. Lim and B. Liu, "Design of cascade form FIR filters with discrete valued coefficients", *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, pp. 1735-1739, Nov. 1988.

[8] 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.

[9] 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.

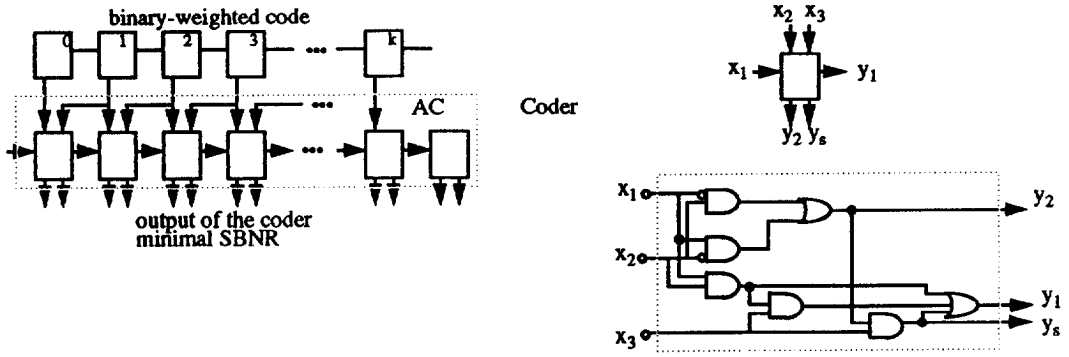

Fig. 1. Binary-weighted to minimal SBNR converter (arithmetic coder)

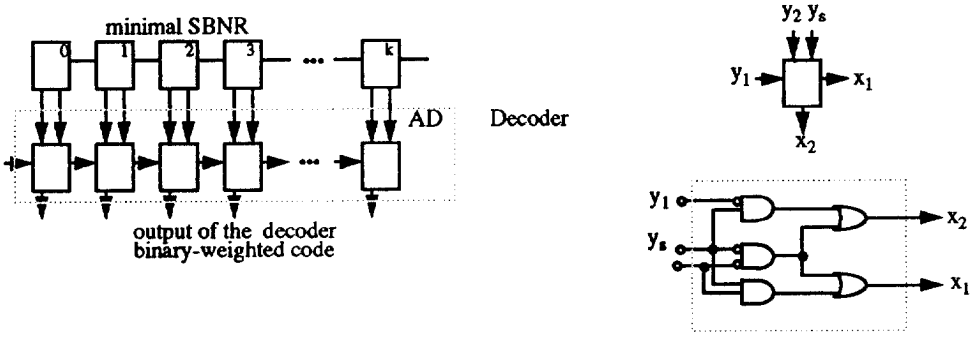

Fig. 2. Decoder from minimal SBNR to binary-weighted code

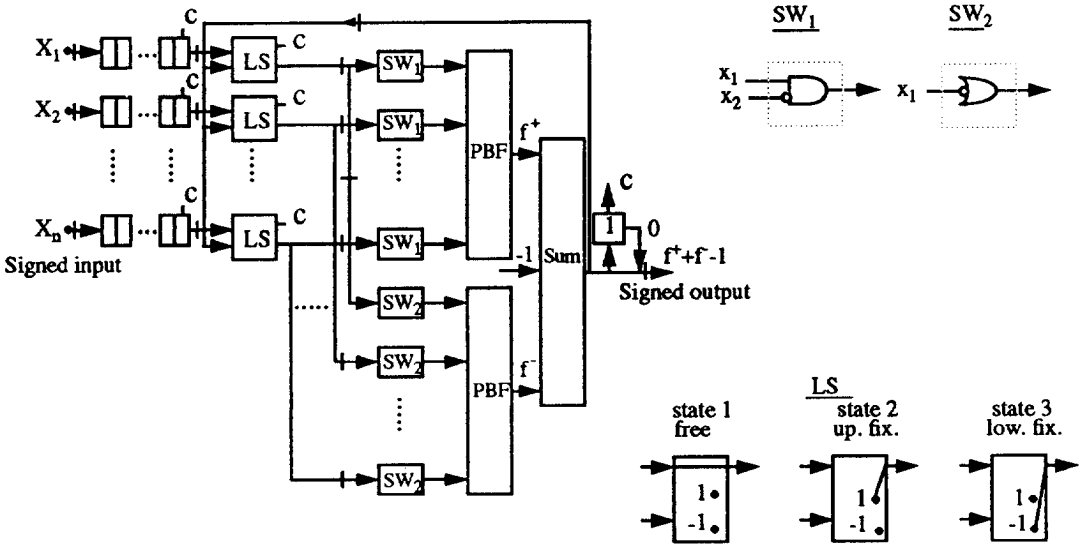

Fig. 3. SBNR processor for stack filter on the base of PBF with binary variables