Pascal-Interpolation-Based Noninteger Delay Filter and Low-Complexity Realization

Parinya SOONTORNWONG, Sorawat CHIVAPREECHA

Dept. of Telecommunication Engineering, Faculty of Engineering,
King Mongkut's Institute of Technology Ladkrabang, Chalongkrung Rd., Ladkrabang, Bangkok 10520, Thailand

s4610104@kmitl.ac.th, sorawat@telecom.kmitl.ac.th

Abstract. *This paper proposes a new method for designing the polynomial-interpolation-type noninteger-delay filter with a new structure formulation. Since the design formulation and the new realization structure are based on the discrete Pascal transform (DPT) and Pascal interpolation, we call the resulting filter Pascal noninteger-delay filter. The kth-order Pascal polynomial is used to pass through the given (k + 1) data points in achieving the kth-order Pascal filter. The Pascal noninteger-delay filter is a real-time filter that consists of two sections, which can be realized into the front-section and the back-section. The front-section contains multiplication-free digital filters, and the number of multiplications in the back-section just linearly increases as order becomes high. Since the new Pascal filter has low complexity and structure can adjust non-integer delay online, it is more suited for fast delay tuning. Consequently, the polynomial-interpolation-type delay filter can be achieved by using the Pascal approach with high efficiency and low-complexity structure.*

Keywords

Low-complexity filter structure, noninteger-delay filter, discrete Pascal transform (DPT), polynomial-interpolation-type delay filter, Pascal noninteger-delay filter

1. Introduction

Since the tunable noninteger-delay filter can change its non-integer group delay or phase delay, it can be applied to many fields of signal processing like one dimensional signal interpolation, sampling-rate conversion and image interpolation. So far, many design methods have been developed in the literature [1–19]. Among those existing filters, the polynomial-interpolation-type noninteger-delay filter has simple and unique features. Its amplitude response is the maximally flat for zero frequency. The derivation and realization of the polynomial-interpolation-type noninteger-delay filter have been dealt with in [9–13]. The most straightforward realization of the polynomial-interpolation-type noninteger-delay filter is to use the direct-form Farrow structure [1]. In practical applications, because low frequency components dominate, employing the polynomial-interpolation-type noninteger-delay filter can achieve sufficiently satisfactory interpolations as described in [15].

The direct-form Farrow structure for realizing the polynomial-interpolation-type noninteger-delay filter requires a large number of multiplications, but non-causal ones can be symmetrically realized by exploiting symmetry transformations. After symmetry transformations, the structures need much fewer multiplications. In order to perform real-time filtering, causal filters are needed, but the direct-form Farrow structure of the causal polynomial-interpolation-type noninteger-delay filters does not possess symmetric or anti-symmetric structures, which cannot exploit symmetries for reducing the number of multiplication operations. In [11–13], transformation matrices have been proposed for transforming causal filters to have low-complexity structures.

This paper develops a new design method and a new realization structure for the causal polynomial-interpolation-type delay filter with very low hardware complexity in order to achieve high-speed filtering for real-time applications. As compared with the methods proposed in [10–13], the method proposed in this paper requires much less multiplications. The resulting filter structures from the methods in [10–13] require more multiplications than the Pascal structure proposed in this paper. As compared with the existing methods for deriving and realizing the causal polynomial-interpolation-type delay filters, we develop a novel design method and a new filter structure using different procedures, but the resulting structure has much lower complexity.

The proposed design method is related to a kind of discrete polynomial transform called discrete Pascal transform (DPT) that was first proposed in [20], and later the DPT is applied to the so-called Pascal interpolation in [21]. The new design method and the new filter structure for the polynomial-interpolation-type delay filter is based on the Pascal interpolation, and the Lagrange polynomial is changed to Pascal polynomial for fitting the data points. Therefore, this new type of filter is called Pascal noninteger-delay filter. The Pascal noninteger-delay filter can

adjust the delay parameter online with high processing speed because it uses a reduced number of multiplications, where the number linearly increases as the filter order increases. Although the coefficient-symmetry techniques in [10–13] can reduce the multiplication operations by about 50%, the techniques still require more multiplications as compared with the Pascal noninteger-delay filter because the numbers of multiplications in [10–13] are not linear functions of the filter order. Consequently, the low-cost and high-speed causal polynomial-interpolation-type filter can be obtained by using the Pascal approach and the filter order can take both odd and even numbers. The Pascal noninteger-delay filter makes online tuning with very low hardware realization and high-speed operation possible, which is the major progress for the polynomial-interpolation-type delay filter.

2. Direct-Form Structure and Transformed Structure

This section briefly reviews the polynomial-interpolation-type noninteger-delay filter realized using the direct-form Farrow structure [1] and the Farrow structures using symmetry transformations [11–13]. The polynomial-type noninteger-delay filter is also termed Lagrange-type noninteger-delay filter as it stems from the interpolation using the Lagrange interpolating polynomial [9], [11–13].

The output of the polynomial-interpolation-type noninteger-delay digital filter is

$$y(n) = \sum_{i=0}^{k} h_i(D)x(n-i) \tag{1}$$

where

$$h_i(D) = \prod_{\substack{l=0 \\ l\neq i}}^{k} \frac{D-l}{i-l} \tag{2}$$

are the coefficients, and D is the fractional delay. Also, k is the filter order. The time-domain expression in (1) corresponds to the frequency-domain transfer function

$$H(z,D) = \sum_{i=0}^{k} h_i(D)z^{-i}. \tag{3}$$

By employing the Farrow structure, $H(z, D)$ can be realized as Fig. 1, where $V_i(z)$, $i = 0, 1, \ldots, k$, are called subfilters. For example, if the filter order $k = 2$, then

$$H(z,D) = V_0(z) + DV_1(z) + D^2V_2(z)$$

with

$$\begin{aligned} V_0(z) &= 1, \\ V_1(z) &= -\frac{3}{2} + 2z^{-1} - \frac{1}{2}z^{-2}, \\ V_2(z) &= \frac{1}{2} - z^{-1} + \frac{1}{2}z^{-2}. \end{aligned} \tag{4}$$

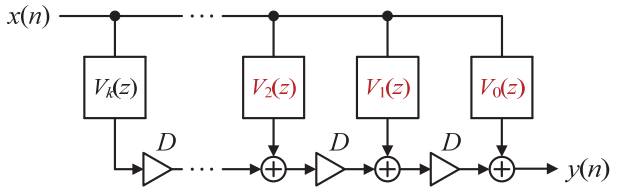

Fig. 1. Direct-form Farrow structure.

Here, the subfilters have constant coefficients, which are fixed in filtering. The non-integer delay of $H(z, D)$ can be adjusted through adjusting the value of the parameter D. The direct realization of $H(z, D)$ using the Farrow structure requires

No. of multiplications = $k^2 + 2k$

No. of additions = $k^2 + k$.

On the other hand, we can see from (4) that the last subfilter $V_2(z)$ has symmetric coefficients, but the others do not have any symmetries. The coefficient symmetry of $V_2(z)$ is taken into account in the realization shown in Fig. 2.

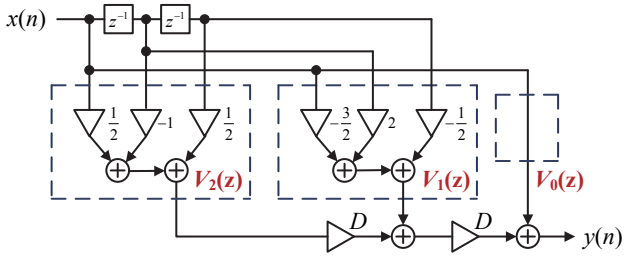

Fig. 2. Example of the direct-form Farrow structure.

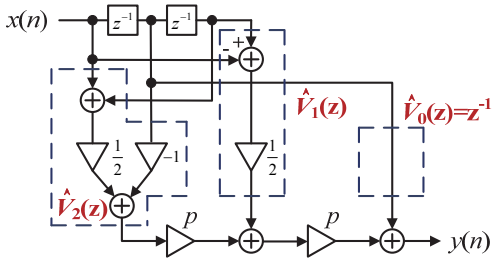

Fig. 3. Example of the transformed Farrow structure.

In order to reduce the number of multiplications needed in the filtering process, it is desirable to symmetrize the coefficients of all the subfilters $V_0(z)$, $V_1(z)$, …, $V_k(z)$. In [10], symmetric structures are proposed for realizing non-causal polynomial-interpolation-type noninteger-delay filters, and in [11–13], transformation approaches are proposed for realizing causal ones.

Figure 3 shows the second-order transformed Farrow structure, which can reduce the number of multiplications by more than 50%. As a result, the Farrow structures after transformations have the following complexities.

Odd-order case: For any odd kth-order case, the computational complexity can be figured out by the same way and can be summarized as follows:

$$\text{No. of multiplications} = \left(\frac{k+1}{2}\right)(k+1)+k = \frac{k^2+4k+1}{2}$$

$$\text{No. of additions} = (k+1)\frac{(k+1)(k-1)}{2}+k = \frac{k^2+4k+1}{2}.$$

Even-order case : For any even kth-order case, the computational complexity can be summarized as

$$\text{No. of multiplications} = \left(\frac{k}{2}+1\right)\frac{k}{2}+\left(\frac{k}{2}\right)^2+k = \frac{k^2+3k}{2}$$

$$\text{No. of additions} = \left(\frac{k^2}{4}+\frac{k}{2}\right)+\left(\frac{k^2-2k}{4}+\frac{k}{2}\right)+k = \frac{k^2+3k}{2}.$$

It is clear that the numbers of multiplications are still the square functions of the filter order k.

In the next section, we will propose a novel design method and a new filter structure called Pascal noninteger-delay filter for the polynomial-interpolation-type filter, which requires much fewer multiplications as compared to other ones. We will begin by discussing the background of the Pascal noninteger-delay filter design, including the generalized Pascal noninteger-delay filter and its filter structure realization, the frequency characteristic and computational complexity from the next section.

3. Discrete Pascal Transform and Pascal Interpolation

Discrete Pascal transform (DPT) was proposed in [22], which is intimately related to the Pascal's triangle shown in Fig. 4,

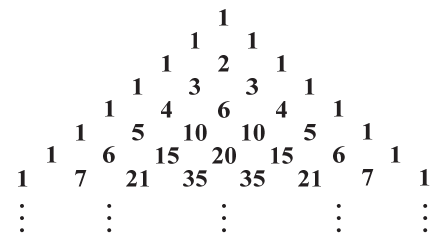

Fig. 4. Pascal's triangle.

and is classified as one kind of discrete polynomial transforms. The operation of DPT is based on Pascal transform matrix that performs as an operator for signal transformation.

The discrete Pascal transform of the signal vector $\mathbf{x}$ with size $(k + 1) \times 1$ is defined as

$$\mathbf{X} = \mathbf{P}\mathbf{x} \tag{5}$$

where $\mathbf{P}$ is the Pascal transform matrix with dimension $(k + 1) \times (k + 1)$, and $\mathbf{x}$ is the input signal vector, and $\mathbf{X}$ is the transformed output vector with size $(k + 1) \times 1$.

3.1 Basis Functions of DPT

From (5), the Pascal transform matrix has the following basis functions, which are the polynomials

$$P_i(x) = P(x,i) = \frac{(-1)^i x^{(i)}}{i!} = (-1)^i \binom{x}{i},\ x,i = 0,1,...,k \tag{6}$$

where $(k + 1) \times (k + 1)$ is the size of the Pascal transform matrix and

$$\binom{x}{i} = \frac{x!}{i!(x-i)!} \tag{7}$$

are called the binomial coefficients (as in the binomial theorem) and are related to Pascal's triangle. The term $(-1)^i$ in (6) will alternate the sign of the columns of the Pascal transform matrix, and $P(x, i)$ are the elements in Pascal transform matrix $\mathbf{P}$, where x corresponds to the row number and i denotes the column number.

The function $x^{(i)}$ is called falling factorial powers defined as

$$\begin{aligned} x^{(i)} &= x(x-1)(x-2)...(x-i+2)(x-i+1) \\ &= \frac{x!}{(x-i)!}, \qquad i = 0,1,...,k \end{aligned} \tag{8}$$

where $x^{(0)} = 1$, and k is also called the order of basis function. For example, the first four basis functions of DPT that consist of four polynomials are

$$\begin{aligned} &P_0(x) = 1, \\ &P_1(x) = -x, \\ &P_2(x) = \frac{1}{2}x(x-1) = -\frac{1}{2}(x-1)P_1(x), \\ &P_3(x) = -\frac{1}{6}x(x-1)(x-2) = -\frac{1}{3}(x-2)P_2(x) \end{aligned}$$

which can be summarized as the recurrence formula

$$P_{i+1}(x) = -\frac{1}{i+1}(x-i)P_i(x). \tag{9}$$

By using (9), we can generate the Pascal transform matrix (size 4×4) for $k = 3$ as follows,

$$\mathbf{P} = \left[P(x,i)\right] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -3 & 3 & -1 \end{bmatrix}.$$

The basic properties of this $\mathbf{P}$ matrix are as follows.

(1) All the elements of the first column equal 1.

(2) The matrix is a lower triangle.

(3) The sum of all the elements of each row (except the first row) equals zero.

(4) The inverse matrix is equal to the forward matrix, that is, $\mathbf{P}^{-1} = \mathbf{P}$.

3.2 Pascal Interpolation

Assume that $x_L(n)$ is obtained by interpolating the original signal $x(n)$ with an interpolation factor of L, then $L-1$ new data points will be inserted between every pair of samples in the original $x(n)$. The new interpolation technique using the DPT in [20] is proposed in [21]. The idea used in Pascal interpolation is similar to the concept of Fourier series expansion, that is, a periodic signal can be expressed by the weighted-sum of exponential basis functions. In the case of discrete Pascal transform (DPT), the basis functions are Pascal polynomials $P_i(x)$ shown in (6). Then, the discrete-time signal $x(n)$ can be represented by using the DPT bases as

$$x(n) = \sum_{i=0}^{k} X_i P_i(n), \qquad 0 \le n \le k \tag{10}$$

where the weights are the Pascal coefficients X_i that are calculated as

$$X_i = \sum_{n=0}^{i} P_n(i) x(n), \qquad 0 \le i \le k. \tag{11}$$

Here, the number of data points of $x(n)$ is $(k+1)$, and the order of the Pascal polynomial $P_i(n)$ is i. The Pascal coefficients X_i are similar to the Fourier coefficients in Fourier series.

To interpolate the input $x(n)$ with an interpolation factor L for $L = k+1$, which produces $(k+1)+k(L-1)$ output samples $x_L(n)$, we scale the index n by $1/L$, i.e., we change the basis functions from $P_i(n)$ to $P_i(n/L)$. Consequently, the resulting interpolated samples $x_L(n)$ are

$$y(n) = x\left(n/L\right) = \sum_{i=0}^{k} X_i P_i\left(n/L\right), \tag{12}$$

for $0 \le n \le \left[(k+1)+k(L-1)\right]-1$. This equation is the important key of the Pascal interpolation. For simplicity, the process from $x(n)$ to $x(n/L)$ can be summarized as Fig. 5.

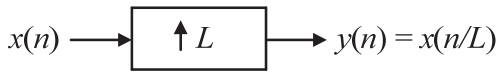

Fig. 5. Concept of Pascal interpolation.

The mechanism inside block diagram in Fig. 5 corresponds to (11) and (12), respectively. For easy understanding, we explain the Pascal interpolation by using the following example.

Example, given a sequence $x(n) = [1 \quad 2 \quad 3]$ and we want to interpolate this sequence with $L = 3$.

Step 1: Before interpolation, the Pascal coefficients must be computed by transformation from $x(n)$ to X_i using the DPT in (11).

$$X_i = \sum_{n=0}^{i} (-1)^n \binom{i}{n} x(n), \quad 0 \le i \le 2.$$

The terms $(-1)^n \binom{i}{n}$ represent the elements of Pascal transform matrix used for DPT, then

$$\mathbf{P} = \left[P(i,n)\right]$$

and

$$P(i,n) = P_n(i) = (-1)^n \binom{i}{n}, \qquad i,n = 0,1,...,k$$

with $k = 2$. The Pascal coefficient vector can be expressed by matrix equation in (5), i.e.,

$$\mathbf{X} = \mathbf{P}\mathbf{x}$$

where $\mathbf{X}=[X_0 \ X_1 \ \dots \ X_k]^{\mathrm{T}}$ and $\mathrm{x}=[x(0) \ x(1) \ \dots \ x(k)]^{\mathrm{T}}$.

Therefore, in this step the Pascal coefficients can be obtained by

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}. \tag{13}$$

Step 2: After we obtain the Pascal coefficients from Step 1, the interpolated signal can be obtained using (12). In this case, new $(L-1)=2$ data points will be inserted between each pair of the input samples $x(n)$.

$$y(n) = x\left(n/3\right) = \sum_{i=0}^{2} X_i P_i\left(n/3\right), \qquad 0 \le n \le 6.$$

Then,

$$y(n) = X_0 P_0\left(n/3\right) + X_1 P_1\left(n/3\right) + X_2 P_2\left(n/3\right), \qquad 0 \le n \le 6.$$

From basis function (6), we get

$$P_0\left(n/3\right) = 1, \qquad P_1\left(n/3\right) = -\frac{n}{3}, \qquad P_2\left(n/3\right) = \frac{1}{2}\frac{n}{3}\left(\frac{n}{3}-1\right).$$

Therefore,

$$y(n) = x\left(n/3\right) = X_0 + \left(-\frac{n}{3}\right)X_1 + \left(\frac{1}{2}\frac{n}{3}\left(\frac{n}{1}-1\right)\right)X_2, \tag{14}$$

for $0 \le n \le 6$.

Substituting X_0, X_1, and X_2 from Step 1 into (14) yields

$$\begin{aligned}
y(0) &= x(0) = 1, \\
y(1) &= x\left(1/3\right) = 4/3 = 1 + \frac{1}{3}, \\
y(2) &= x\left(2/3\right) = 5/3 = 1 + \frac{2}{3}, \\
y(3) &= x(1) = 2, \\
y(4) &= x\left(4/3\right) = 7/3 = 2 + \frac{1}{3}, \\
y(5) &= x\left(5/3\right) = 8/3 = 2 + \frac{2}{3}, \\
y(6) &= x(2) = 3.
\end{aligned}$$

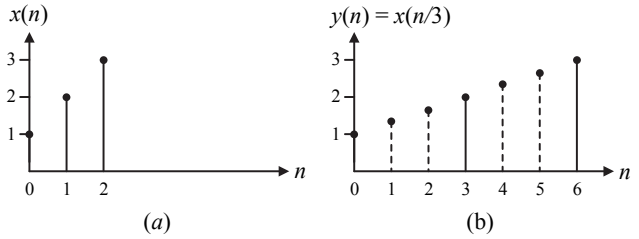

Fig. 6. Example of the Pascal interpolation with a factor $L = 3$.

The original discrete-time input signal $x(n)$ and the interpolated output signal $x(n/3)$ are shown in Fig. 6.

From the above example, we can see that the kth-order (degree) Pascal polynomial can fit the $(k + 1)$ samples, the second-order Pascal polynomial fits 3 input data points. This property is also identical to the Lagrange interpolation, which has been used to derive the polynomial-interpolation-type noninteger-delay filter. Therefore, the Pascal interpolation can also be used for developing a new structure for the polynomial-interpolation-type filter.

4. Pascal Noninteger-Delay Filter

In this section, we will briefly discuss why the Pascal interpolation can be used for developing a new structure for the polynomial-interpolation-type filter, which is called Pascal noninteger-delay filter.

4.1 Origin of Pascal Noninteger-Delay Filter

Below, we develop the Pascal noninteger-delay filter based on Pascal interpolation. The Pascal polynomial is used for fitting the data points instead of the Lagrange polynomial. The basic idea is directly related to the DPT and Pascal interpolation, which is described in the previous section. The example used in the previous section is continuously used to develop the Pascal noninteger-delay filter.

From the previous example in Step 1 (the stage of the Pascal coefficients calculation from input data vector using (13)), the matrix equation can be rewritten as

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \end{bmatrix}. \tag{15}$$

In the real-time applications, the system must be causal. Then, we can set

$x(0) = x(n)$; present input

$x(1) = x(n-1)$; 1 sample delayed input

$x(2) = x(n-2)$; 2 samples delayed input.

Therefore,

$X_0 = x(n)$,

$X_1 = x(n) - x(n-1)$,

$X_2 = x(n) - 2x(n-1) + x(n-2)$.

From this point, we know that X_0, X_1, and X_2 can be regarded as the outputs of the filters $(1 - z^{-1})^i$, $i = 0, 1, 2$. For any ith-order $(0 \le i \le k)$ the Pascal coefficients in z-transform $Z\{X_i\}$ are obtained by

$$Z\{X_i\} = H_i(z)X(z) = (1 - z^{-1})^i X(z) \tag{16}$$

with $H_i(z) = (1 - z^{-1})^i$.

The next stage is to change n/L in (14) to the delay parameter (D), which leads to

$$x(n-D) = y(n) = X_0 + (-D)X_1 + \left(\frac{1}{2}D(D-1)\right)X_2 \tag{17}$$

where k is also the Pascal noninteger-delay filter order.

To confirm that our Pascal noninteger-delay filter is correct and can operate, the second-order Pascal noninteger-delay filter $(k = 2)$ in (17) will be tested in 2 cases: integer-delay and noninteger-delay.

Integer-delay:

$D = 0$; $y(n) = X_0 = x(n)$; no delay

$D = 1$; $y(n) = X_0 - X_1$
$= x(n-1)$; 1 sample delayed

$D = 2$; $y(n) = X_0 - 2X_1 + X_2$
$= x(n-2)$; 2 samples delayed

Noninteger-delay:

$$D = 0.5 = \frac{1}{2};\ y(n) = x(n-0.5) = X_0 - \frac{1}{2}X_1 - \frac{1}{8}X_2$$
$$= \left[1 - \frac{1}{2} - \frac{1}{8}\right]x(n) + \left[\frac{1}{2} + \frac{1}{4}\right]x(n-1) - \frac{1}{8}x(n-2).$$

Then,

$$y(n) = \frac{3}{8}x(n) + \frac{3}{4}x(n-1) - \frac{1}{8}x(n-2) = x(n-0.5).$$

Assume that $x(n) = 3$, $x(n-1) = 2$, and $x(n-2) = 1$. Then, substituting these values into the above equation obtains the output $y(n) = x(n-D) = x(n-0.5) = 2.5$, which is equivalent to the noninteger-delayed (0.5 sample delayed) input signal $x(n)$.

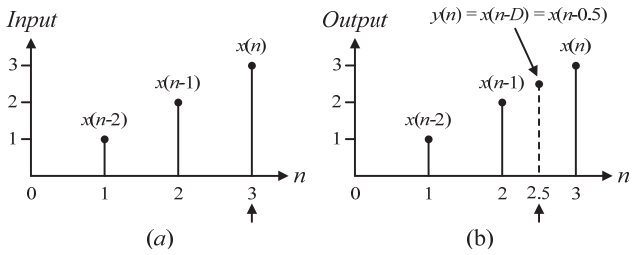

Fig. 7. Graphical representations of the input and output of the second-order Pascal noninteger-delay filter.

The graphical representation of the input and output of the second-order Pascal noninteger-delay filter are shown in Fig. 7 for understanding the mechanism using the Pascal noninteger-delay filter to delay the input signal $x(n)$ in a noninteger fashion. The tested results show that the Pascal noninteger-delay filter operates properly. Any kth-order Pascal noninteger-delay filter and its structure will be described in the next section.

Finally but more importantly, the transfer function of the second-order Pascal noninteger-delay filter will be formulated in this section. From (17), we can get

$$y(n) = x(n) + D\left[-\frac{3}{2}x(n) + 2x(n-1) - \frac{1}{2}x(n-2)\right] + D^2\left[\frac{1}{2}x(n) - x(n-1) + \frac{1}{2}x(n-2)\right].$$

Therefore,

$$\hat{H}(z,D) = H(z,D) = 1 + D\left[-\frac{3}{2} + 2z^{-1} - \frac{1}{2}z^{-2}\right] + D^2\left[\frac{1}{2} - z^{-1} - \frac{1}{2}z^{-2}\right]. \quad (18)$$

The transfer function (18) is equivalent to the direct form in (4), which verifies that our derived Pascal noninteger delay filter is the same polynomial-interpolation-type. However, we will not use the expression in (18) to realize it. Instead, the following subsections will propose a more efficient realization structure than the Farrow structures mentioned above.

4.2 Pascal Noninteger-Delay Filter and Its Structure

The Pascal noninteger-delay filtering consists of 2 operations. The first one passes the input signal $x(n)$ through the highpass filters that have transfer functions in (16) in order to obtain the Pascal coefficients, X_i (i = 0, 1, …, k). The transfer functions in (16) can be written again as

$$H_i(z) = \frac{Z\{X_i\}}{X(z)} = (1 - z^{-1})^i, \qquad i = 0,1,...,k. \quad (19)$$

In the second step, the Pascal coefficients X_i will go to the next operation as follows,

$$y(n) = \sum_{i=0}^{k} P_i(D) X_i \quad (20)$$

with

$$P_i(D) = \frac{(-1)^i D^{(i)}}{i!} \quad (21)$$

and $D^{(i)} = D(D-1)(D-2)...(D-i+2)(D-i+1)$.

In this step, we will consider X_i as the inputs of a system, and the coefficients $P_i(D)$ are the functions of the delay parameter D. Here, we call $P_i(D)$ the "Pascal-delayed" coefficients. The coefficients $P_i(D)$ control the delay of the Pascal noninteger-delay filter, which play a vital role in the Pascal noninteger-delay filter. Table 1 shows the coefficients $P_i(D)$ of the seventh-order Pascal noninteger-delay filter for various values of the delay parameter D. The Pascal noninteger-delay filter includes 2 systems, the first system is shown in (19), and the second one is given in (20). Consequently, the mechanism of the Pascal noninteger-delay filter is summarized in Fig. 8 for easy understanding.

Combining (19) and (20) gets the whole transfer function for the Pascal noninteger-delay filter as

$$\hat{H}(z,D) = P_0(D) + P_1(D)(1 - z^{-1}) + P_2(D)(1 - z^{-1})^2 + ... + P_k(D)(1 - z^{-1})^k$$

$$\hat{H}(z,D) = \sum_{i=0}^{k} P_i(D)(1 - z^{-1})^i \quad (22)$$

which can also be written in matrix form as

$$\hat{H}(z,D) = \sum_{i=0}^{k} P_i(D)(1 - z^{-1})^i = \hat{\mathbf{p}}^{\mathrm{T}}\mathbf{P}\mathbf{z} \quad (23)$$

where the Pascal-delayed coefficients vector $\hat{\mathbf{p}}$ is defined as

$$\hat{\mathbf{p}} = \begin{bmatrix} P_0(D) \\ P_1(D) \\ P_2(D) \\ \vdots \\ P_k(D) \end{bmatrix}, \qquad \mathbf{z} = \begin{bmatrix} 1 \\ z^{-1} \\ z^{-2} \\ \vdots \\ z^{-k} \end{bmatrix} \quad (24)$$

and $\mathbf{P}$ is the Pascal transform matrix as discussed in Sec. 3.

In order to realize the Pascal noninteger-delay filter, we will consider the first system and the second system separately as shown in Fig. 8.

The first system can be considered as a single-input, multi-output (SIMO) system, and we call it the front-section of the Pascal noninteger-delay filter. The second system is a multi-input, single-output (MISO) system, and

$P_i(D)$	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$	$i = 7$
$D = 0.00$	1.0000	0	0	0	0	0	0	0
$D = 0.25$	1.0000	–0.2500	–0.0938	–0.0547	–0.0376	–0.0282	–0.0223	–0.0183
$D = 0.50$	1.0000	–0.5000	–0.1250	–0.0625	–0.0391	–0.0273	–0.0205	–0.0161
$D = 0.75$	1.0000	–0.7500	–0.0938	–0.0391	–0.0220	–0.0143	–0.0101	–0.0076
$D = 1.00$	1.0000	–1.0000	0	0	0	0	0	0

Tab. 1. Pascal-delayed coefficients of the seventh-order Pascal noninteger-delay filter for various D.

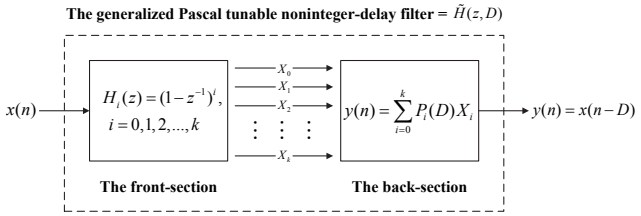

Fig. 8. The mechanism of the proposed Pascal noninteger-delay filter.

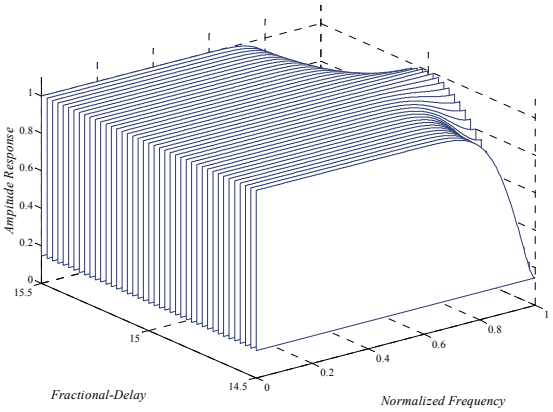

Fig. 9. Amplitude Response of the 30th-order Pascal noninteger-delay filter ($k=30$).

we call it the back-section of the Pascal noninteger-delay filter, which is the key of the Pascal noninteger-delay filter for adjusting the delay. Therefore, from the concept in Fig. 8, and (19) to (24), we can show the amplitude response of the Pascal noninteger-delay filter as Fig. 9 ($k=30$).

In Fig. 9, the value of delay parameter D in noninteger-delay is between $D \in [0.5k-0.5, 0.5k+0.5]$. The details for realizing both front-section and back-section of the Pascal noninteger-delay filter will be described in the next subsections.

4.3 Front-Section Realization

The front-section of Pascal noninteger-delay filter, which is the first system in Fig. 8, is used for filtering by the filter $H_i(z)$ in (19) in order to obtain the Pascal coefficients X_i. In fact, the Pascal coefficients are also the time-domain signals. After such Pascal coefficients are computed, they will be passed to the second system called the back-section.

In order to realize the front-section of Pascal noninteger-delay filter, we will be considered by factorizing the Pascal transform matrix **P** into binary {1, 0, –1} matrices. The front-section can be realized without any multiplications, and then the obtained front-section of Pascal noninteger-delay filter will be multiplierless digital filters.

The concept of front-section comes from the factorization of the Pascal transform matrix **P** into binary {1, 0, –1} matrices. From (19), we can write the vector of transfer function $H_i(z)$ as follows,

$$\begin{bmatrix} H_0(z) \\ H_1(z) \\ H_2(z) \\ \vdots \\ H_k(z) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & P_1(k) & P_2(k) & \cdots & P_k(k) \end{bmatrix} \begin{bmatrix} 1 \\ z^{-1} \\ z^{-1} \\ \vdots \\ z^{-k} \end{bmatrix} \quad (25)$$

or,

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \vdots \\ X_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & P_1(k) & P_2(k) & \cdots & P_k(k) \end{bmatrix} \begin{bmatrix} x(n) \\ x(n-1) \\ x(n-2) \\ \vdots \\ x(n-k) \end{bmatrix} \quad (26)$$

The Pascal coefficients X_i can be computed by using (26), i.e., the DPT of input vector **x**, where $\mathbf{x}^{\mathrm{T}} = [x(n)\ x(n-1)\ \ x(n-2)\ldots x(n-k)]$. Usually, the matrix transformation requires many multiplications and additions, which is dependent on the dimension of the used matrix operator. This paper proposes a method for factorizing the Pascal transform matrix **P** into binary {1, 0, –1} matrices, that will allow the transformation using matrix **P** to operate without multiplications and only additions are used. Therefore, the hardware realization for transformation circuits can be efficiently designed by using the so-called butterfly unit of DPT to establish the whole structure. Consider the 2×2 Pascal transform matrix **P**, which can be denoted as $\mathbf{P}_2$ (subscript 2 is the dimension of $\mathbf{P}_2$)

$$\mathbf{P}_2 = \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix}$$

and from (26), we can get

$$\begin{bmatrix} X_0 \\ X_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x(n) \\ x(n-1) \end{bmatrix}.$$

Also, from this matrix equation, we can make the data flow graph as Fig. 10,

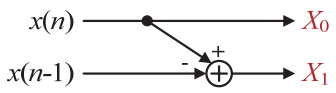

Fig. 10. Butterfly unit of the DPT.

This flow graph is called a butterfly unit of DPT [22] and used as a basic unit for constructing the front-section of any kth-order Pascal noninteger-delay filter. The method for factorizing the Pascal transform matrix **P** into binary {1, 0, –1} matrices is based on Gaussian elimination [23], which can be described as follows. The elimination matrix in [23] is used to factorize another kind of Pascal matrix (all elements in matrix are positive), but Pascal transform matrix used in DPT (alternating sign by $(-1)^i$; i is column index) is different from that in [23]. With some modification from [23], the elimination matrix **E** for the Pascal transform matrix **P** has entry $E_{x,x} = -1$, except $E_{0,0} = 1$,and $E_{x,x-1} = 1$, and all other entries are zero.

For example, the front-section of the third-order Pascal noninteger-delay filter can be shown as below

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -3 & 3 & -1 \end{bmatrix} \begin{bmatrix} x(n) \\ x(n-1) \\ x(n-2) \\ x(n-3) \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}}_{\text{stage 3}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}}_{\text{stage 2}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}}_{\text{stage 1}} \begin{bmatrix} x(n) \\ x(n-1) \\ x(n-2) \\ x(n-3) \end{bmatrix}.$$

This computation can be realized as Fig. 11.

Notice that the matrix **P** has the size $(k+1)^2$; k is the order of the Pascal noninteger-delay filter. For any kth-order case, the front-section can be realized in the similar fashion and shows the details as Fig. 12.

By considering the realization diagram, we know that the number of additions for the front-section of the Pascal noninteger-delay filter is

$$\text{No. of additions} = \frac{k(k+1)}{2}.$$

The front-section does not require any multiplications. Thus, the obtained front-section of the Pascal noninteger-delay filter contains only multiplierless digital filters.

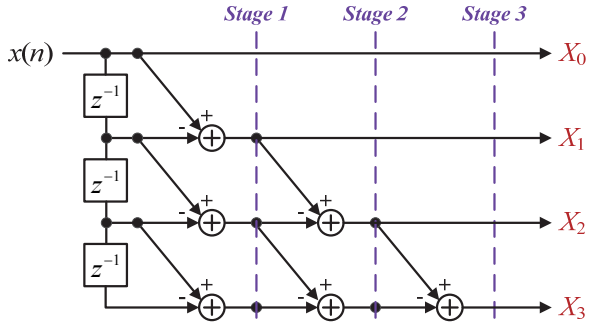

Fig. 11. An example for front-section of the third-order Pascal noninteger-delay filter.

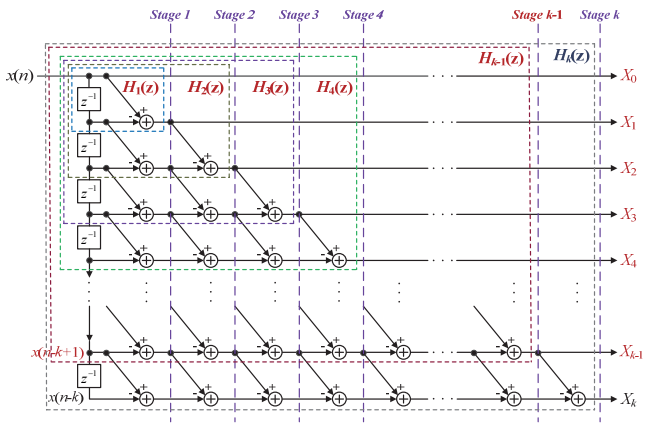

Fig. 12. The front-section of the kth-order Pascal noninteger-delay filter.

4.4 Back-Section Realization

The back-section of the Pascal noninteger-delay filter changes the delay of the Pascal noninteger-delay filter online, and it is realized by using (20) and (21). The front-section outputs X_i, and then X_i are fed to the back-section as

$$\begin{aligned} x(n-D) = y(n) &= \sum_{i=0}^{k} P_i(D) X_i \\ &= P_0(D)X_0 + P_1(D)X_1 + P_2(D)X_2 \\ &\quad + \ldots + P_{k-1}(D)X_{k-1} + P_k(D)X_k. \end{aligned} \tag{27}$$

From (21) and definition of falling factorial powers $D^{(i)}$ we have

$$\begin{aligned} y(n) = X_0 - DX_1 + \frac{1}{2}D(D-1)X_2 - \frac{1}{6}D(D-1)(D-2)X_3 \\ + \ldots + \frac{(-1)^{k-1}}{(k-1)!}D(D-1)(D-2)\ldots(D-k+2)X_{k-1} \\ + \frac{(-1)^k}{k!}D(D-1)(D-2)\ldots(D-k+1)X_k. \end{aligned} \tag{28}$$

Next, the Horner's rule is applied for sharing the common terms as

$$\begin{aligned} y(n) = X_0 + \Bigg[-X_1 + \frac{(D-1)}{2}\Bigg[X_2 + \ldots + \frac{(D-k-2)}{k-1}\Big[(-1)^{k-1}X_{k-1} \\ + (-1)^k \frac{(D-k-1)}{k}X_k \Big]\Bigg]\ldots\Bigg]. \end{aligned} \tag{29}$$

Consequently, by using (29), the back-section of the any kth-order Pascal noninteger-delay filter can be realized as Fig. 13.

Here, we can summarize the computational complexity of the back-section realization as

No. of multiplications = $3k - 3$

No. of additions = $2k - 1$.

Since the whole Pascal noninteger-delay filter consists of both the front-section and the back-section, and the front-section consists of only mutiplierless filters, all the multiplications of the Pascal noninteger-delay filter occur only from the back-section. The total computational complexity of the kth-order Pascal noninteger-delay filter will be considered in the next section and the computational complexities will be also compared with those of the direct-form Farrow structure as well as the Farrow structures after symmetry transformations proposed in [11–13].

5. Complexity of the Pascal Noninteger-Delay Filter

This section evaluates the Pascal noninteger-delay filter complexity by counting both multiplication and addition operations. For the example, the third-order Pascal

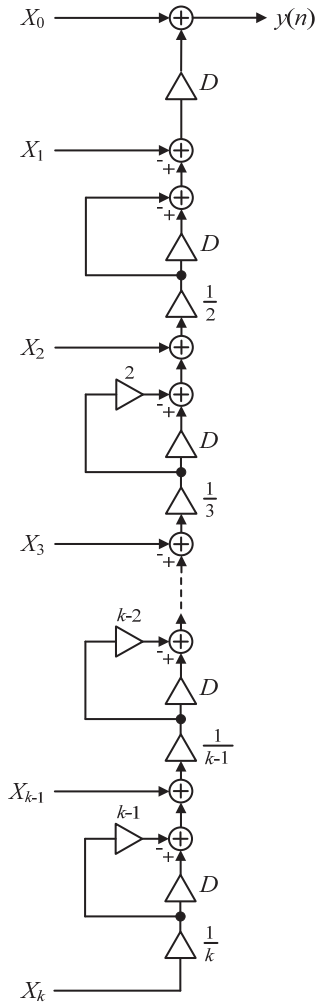

Fig. 13. The back-section of the *k*th-order Pascal noninteger-delay filter.

noninteger-delay filter can be realized by using the structures from front-section and back-section as shown in Fig. 14.

The third-order Pascal noninteger-delay filter in Fig. 14 needs 6 multiplications and 11 additions. Table 2 lists the numbers of multiplications and additions of the third-order polynomial-interpolation-type filters using three types of structures.

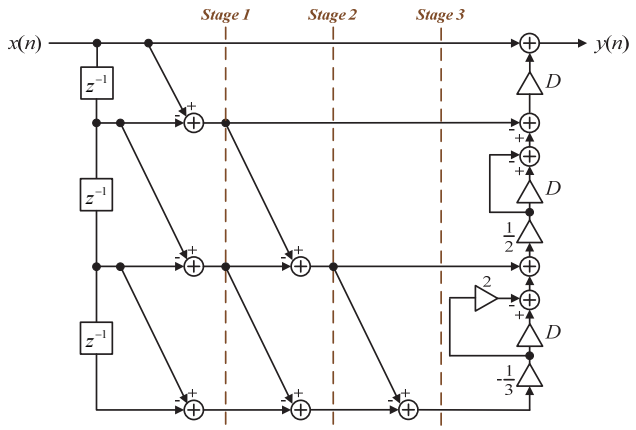

Fig. 14. The third-order Pascal noninteger-delay filter.

Type of filter	No. of Multiplications	No. of additions
Direct-form Farrow structure	15	12
Transformed Farrow structure	11	11
Pascal noninteger-delay filter	6	11

Tab. 2. Comparison of computational complexities of the third-order polynomial-interpolation-type filters.

Type of filter	No. of Multiplications	No. of additions
Direct-form Farrow structure	k^2+2k	k^2+k
Transformed Farrow structure in [12–14]		
(Odd-order)	$\frac{k^2+4k+1}{2}$	$\frac{k^2+4k+1}{2}$
(Even-order)	$\frac{k^2+3k}{2}$	$\frac{k^2+3k}{2}$
Pascal noninteger-delay filter	$3k-3$	$\frac{k^2+5k-2}{2}$

Tab. 3. Computational complexities of the *k*th-order polynomial-interpolation-type delay filters.

Clearly, the Pascal noninteger-delay filter has less complexity than other 2 types of the polynomial-interpolation-type filters, especially the number of multiplications is much less.

The complexity of the *k*th-order Pascal noninteger-delay filter is compared with the polynomial-interpolation-type noninteger-delay filters having the direct-form Farrow structure and the Farrow structures after symmetry transformations are shown in Tab. 3.

The computational complexities from the second-order to 30th-order of 3 types of polynomial-interpolation-type delay filters are shown in Fig. 15 and Fig. 16, respectively.

By using Fig. 15, we can compare the numbers of multiplications of 3 types of realization structures. As compared to the direct-form Farrow structure [1], the structures using symmetry transformations [11–13] can reduce the number of multiplications by about 50%. However, the number of multiplications required by the method

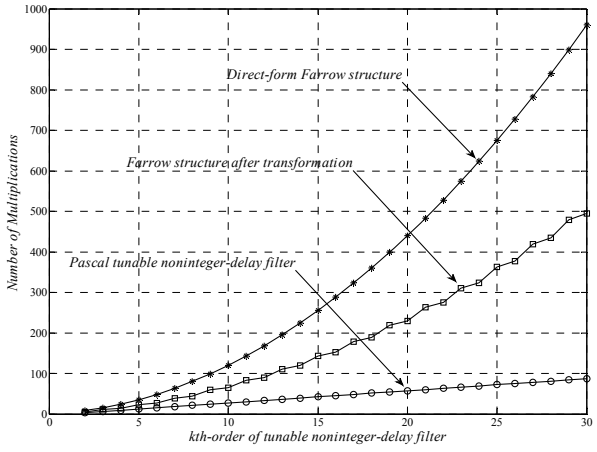

Fig. 15. Number of multiplications of 3 types of the polynomial-interpolation-type delay filters.

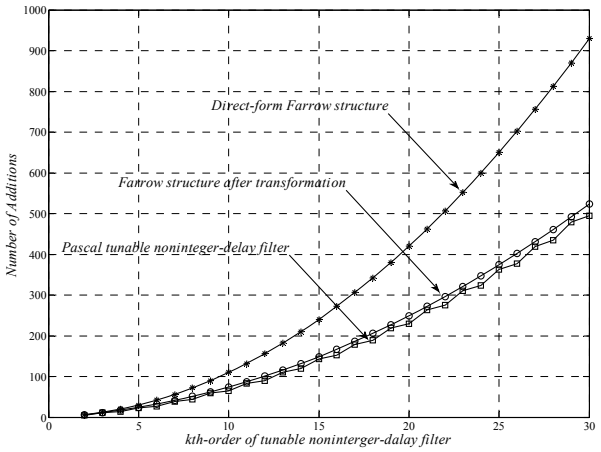

Fig. 16. Number of additions of 3 types of the polynomial-interpolation-type delay filters.

in [11–13] is not linear with respect to the order k, but it is a square function. The Pascal noninteger-delay filter requires fewer multiplications than the Lagrange-type with transformed Farrow structures [11–13].

The number of multiplications is linear with respect to the order k. In Fig. 16, we compare the numbers of additions of 3 types of polynomial-interpolation-type delay filters, the Pascal noninteger-delay filter requires less additions than the direct-form Farrow structure [1]. Obviously, since the Pascal noninteger-delay filter requires the lowest computational complexity, it is more suitable for fast filtering.

6. Conclusion

In this paper, we have derived a new filter realization structure for the polynomial-interpolation-type delay filter. Since it is formulated from the discrete Pascal transform (DPT) and Pascal interpolation, we call it Pascal noninteger-delay filter. The resulting Pascal noninteger-delay filter has a low-complexity structure that contains two sections (front-section and back-section). The former consists of multiplierless filters only, and the latter requires less multiplications. Because the proposed Pascal structure can online adjust the delay with less multiplications, it is suited for fast tuning applications. Moreover, we have compared the numbers of multiplications of 3 types of the polynomial-interpolation-type delay filters, and concluded that the Pascal structure requires the smallest number of multiplications. Consequently, the polynomial-interpolation-type delay filter can be achieved by using the Pascal approach with high efficiency and the lowest complexity filter structure.

References

[1] FARROW, C. W. A continuously variable digital delay element. In *Proceedings of IEEE International Symposium on Circuits Systems (ISCAS)*. Espoo (Finland), 1988, vol. 3, p. 2641–2645. DOI: 10.1109/ISCAS.1988.15483

[2] DENG, T.-B. Discretization-free design of variable fractional-delay FIR digital filters. *IEEE Transactions on Circuits and System II: Analog Digital Signal Processing*, 2001, vol. 48, no. 6, p. 637–644. DOI: 10.1109/82.943337

[3] ZHAO, H., YU, J.-B. A simple and efficient design of variable fractional delay FIR filters. *IEEE Transactions on Circuits and System II: Express Briefs*, 2006, vol. 53, no. 2, p. 157–160. DOI: 10.1109/TCSII.2005.856673

[4] TSENG, C.-C. Design of variable fractional delay FIR filters using differentiator bank. In *Proceedings of IEEE International Symposium on Circuits Systems (ISCAS 2002)*. Phoenix (Arizona, USA), 2002, vol. 4, p. 421–424. DOI: 10.1109/ISCAS.2002.1010481

[5] DENG, T.-B., NAKAGAWA, Y. SVD-based design and new structure for variable fractional-delay digital filters. *IEEE Transactions on Signal Processing*, 2004, vol. 52, no. 9, p. 2513–2527. DOI: 10.1109/TSP.2004.831922

[6] DENG, T.-B., LIAN, Y. Weighted-least-squares design of variable fractional-delay FIR filters using coefficient symmetry. *IEEE Transactions on Signal Processing*, 2006, vol. 54, no. 8, p. 3023–3038. DOI: 10.1109/TSP.2006.875385

[7] DENG, T.-B., QIN, W. Coefficient relation-based minimax design and low-complexity structure of variable fractional-delay digital filters. *Signal Processing*, 2013, vol. 93, no. 4, p. 923–932. DOI: 10.1016/j.sigpro.2012.11.004

[8] DENG, T.-B., QIN, W. Improved bi-equiripple variable fractional-delay filters. *Signal Processing*, 2014, vol. 94, no. 1, p. 300–307. DOI: 10.1016/j.sigpro.2013.07.004

[9] LIU, G.-S., WEI, C.-W. A new variable fractional sample delay filter with nonlinear interpolation. *IEEE Transactions on Circuits and System II: Analog Digital Signal Processing*, 1992, vol. 39, no. 2, p. 123–126. DOI: 10.1109/82.205818

[10] DENG, T.-B. Coefficient-symmetries for implementing arbitrary-order Lagrange-type variable fractional-delay digital filters. *IEEE Transactions on Signal Processing*, 2007, vol. 55, no. 8, p. 4078 to 4090. DOI: 10.1109/TSP.2007.893967

[11] DENG, T.-B. Symmetric structures for odd-order maximally flat and weighted-least-squares variable fractional-delay filters. *IEEE Transactions on Circuits and System I: Regular Papers*, 2007, vol. 54, no. 12, p. 2718–2732. DOI: 10.1109/TCSI.2007.905649

[12] DENG, T.-B. Transformation matrix for odd-order Lagrange-type variable fractional-delay filters. In *Proceedings of 6th International Conference on Information Communications and Signal Processing (ICICS 2007)*. Singapore, 2007, p. 1–5. DOI: 10.1109/ICICS.2007.4449557

[13] DENG, T.-B. Transformation matrix for even-order Lagrange-type variable fractional-delay digital filters. In *Proceedings of International Conference on Intelligent and Automation Systems (ICIAS 2007)*. Kuala Lumpur (Malaysia), 2007, p. 1179–1182. DOI: 10.1109/ICIAS.2007.4658570

[14] PEI, S.-C., TSENG, C.-C. A comb filter design using fractional-sample delay. *IEEE Transactions on Circuits and System II: Analog Digital Signal Processing*, 1998, vol. 45, no. 6, p. 649–653. DOI: 10.1109/82.673650

[15] DENG, T.-B. High-resolution image interpolation using two-dimensional Lagrange-type variable fractional-delay filter. In *Proceedings of International Symposium on Nonlinear Theory and Applications (NOLTA 2005)*. Bruges (Belgium), 2005, p. 214–217.

[16] SHYU, J.-J., PEI, S.-C., CHAN, C.-H. Minimax phase error design of allpass variable fractional-delay digital filters by iterative weighted least-squares method. *Signal Processing*, 2009, vol. 89, no. 9, p. 1774–1781. DOI: 10.1016/j.sigpro.2009.03.021

[17] TSENG, C.-C. Closed-form design of digital IIR integrators using numerical integration rules and fractional sample delays. *IEEE Transactions on Circuits and System I: Regular Papers*, 2007, vol. 54, no. 3, p. 643–655. DOI: 10.1109/TCSI.2006.887641

[18] SHYU, J.-J., PEI, S.-C., HUANG, Y.-D. Two-dimensional Farrow structure and the design of variable fractional delay 2-D FIR digital filters. *IEEE Transactions on Circuits and System I: Regular Papers*, 2009, vol. 56, no. 2, p. 395–404. DOI: 10.1109/TCSI.2008.2001828

[19] TSENG, C.-C. Design of 1-D and 2-D variable fractional delay allpass filters using weighted least-squares method. *IEEE Transactions on Circuits and System I: Fundamental Theory Applications*, 2002, vol. 49, no. 10, p. 1413–1422. DOI: 10.1109/TCSI.2002.803361

[20] ABURDENE, M. F., GOODMAN, T. J. The discrete Pascal transform and its applications. *IEEE Signal Processing Letters*, 2005, vol. 12, no. 7, p. 493–495. DOI: 10.1109/LSP.2005.849498

[21] GOODMAN, T. J., ABURDENE, M. F. Interpolation using the discrete Pascal transform. In *Proceedings of 40th Annual Conference on Information Sciences and Systems (CISS 2006)*. New Jersey (USA), 2006, p. 1079–1083. DOI: 10.1109/CISS.2006.286626

[22] SKODRAS, A. N. Efficient computation of the discrete Pascal transform. In *Proceedings of 14th European Signal Processing Conference (EUSIPCO 2006)*. Florence (Italy), 2006, p. 1–4. DOI: 10.1109/CISS.2006.286626

[23] EDELMAN, A., STRANG, G., USA. *Pascal Matrices*. 12 pages. [Online] Cited 2014-05-19. Available at: http://www-math.mit.edu/~gs/papers/pascal-work.pdf

About the Authors ...

Parinya SOONTORNWONG was born in Chonburi, Thailand. He received his B.S.Id.Ed. in Telecommunication Engineering from King Mongkut's Institute of Technology Ladkrabang (KMITL) in 2000 and M. Eng. in Electrical Engineering from Prince of Songkla University (PSU) in 2006. He is a D. Eng. candidate student in Electrical Engineering in King Mongkut's Institute of Technology Ladkrabang (KMITL) support by Thai Government Science and Technology Scholarship, Ministry of Science and Technology (MOST) Scholarship. His research interests include digital filter design and digital signal processing (DSP).

Sorawat CHIVAPREECHA was born in Nakornpathom, Thailand. He is an Assistant Professor with the Department of Telecommunication Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang (KMITL). He received his B. Eng. in Telecommunication Engineering from Suranaree University of Technology (SUT) in 1998, M. Eng. and D. Eng. in Electrical Engineering from King Mongkut's Institute of Technology Ladkrabang (KMITL), in 2001 and 2008, respectively. His research interests include digital filter design and implementation, VLSI for digital signal processing, information science and satellite engineering.