International Journal of Information Systems in the Service Sector
Volume 8 • Issue 2 • April-June 2016

Software Implementation of Real-time Discrete Wavelet Transform Algorithm with Filter Banks

Nikolajs Bogdanovs, Department of Transport Electronics and Telematics, Riga Technical University, Riga, Latvia
Elans Grabs, Department of Transport Electronics and Telematics, Riga Technical University, Riga, Latvia
Ernests Petersons, Department of Transport Electronics and Telematics, Riga Technical University, Riga, Latvia

ABSTRACT

This article describes real-time discrete wavelet transform algorithm implementation for high-level programming language. The article describes multiscale transform algorithms both for direct discrete wavelet transform and inverse discrete wavelet transform. This algorithm has been implemented in C++ programming language and tested with Raspberry Pi microprocessor system. This article proposes the improved delay line algorithm without full shifting of register. New algorithm requires single reading operation, single writing operation and one division calculation for any length of delay line. The article includes experimental measurements of processing time on Raspberry Pi for various scale numbers. The algorithm described in this article can be used with any software tool capable of using high level programming language, for example Matlab, Octave, Opnet, etc. This is the main purpose – to create algorithm which is not tied strictly to hardware implementation but also, nonetheless, provides real-time discrete wavelet analysis capability.

KEYWORDS

Discrete Wavelet Transform, Filter Banks, Multiscale Analysis, Raspberry Pi Microprocessor System

INTRODUCTION

The brief analysis of literature shows that multiscale discrete wavelet transform software implementation is not widely discussed. The main interest in most works is concerned with hardware implementation instead. The authors of this article couldn't find any real-time algorithm proposed for discrete wavelet transform calculation. It is possible to segment data and process these segments in real-time if performance of the system is high enough. However the length of segments is limited from below by the number of analysis scales. So, for multiscale analysis it is necessary to increase the length of the segments.

DOI: 10.4018/IJISSS.2016040106

Mallat pyramidal algorithm is being used for most of implementations and it is described in details in many sources, for example in (Strang & Ngueyen, 1996) or (Mallat, 1999). According to this algorithm, data segment is being accumulated and further is processed by $j = 1$ scale filter bank. The result is composed of approximation coefficients $a\left[k\right]$ and detail coefficients $d\left[k\right]$. The approximation coefficients are being processed further by $j = 2$ scale filter bank and this process is iteratively being repeated. It is possible to process data in real-time if segments are small enough and data rate is very high, for instance in master's thesis (Bomers, 2000) such approach is described for audio signal real-time processing. Other works and articles describe segmentation with or without overlapping in order to minimize the border effect of segmentation (for example, in (Prusa & Rajmic, 2011)).

There is wide variety of hardware implementation descriptions. It is very popular to implement such filter banks with Field Programmable Gate Array (FPGA) logic, examples are provided in (Bahoura & Ezzaidi, 2010; Cavuslu & Karakaya, 2010)). Also such implementations are described in details in master's thesis (Sripath, 2003). This work proposes different implementations – direct, polyphase, lattice structures. There are many articles with performance improvements, such as (Jing & Yuan Bin, 2007; Wenbing & Yingmin, 2008). Another optimization is related to decreased power consumption for multiplications, for example in (O'Brien & Conway, 2008).

There are discrete wavelet transform implementation algorithms for DSP processors as well, for example (Ben Hnia Gazzah *et al*., 2008; QiWei Lin *et al*., 2009), including parallel processing in (Wilburn & Alexander, 1994) to increase computational efficiency of discrete wavelet transform algorithm.

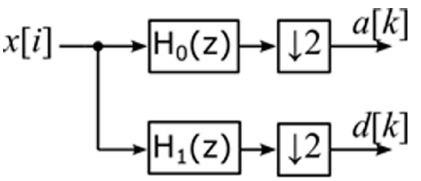
The publishing years of most of these works show that this topic is well researched and evolved. Of course, there are new works describing multidimensional transforms and more efficient processing algorithms, for example in (Darji *et al*., 2014; Darji *et al*., 2014). In other works the algorithms are being optimized for specific tasks, few of many examples are (Sardar & Babu, 2014; Cutajar *et al*., 2014). Another important topic is energy efficiency, which is also being researched at the present moment and some publications can give insight in this, for example (Tang-Hsuan *et al*., 2014) uses discrete wavelet transform for neuron network processing.

However the hardware implementation is very specific to selected platform and can't be used generally in software, for example to perform simulations. It would be necessary to simulate this hardware platform in the first place. When discussing general purpose and microprocessor or microcontroller systems, we were unable to find such real-time algorithm. In such case the segmentation is being used and filter banks process entire segment either via discrete convolution calculation or by using FFT. At the same time such filter banks are easy to implement directly. The performance can be further improved by implementing polyphase or lattice filter banks.

REAL-TIME DISCRETE WAVELET TRANSFORM

The main disadvantage of segmentation based algorithms is impossibility to use result immediately – the entire segment has to be processed first. Since filter bank shown in Figure 1 includes downsampling by factor of 2, the segment length for each consecutive step (scale) is reduced by 2 times. That means, for multiscale analysis over J scales it is necessary to assemble segment with length 2^J at the first scale. The higher is number of scales, the greater segment must be. For very high numbers of scales it is impossible to calculate even the first scale coefficients unless segment has been accumulated. However, such digital filtering should be possible in real-time and it is possible to provide approximation and detail coefficients at least for lower numbered scales.

Figure 1. Direct discrete wavelet transform with filter bank



Filter banks usually consist of Finite Impulse Response (FIR) filters, and as of such the time delay before first coefficient at the filter output can be easily estimated for any scale number j. This delay determines the time necessary to wait until the first coefficient value, the further coefficients shall follow without additional time delay. It is important to know this delay in order to perform perfect signal reconstruction.

The generalized Daubechies-3 wavelet transform filters have order of $N = 5$, which are used for analysis (decomposition) and synthesis (reconstrution) filter bank implementation. These filters have different impulse response coefficients:

- **Analysis Filter Bank:** lowpass filter with $h_0\left[n\right]$ and highpass filter with $h_1\left[n\right]$, where $n = 0,1,2,\ldots,N$;
- **Synthesis Filter Bank:** lowpass filter with $f_0\left[n\right]$ and highpass filter with $f_1\left[n\right]$, where $n = 0,1,2,\ldots,N$;

For orthogonal filter bank (such is the case of Daubechies-3 wavelets) transfer functions for any of these filters can be calculated from analysis filter bank lowpass filter impulse response coefficients $h_0\left[n\right]$ according to following equations (Strang & Ngueyen, 1996) for analysis filter bank highpass filter:

$$H_1\left(z\right) = z^{-N} H_0\left(-z^{-1}\right) == \sum_{n=0}^{N}\left(-1\right)^n h_0\left[n\right] z^{-\left(N-n\right)}, \tag{1}$$

where N is order of the filter, synthesis filter bank lowpass (approximation) filter:

$$F_0\left(z\right) = z^{-N} H_0\left(z^{-1}\right) == \sum_{n=0}^{N} h_0\left[n\right] z^{-\left(N-n\right)}, \tag{2}$$

and synthesis filter bank highpass (detail) filter:

$$F_1\left(z\right) = -H_0\left(-z^{-1}\right) == -\sum_{n=0}^{N}\left(-1\right)^n h_0\left[n\right] z^{-n}. \tag{3}$$

Since it is proposed to implement an algorithm in software with high level programming language (specifically, C++ has been chosen), it is necessary to save the state for each filter at every scale to use it for next signal sample processing. These states $MEM_0\left[n\right]$ and $MEM_1\left[n\right]$ are the same for

analysis filter bank, since both filters are FIR filters and have the same input samples. Therefore they share a common delay line with state $MEM\left[n\right],\ n = 0,1,2,\ldots,N$.

In such case, for Daubechies-3 wavelets, the structure of analysis filter bank approximation/detail filters can be formed according to Figure 2. The length of delay line is equal to the number of FIR filter impulse response coefficients $L = N + 1$. The approximation filter coefficients $h_0\left[n\right]$ and detail filter coefficients $h_0\left[n\right]$ have to be specified as well for $n = 0,1,2,\ldots,N$:

$$a_k = h_0\left[0\right]x_i + \sum_{n=1}^{N} MEM\left[n-1\right]h_0\left[n\right], \tag{4}$$

where x_i is current input sample and

$$d_k = h_1\left[0\right]x_i + \sum_{n=1}^{N} MEM\left[n-1\right]h_1\left[n\right]. \tag{5}$$

Both of these discrete convolutions can be calculated in single cycle with further time shift of the filter state. As a result of function calculation the approximation coefficient a_k and detail coefficient d_k are being determined for every input signal sample x_k . The last previous $L = N + 1$ signal samples are saved in filter state.

The algorithm for this function calculates a_k and d_k coefficients for every input sample. The filter bank includes downsampling operator, which removes either odd, or even numbered samples (the choice can be made) leaving the half of original number of samples. As of such, after calculation the result is ready to be used immediately and can be sent to the next scale analysis filter bank input. This process can be iteratively repeated and still will yield result immediately after it has been calculated. There is no need to wait for entire segment processing.

Figure 2. Digital filter bank with common delay line for Daubechies-3 wavelets

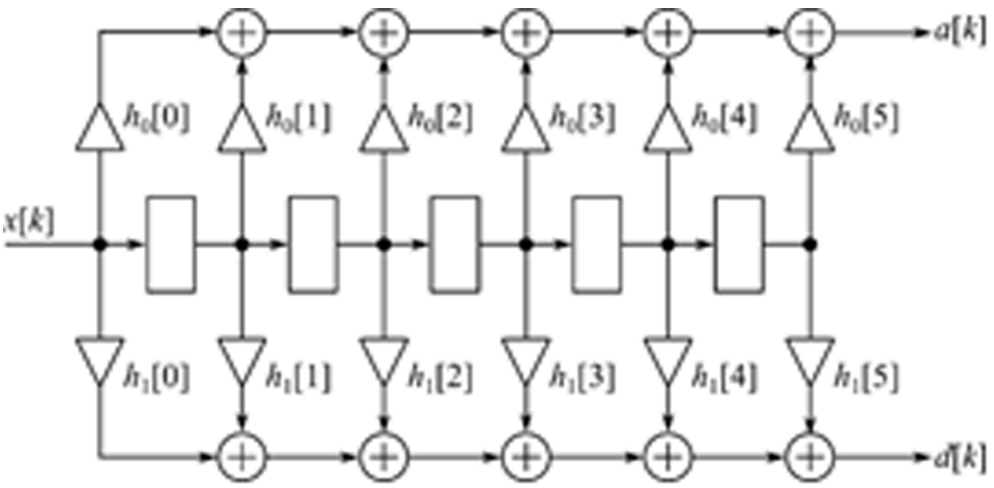


Figure 3. Discrete wavelet transform filter bank delay line shift operation algorithm

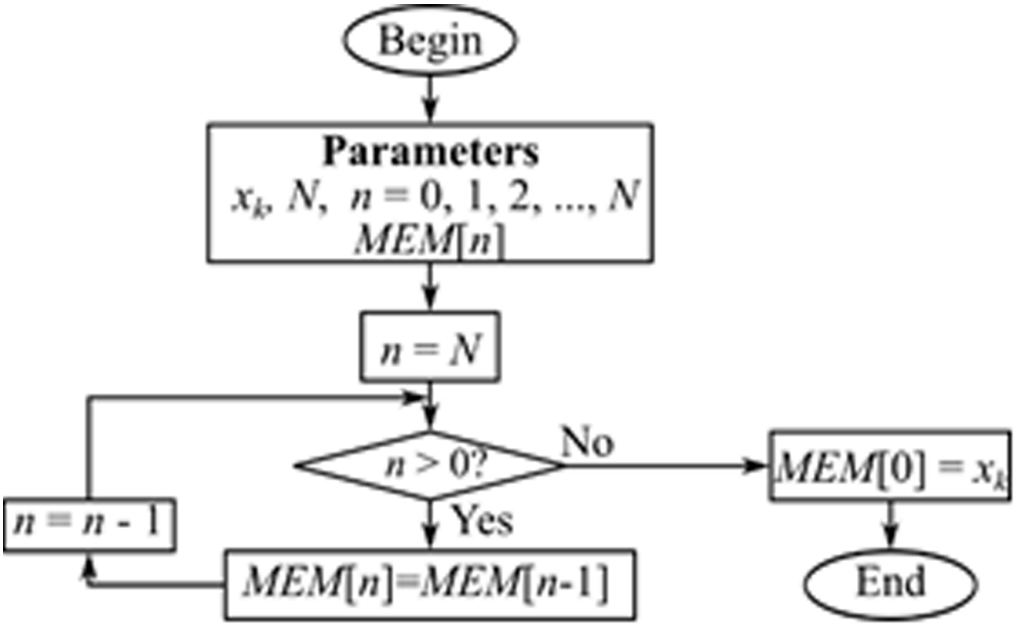
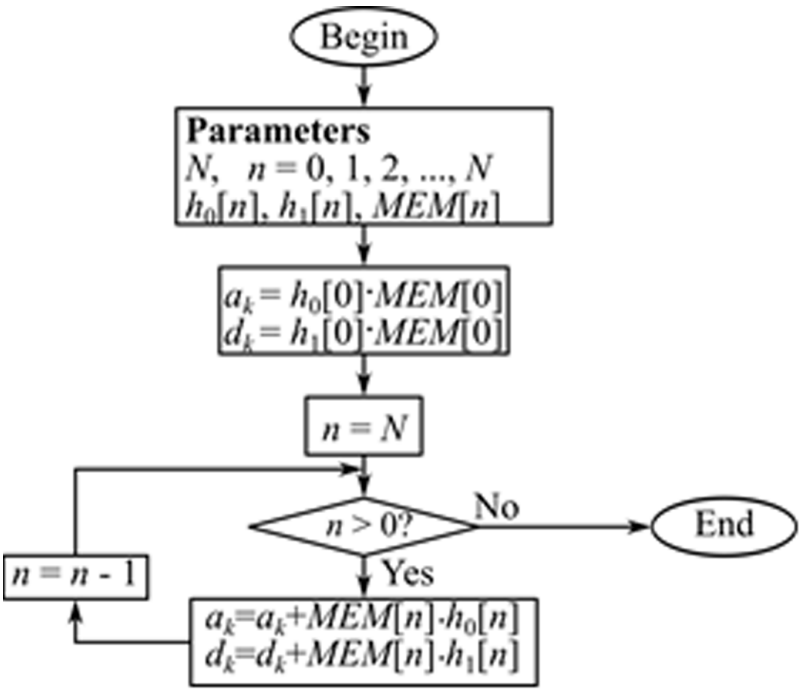


Figure 4. Discrete wavelet transform filter bank convolution operation algorithm



It is inefficient to calculate all coefficients, when half of them will be removed by downsampling. Therefore, the function has been divided in two parts: time shift part in Figure 3 and convolution part in Figure 4. If the sample is skipped (its number is either odd, or even) then only time shift part is being applied. Otherwise, sample is being saved and both algorithm parts apply.

REAL-TIME INVERSE DISCRETE WAVELET TRANSFORM

The inverse discrete wavelet transform is also implemented with filter banks – the synthesis filter bank shown in Figure 5 with approximation filter transfer function $F_0\left(z\right)$ and detail filter transfer function $F_1\left(z\right)$. This filter bank has two inputs – approximation coefficients input and detail coefficients input respectively. Before filtering, the coefficients are being upsampled, i.e. the missing samples from downsampling are replaced by zeroes.

The transfer functions for synthesis filter bank can be expressed from analysis filter bank transfer functions according to (2) for approximation filter and (3) for detail filter. After filtration procedure the outputs of both filters are added and signal samples x_i are reconstructed. In case of multiscale transform the higher scale $j-1$ approximation coefficients $a_{j-1,i}$ are reconstructed. It is obvious, the inverse wavelet transform is calculated for all scales in opposite order comparing to direct discrete wavelet transform, i.e. $J \geq j \geq 1$, where J is the number of scales.

Figure 6 shows algorithm for inverse discrete wavelet transform with filter banks. The input samples for two filters are different – for lowpass filter with $F_0\left(z\right)$ the input data are approximation coefficients $a\left[k\right]$ and for highpass filter with $F_1\left(z\right)$ the input data are detail coefficients $d\left[k\right]$. This means, that contrary to direct transform algorithm, the inverse transform algorithm requires to store each filter state separately in $MEM_0\left[n\right]$ and $MEM_1\left[n\right]$ accordingly, $n = 0,1,2,\ldots,N$. The length of each state equals to length of filter impulse response $L = N+1$.

Similar to direct discrete wavelet transform algorithm, the next part is calculation of two discrete convolutions to be added for sample y_k reconstruction:

$$y_0 = f_0\left[0\right]a_k + \sum_{n=1}^{N} MEM_0\left[n-1\right]f_0\left[n\right], \tag{6}$$

and

$$y_1 = f_1\left[0\right]d_k + \sum_{n=1}^{N} MEM_1\left[n-1\right]f_1\left[n\right]. \tag{7}$$

Figure 5. Real-time inverse discrete wavelet transform with filter bank

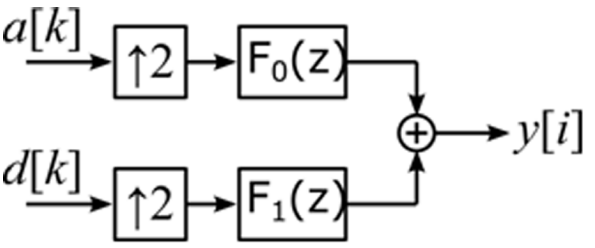
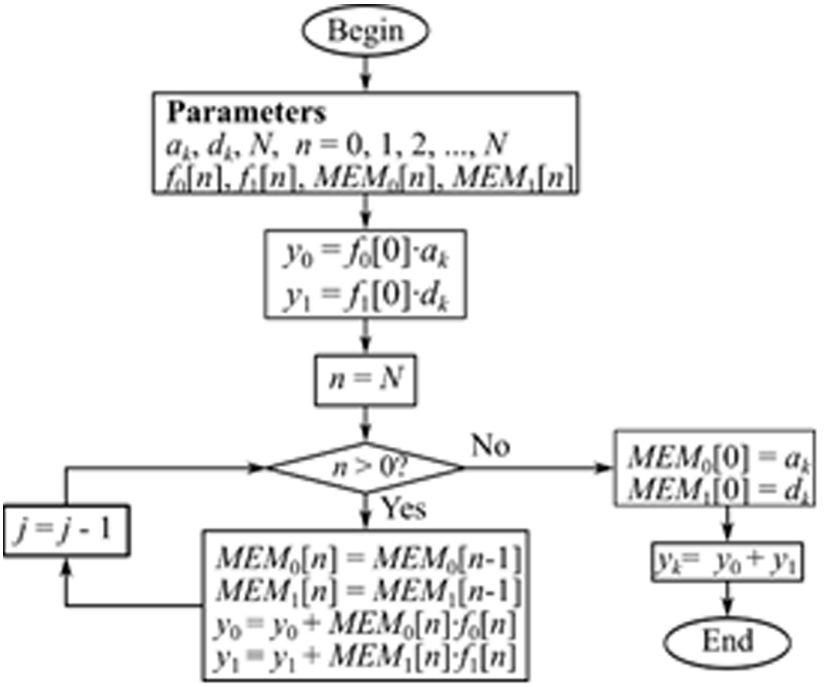


Figure 6. Real-time inverse discrete wavelet transform algorithm



Both discrete convolutions are calculated in single cycle with further time shift of the filters states. As a result of this function there are two parts of output sample: approximation part y_0 and detail part y_1. These parts are added to calculate value of the output sample: $y_k = y_0 + y_1$.

As it has been done with analysis filter bank, the sampling frequency is left unchanged. There is no upsampling operation in this algorithm from filter bank shown in Figure 5. This upsampling has to be done before calling of this function. Upsampling inserts "0" value after each approximation coefficient a_k and detail coefficient d_k, so the original number of samples can be restored: $a_1,0,a_2,0,\ldots$ and $d_1,0,d_2,0,\ldots$

REAL-TIME MULTISCALE DISCRETE WAVELET TRANSFORM

The real-time discrete wavelet transform algorithm can be generalized for J scales to calculate approximation coefficients $\{a_{j,k}\}$ and detail coefficients $\{d_{j,k}\}$ for all scales

$j = 1,2,\ldots,J$.

In order to perform a perfect signal reconstruction for multi-scale wavelet transform all detail coefficients $d_{j,k}$ are required for every scale j as well as last scale J approximation coefficients $a_{J,k}$. For other scales $j = 1,2,\ldots,J-1$ approximation coefficients $a_{j,k}$ are not required, though algorithm described further allows to save them as well.

Figure 7 shows algorithm of real-time discrete wavelet transform calculation implementation over J scales. This algorithm is based on filter banks shown in Figure 2. The algorithm requires approximation filter impulse response coefficients $h_0[n]$ and detail filter impulse response coefficients $h_1[n]$ for $n = 0,1,2,\ldots,N$. It is also necessary to specify required number of scales J. This number is the upper limit of calculated scales, so if the number of processed samples allows calculation for bigger scale, only specified J scales will be processed.

The number of scales J is equal to the number of used filter banks from Figure 2. These filter banks are completely equivalent and have transfer functions $H_0(z)$ and $H_1(z)$ and they are implemented by performing both algorithms shown in Figure 3 and Figure 4. The filter state $MEM[j][n]$ for each scale j has to be saved separately, so the filters state matrix has dimensions $J \times L$, where L is the number of impulse response coefficients. Every row of this matrix $MEM[j][n]$ is filter state for scale j. All filter states are set to zeros at the beginning of calculation. The indexes of approximation and detail coefficients $k(j)$ are also set to zeros for every scale

$j = 0,1,2,\ldots,J-1$.

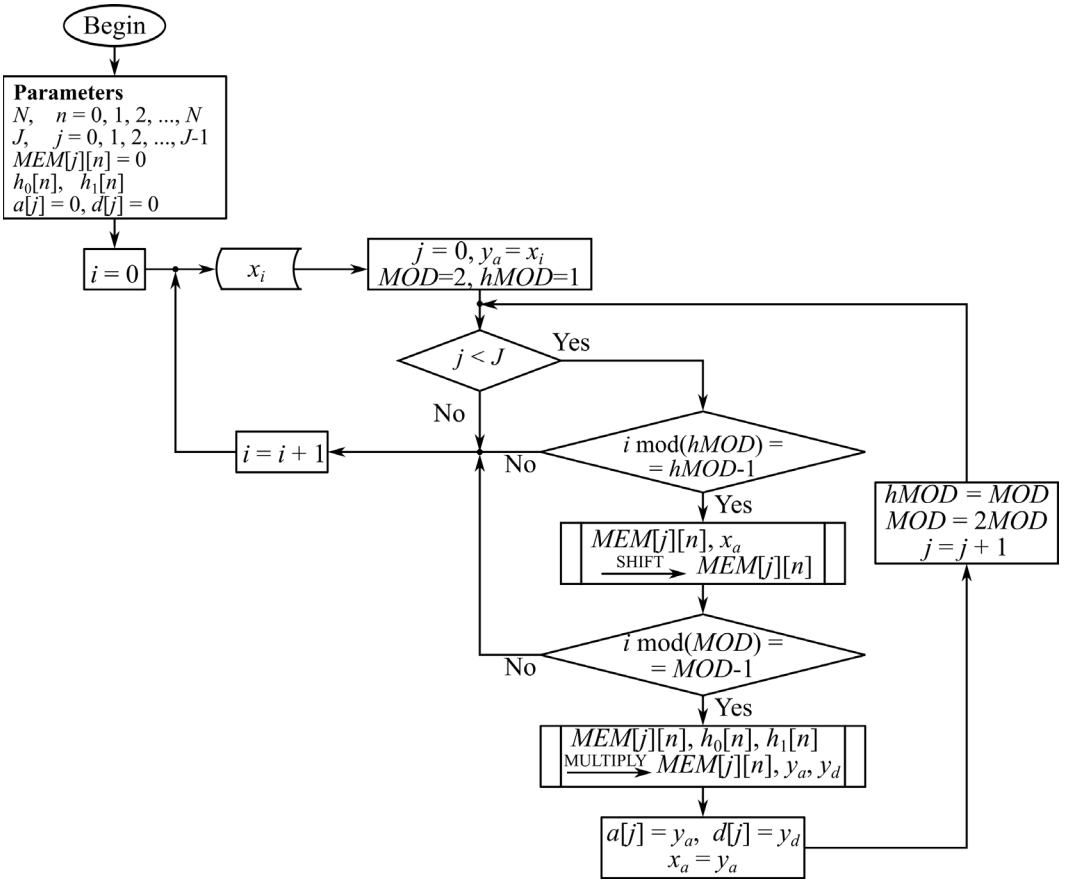
The first signal sample x_i is at $i=0$ on the input of the first filter bank. Since for filter bank with scale (number) $j=2$ the input data are previous scale approximation coefficients $a_{j-1,k}$, it is convenient to assume that signal samples are approximation coefficients at the scale $j=0$, i.e. $a_{0,i} = x_i$. This complies with discrete wavelet transform concept, when from coarse approximation coefficients $a_{j,k}$ with every further scale the more coarse coefficients are extracted (low-frequency components) and more details (high-frequency components. The values of approximation and detail coefficients are saved at y_a and y_d, accordingly. The y_a values become more coarse with every scale, so it is safe to assume at the beginning of algorithm that $y_a = x_i$.

The algorithm calculates transforms of this sample for every scale j up to specified limit J, when that is possible, or up to maximal possible scale, based on the number of the sample. After filtering there is downsampling operation by factor of 2^j for scale j. So, it is possible to state, that downsampling before filtering is made by factor of 2^{j-1} for scale j (no downsampling at the input for $j=1$). After such operation only samples with numbers multiple of 2^{j-1} remain. These samples are sent to the input of filter bank with further downsampling by factor of 2 (in this algorithm only odd numbered samples are saved). The coefficients are saved at current scale and the number of scales is increased by one. If the x_i sample number i isn't multiple of specified falue, the sample is skipped at this scale and every next scale after that, meaning that processing of this sample has ended.

For bigger scales the number of output signal samples – approximation coefficients is decreased by factor of 2. So for the first scale $j=1$ every multiple number of 2^1 sample is being saved, for scale $j=2$ – every multiple number of 2^2 sample is being saved (save as the every 2-nd sample of $j=1$ samples), for $j=3$ scale – every multiple number of 2^3 and so on.

It is possible to precisely estimate the number of wavelet transformation scales based on the number of sample i. It is notable, that although not every coefficient is necessary to be calculated and saved, every coefficient of previous scale affects the filter state. The downsampling by factor of 2 is at the output of the filter, so that means the filter still calculates these obsolete samples. This is inefficient approach, since it is possible to reduce the number of multiplication operations by 2 times, increasing the performance of algorithm. In the algorithm described this is achieved by splitting

Figure 7. *Real-Time multiscale discrete wavelet transform for J scales with filter banks*



filtering procedure into time shifting and convolution calculation. The time shifting is performed for every input sample and the convolution is calculated only for odd-numbered input samples.

Generalizing described above:

- At the scale $j=1$ every multiple of $\Delta_1=1$ numbered sample is processed and every multiple of $\Delta_2=2$ numbered coefficients are saved;
- At the scale $j=2$ every multiple of $\Delta_1=2$ numbered sample is processed and every multiple of $\Delta_2=4$ numbered coefficients re saved;
- At the scale $j=3$ every multiple of $\Delta_1=4$ numbered sample is processed and every multiple of $\Delta_2=8$ numbered coefficients re saved;
- At the scale $j=J$ every multiple of $\Delta_1=2^{J-1}$ numbered sample is processed and every multiple of $\Delta_2=2^J$ numbered coefficients re saved.

Therefore, for every scale $j=1,2,\ldots,J$ there is a check to determine from number of sample i whether there is filtering at the current scale to be done. The check is made by calculating a remainder from division of i by $hMOD=2^{j-1}$ according to:

$$R_1 = i \bmod \left(hMOD\right),\ i = 0,1,2,\ldots,K, \tag{8}$$

where K is the number of input signal samples.

When remainder R_1 is maximal $\max\left(R1\right) = hMOD - 1$, the sample is being processed by performing time shift according to algorithm shown in Figure 3. For any other case the sample must be removed by downsampling at previous scales, so processing of this sample should be interrupted, since there is no data to the filter bank input. The next sample $x_i + 1$ is being processed from the scale $j=1$. Note, at the scale $j=1$ $hMOD=1$ and condition (8) is true, since $\max\left(R1\right) = hMOD - 1 = 0$. Thus at the scale $j=1$ time shift processing is performed for every input sample.

If the time shifting has been performed, then the filter response (convolution) is calculated for every 2-nd sample (odd numbered) according to algorithm shown in Figure 4. This means that coefficient is being saved if the maximal remainder of sample number *i* is from division by 2 times greater divisor:

$$R_2 = i \bmod \left(MOD\right),\ i = 0,1,2,\ldots,K, \tag{9}$$

where *K* is the number of input samples.

When (9) check yields maximum $\max\left(R2\right) = MOD - 1$, the convolutions for filter bank responses are calculated according to algorithm from Figure 4. The output values y_a and y_d are saved at the scale j coefficients $a_{j,k} = a\left[j\right]$ and $d_{j,k} = d\left[j\right]$ accordingly. Otherwise the calculation of convolutions is skipped, since this data will be removed after downsampling.

REAL-TIME MULTISCALE INVERSE DISCRETE WAVELET TRANSFORM

Similar to analysis filer banks, the synthesis filter banks used for inverse discrete wavelet transform operate in real-time. This allows calculating inverse discrete wavelet transform for every sample at maximal scale J in real-time. However, it is necessary to provide all the detail coefficients from every smaller scale. The input data for real-time multiscale inverse discrete wavelet transform will be (assuming scale numeration is starting from $j = 0$ instead of $j = 1$, for convenience for use in programming languages):

- $j = J - 1$ scale approximation coefficient $a_{J-1}\left[k\right]$ and detail coefficient $d_{J-1}\left[k\right]$, $k = 0$, total coefficients number $KK\left[J-1\right] = 1$;
- $j = J - 2$ scale detail coefficients $d_{J-1}\left[k\right]$, $k = 0,1$, with total number of coefficients $KK\left[J-2\right]=2$;
- For every smaller scale j the total number of coefficients increases twice until the number 2^{J-1} at scale $j=0$.

In general, for successful reconstruction of signal up to the scale $j=0$, for arbitrary scale j it is necessary to provide $KK\left[j\right]$ detail coefficients. This number can be estimated as follows:

$$KK\left[j\right] = 2^{J-j-1}, \tag{10}$$

where $j = 0,1,\ldots,J-1$ and J the number of scales for discrete wavelet transform.

As it can be easily seen from (10), for every transformed scale the number of required coefficients is doubled. This means it is impossible to perform inverse discrete wavelet transform unless all required coefficients for all scales have been accumulated in memory. Basically, the true real-time transform can't be achieved even for hardware implementation with FPGA or DSP, since it's necessary to delay all coefficients before the highest scale J coefficients a and d have been received. These coefficients correspond to last (or first, depending on downsampling operation) received coefficients for lower scales $j<J$. So for high number of scales J it is impractical to expect high performance due to long segments of coefficients. In real-time hardware implementation such real-time processing can be achieved with delay. If downsampling has been performed so that only even-numbered samples are saved, this delay can be estimated at scale j from higher scale $j+1$ delay according to the expression:

$$L_{\text{delay}}\left[j\right] = 2L_{\text{delay}}\left[j+1\right] + N. \tag{11}$$

When downsampling saves only odd-numbered samples, the same delay can be calculated according to following expression:

$$L_{\text{delay}}\left[j\right] = 2L_{\text{delay}}\left[j+1\right] + N - 1. \tag{12}$$

Here N is the order of filters used in analysis filter bank. Note, the sampling frequency decreases when number of scale j grows, meaning that all sets of coefficients for every scale exist over the same time interval.

The algorithm of multiscale real-time inverse wavelet transform algorithm is shown in Figure 8. This algorithm is based on algorithm of synthesis filter bank from Figure 6. This algorithm of filter bank has been extended by adding accumulation of coefficients, the time delay of coefficients and upsampling operation.

In this algorithm the greatest number of operations is related to delay lines operation via shift register according to algorithm shown in Figure 3. Such approach has important disadvantage: according to (12) expression the total delay line length for lowest scale $j=1$ can achieve value $L_{\text{delay}}=4092$ at highest scale $J=10$ for Daubechies-3 wavelet filter bank with order $N=5$. If the highest scale is increased up to

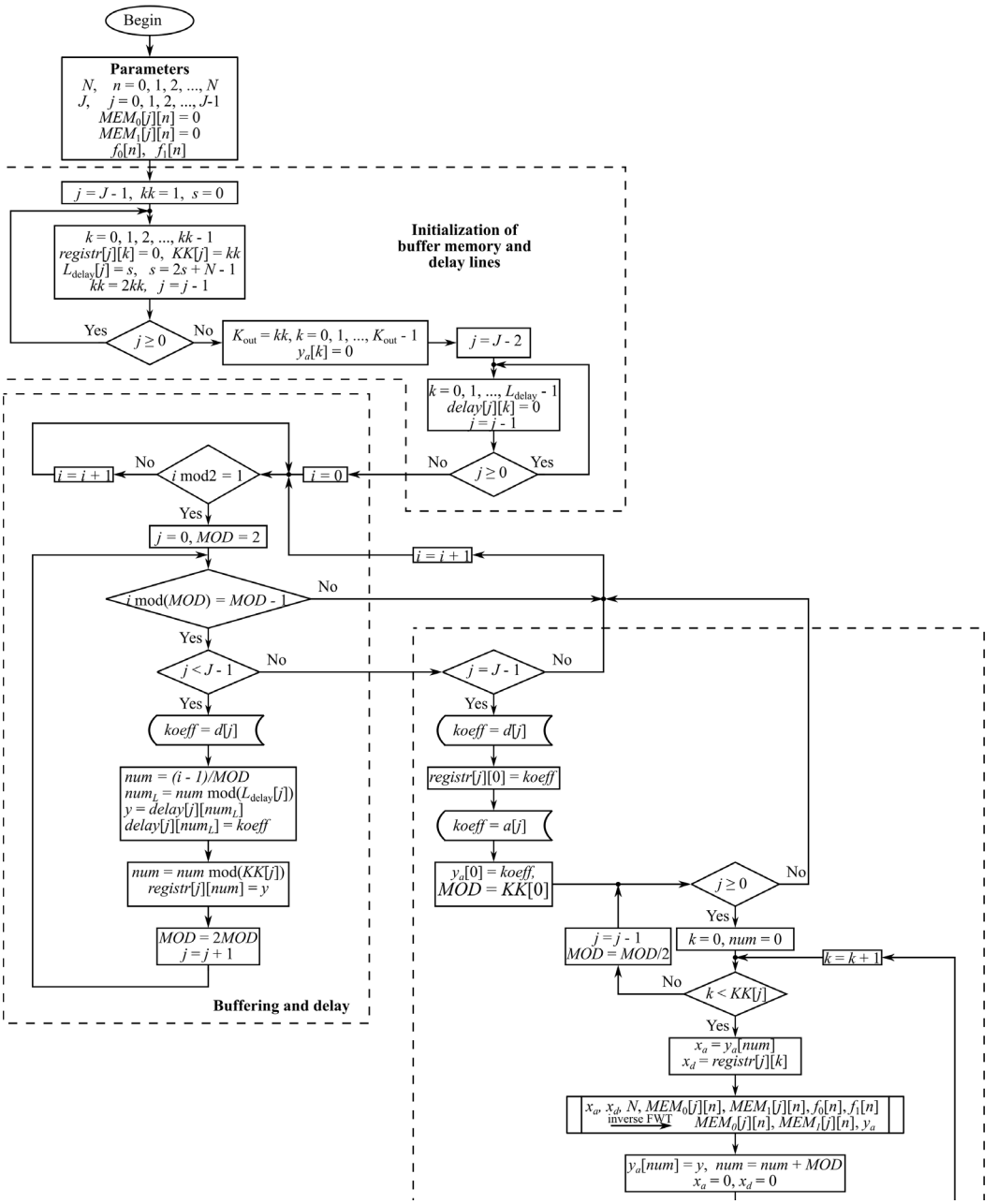
$J = 20$, this delay line length is also increased considerably $L_{\text{delay}} = 4194300$. Every delay requires reading and writing operation, so in total it is necessary to perform L_{delay} reading/writing operations.

At the same time, the only important elements of this delay line are input coefficient and output coefficient. The rest of coefficients are not necessarily required immediately. It would be much more efficient to read required coefficient directly from buffer memory and write in its place a new coefficient value. The volume of required buffer memory is unaffected in this case – it is still $L_{\text{delay}}\left[j\right]$ elements for scale j.

The coefficients are processed as follows:

- The index of coefficient is being determined as $num=\left(i-1\right)/MOD$;
- This index value $num_L = num$ is increased after every detail coefficient reading operation by one. In order to set this value to zero after the maximum index $num_L = L_{\text{delay}}\left[j\right]-1$ has been

Figure 8. *Real-time inverse multiscale discrete wavelet transform algorithm with filter banks over J scales*



reached at scale j it is possible to use remainder from the division of index num with delay line length $L_{\text{delay}}\left[j\right]$ instead of pure index value.

- Thus the element of array $y=delay\left[j\right]\left[num_L\right]$ with index num_L is being read and instantly overwritten with: $delay\left[j\right]\left[num_L\right]=koeff$. After $L_{\text{delay}}\left[j\right]$ of such operations, the index num_L will point at this element again. Thus, it will be read with delay equal to the length of delay line $L_{\text{delay}}\left[j\right]$.

Such approach makes it possible to reduce the number of reading and writing operation from $L_{\text{delay}}\left[j\right]$ to single operation with additional remainder calculation. This number of operations is minimal and unaffected neither by order of filters used in filter banks, nor by number of scale and thus the length of delay line $L_{\text{delay}}\left[j\right]$.

EXPERIMENTAL MEASUREMENTS OF PROCESSINGTIME ON RASPBERRY PI

The Raspberry Pi microprocessor system has been chosen as low cost Linux OS based microprocessor system. Both algorithms (for direct and inverse transforms) have been implemented in C++ programming language for this microprocessor system.

The main topics of interest for this experiment are:

1. How fast real-time multiscale discrete wavelet transform is being calculated?
2. How choice of wavelets affects performance?
3. How number of analysed scales affects performance?

The measurements have been made for three Daubechies wavelets: Daubechies-2, Daubechies-3 and Daubechies-4 wavelets. The numerical notation describes the number of vanishing moments of these wavelets. The orders of filters used in filter banks are, accordingly, $N_2=3$, $N_3=5$ and $N_4=7$. Each filter bank has been estimated for following numbers of scales J: 5, 10, 15, 20. The total number of data samples is $K=2^{21}$ and the samples themselves are represented by their indexes in floating point format number. The measurements results are shown in Table 1 for direct wavelet transform and in Table 2 for inverse wavelet transform.

In order to estimate the time of single transform it is necessary to determine the total number of transforms. Since downsampling operation reduce the number of filtered samples by 2 times at every scale, it is possible to calculate the total number of transforms for scale j as $M_J=K/2^{j-1}$. The total number of transforms up to scale J in such case is a sum:

$$M_J = K\sum_{j=0}^{J-1} 1/2^j, \tag{13}$$

where $K=2^{21}$ is number of samples.

Then it is possible to estimate the processing time of single scale wavelet transform as $T_{\min}=T/M_J$, where T is total processing time. For any other scale processing time is to be multiplied by number of scales up to maximum scale J with processing time $T_{\max}=JT_{\min}$.

For inverse discrete wavelet transform there are two measurements – with delay line and without. The second option, obviously, leads to incorrect signal reconstruction but the main interest here is to estimate processing time for delay line operations, which can be determined as a difference of these two measurements. Then, the processing time of single inverse wavelet transform iteration can be estimated as:

$$T_{\text{iter}} = \frac{\left(T - T_{\text{delay}}\right) K}{2^{J}}, \tag{13}$$

where $K = 2^{21}$ is total number of signal samples. The iteration of wavelet transform here processes entire segment of coefficients with length of 2^{J} at lowest scale $j = 1$.

CONCLUSION

The processing time of single discrete wavelet transform iteration $T_{\min}$ at the scale $j = 1$ is unaffected by number of scales. For any higher scale $j > 1$ the processing time increases proportionally up to $jT_{\min}$.

The length of filter impulse response, which is determined by chosen wavelet, changes processing time proportionally both for direct and inverse wavelet transform.

The iteration time of inverse discrete wavelet transform depends greatly on number of scales J. As such, for high numbers of analysed scales it is impossible to achieve real-time reconstruction of signal, since it is necessary to accumulate a large set of data and process it all at once.

The total processing time of inverse wavelet transform, however is proportionally greater, than total processing time of direct wavelet transform. There are two reasons, that impact this increase the

Table 1. Real-time multiscale discrete wavelet transform processing time for various numbers of scales

Number of scales	Processing time		
	Daubechies-2	Daubechies-3	Daubechies-4
$J = 5$	Total: 3.54s	Total: 4.73s	Total: 5.82s
	Min: 1.74μs	Min: 2.33μs	Min: 2.86μs
	Max: 8.71μs	Max: 11.65μs	Max: 14.32μs
$J = 10$	Total: 3.64s	Total: 4.9s	Total: 6.02s
	Min: 1.74μs	Min: 2.34μs	Min: 2.87μs
	Max: 17.36μs	Max: 23.4μs	Max: 28.74μs
$J = 15$	Total: 3.65s	Total: 4.93s	Total: 6.01s
	Min: 1.74μs	Min: 2.35μs	Min: 2.87μs
	Max: 26.1μs	Max: 35.26μs	Max: 43μs
$J = 20$	Total: 3.66s	Total: 4.92s	Total: 6.02s
	Min: 1.74μs	Min: 2.35μs	Min: 2.87μs
	Max: 34.89μs	Max: 46.94μs	Max: 57.41μs

Table 2. Real-time multiscale inverse discrete wavelet transform processing time for various numbers of scales

Number of scales	Processing time		
	Daubechies-2	**Daubechies-3**	**Daubechies-4**
$J = 5$	Total: 3.54s	Total: 4.73s	Total: 5.82s
	Min: 1.74μs	Min: 2.33μs	Min: 2.86μs
	Max: 8.71μs	Max: 11.65μs	Max: 14.32μs
$J = 10$	Total: 3.64s	Total: 4.9s	Total: 6.02s
	Min: 1.74μs	Min: 2.34μs	Min: 2.87μs
	Max: 17.36μs	Max: 23.4μs	Max: 28.74μs
$J = 15$	Total: 3.65s	Total: 4.93s	Total: 6.01s
	Min: 1.74μs	Min: 2.35μs	Min: 2.87μs
	Max: 26.1μs	Max: 35.26μs	Max: 43μs
$J = 20$	Total: 3.66s	Total: 4.92s	Total: 6.02s
	Min: 1.74μs	Min: 2.35μs	Min: 2.87μs
	Max: 34.89μs	Max: 46.94μs	Max: 57.41μs

most. Firstly, accumulation and delay operations, which can't be avoided for inverse discrete wavelet transform. As these experiments show, the total time spent for delay operations does not depend on any parameters, such as filter impulse response length or number of scales. It is affected only by total number of processed samples.

The second source of additional computations is upsampling operation. In analysis filter bank convolution has been calculated only for odd-numbered samples. For synthesis filter bank the number of calculated convolutions is doubled. It is possible to reduce number of operations in both filter banks by implementing polyphase filter banks. Another option is to create lattice filters, which also divide signal in two phases and process them with reduced number of operations.

REFERENCES

Bahoura, M., & Ezzaidi, H. (2010). Real-time implementation of discrete wavelet transform on FPGA. *Proceedings of the 2010 IEEE 10th International Conference on Signal Processing* (pp.191-193). IEEE Press Piscataway. doi:10.1109/ICOSP.2010.5655177

Ben Hnia Gazzah, I., Souani, C., & Besbes, K. (2008). DSP implementation and performances evaluation of 1D and 2D DWT using the lifting scheme. *Proceedings of the Design and Test Workshop* (pp. 166-172).

Bomers, F. (2000). *Wavelets in real time digital audio processing: Analysis and sample implementations* [Unpublished master's thesis]. University of Mannheim.

Cavuslu, M. A., & Karakaya, F. (2010). Hardware implementation of Discrete Wavelet Transform and Inverse Discrete Wavelet Transform on FPGA. *Proceedings of the 2010 IEEE 18th Signal Processing and Communications Applications Conference (SIU)* (pp.141-144). IEEE Press Piscataway. doi:10.1109/SIU.2010.5653126

Cutajar, M., Gatt, E., Grech, I., Casha, O., & Micallef, J. (2014). Design of a hardware-based discrete wavelet transform architecture for phoneme recognition. *Proceedings of the 2014 6th International Symposium on Communications, Control and Signal Processing (ISCCSP)* (pp.554-557). doi:10.1109/ISCCSP.2014.6877935

Darji, A., Shukla, S., Merchant, S. N., & Chandorkar, A. N. (2014). Hardware Efficient VLSI Architecture for 3-D Discrete Wavelet Transform. *Proceedings of the 2014 27th International Conference on VLSI Design and the 2014 13th International Conference on Embedded Systems* (pp.348-352).

Darji, A. D. Shashikanth, Konale, Limaye, Ankur, Merchant, S.N. & Chandorkar, A.N. (2014). Flipping-based high speed VLSI architecture for 2-D lifting DWT. *Proceedings of the 2014 IEEE 57th International Midwest Symposium on Circuits and Systems (MWSCAS)* (pp.193-196).

Daubechies, I., & Sweldens, W. (1998). Factoring Wavelet Transforms into Lifting Steps. *Fourier Analysis and Applications*, *4*(3), 247–269. doi:10.1007/BF02476026

Jing, C., & Bin, H.Y. (2007). Efficient Wavelet Transform on FPGA Using Advanced Distributed Arithmetic. *Proceedings of the 8th International Conference on Electronic Measurement and Instruments ICEMI '07* (pp. 2-515 – 5-515). doi:10.1109/ICEMI.2007.4350730

Mallat, S. (1999). *A wavelet tour of signal processing* (2nd ed.). Academic press.

O'Brien, A., & Conway, R. (2008). Lifting scheme discrete Wavelet Transform using Vertical and Crosswise multipliers. Proceedings of the *Signals and Systems Conference ISSC '08* (pp.331-336). IET Irish.

Prusa, Z., & Rajmic, P. (2011). *Real-Time lifting wavelet transform algorithm* (Vol. 2, pp. 53–59). International Society for Science and Engineering.

QiWei Lin. Zhenhui Liu & Gui Feng (2009). DWT based on watermarking algorithm and its implementing with DSP. *Proceedings of the 3rd International Conference on Anti-counterfeiting, Security, and Identification in Communication ASID '09* (pp. 131-134).

Sardar, S., & Babu, K. A. (2014). Hardware Implementation of Real-Time, High Performance, RCE-NN Based Face Recognition System. *Proceedings of the 2014 27th International Conference on VLSI Design and the 2014 13th International Conference on Embedded Systems* (pp. 174-179).

Sripath, D. (2003). *Efficient Implementations of Discrete Wavelet Transforms Using FPGAs* [Unpublished master's thesis]. The Florida State University.

Strang, G., & Ngueyen, T. (1996). *Wavelets and filter banks*. Wellesley: Wellesley-Cambridge Press.

Wang, T.-H., Huang, P.-T., Chen, K.-N., Chiou, J.-C., Chen, K.-H., Chiu, C.-T., Tong, H.-M., Chuang, C.-T., & Hwang, W. (2014). Energy-efficient configurable discrete wavelet transform for neural sensing applications. *Proceedings of the 2014 IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 1841-1844).

Wenbing, F., & Yingmin, G. (2008). FPGA Design of Fast Lifting Wavelet Transform. Proceedings of the Congress on Image and Signal Processing CISP '08 (Vol. 4, pp. 362-365).

Wilburn, V. C., & Alexander, W. E. (1994). A parallel implementation of the discrete wavelet transform. *Proceedings of the 26th Southeastern Symposium, System Theory* (pp. 260-264). doi:10.1109/SSST.1994.287872

Nikolajs Bogdanovs is researcher in Transport Electronics and Telematics at the Faculty of Electronics and Communications of the Riga Technical University, Latvia. He obtained his PhD degree in Computer Control, Information and Electronics Systems of Transport. His main research interests are related to sensors, sensor networks, network traffic control and processing, digital signal processing and microcontroller programming.

Elans Grabs is researcher in Transport Electronics and Telematics at the Faculty of Electronics and Communications of the Riga Technical University, Latvia. He obtained his Master's Transport Electronics and Telematics, and is presently completing his PhD degree in Computer Control, Information and Electronics Systems of Transport. His main research interests are related to network traffic control and processing, digital signal processing and microcontroller programming, wireless communication systems, digital systems and filtering.

Ernests Petersons is chief researcher and professor in Transport Electronics and Telematics at the Faculty of Electronics and Communications of the Riga Technical University, Latvia. He obtained his Habilitated Doctor's degree in Electronics and Computer Science. His main research interests are related to computers & network performance evaluation, fault-tolerant computer structures, information technology in aviation. Lately he has become interested in application of Catastrophe Theory for understanding complex computer systems instability.