We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

5

Computational Models Designed in MATLAB to Improve Parameters and Cost of Modern Chips

Peter Malík
Institute of Informatics, Slovak Academy of Sciences
Slovak Republic

1. Introduction

Methods and techniques to design integrated circuits made a huge progress since German engineer Werner Jacobi filed a first patent for an integrated-circuit-like semiconductor amplifying device in 1949 or since Jack Kilby successfully demonstrated the first working integrated circuit on September 12, 1958. The first integrated circuits were composed of a small number of transistors and their functionality was very limited. They were designed manually without any effective calculation tools and therefore the developing phase took long time. Each design error was difficult to discover and the whole process had to be repeated when uncovered during physical prototype testing. This further increased time necessary to develop a working chip. An important milestone in integrated circuits design is the first microprocessor Intel 4004 produced in April 1971. The significance lies in the beginning of mutual bond between produced microprocessors and methods to design them. It was discovered very soon that the high computation power of microprocessors can be utilized directly by the design process to improve future microprocessors. High performance microprocessors started to be used for calculations of an estimated behavior of future prototypes and their parameters. The simple models were transformed to more complex models that closer represented the reality. More precise models increased a chance to uncover a design error in early design phases and accelerated its correction. The continual upgrading process of design methods and tools matures into computer aided design (CAD) software platforms that are able to carry out many tasks automatically which had to be done manually in the past.

Constant refinement of design methods, tools and technology secured the steady grow of integrated circuits internal complexity. Intel co-founder Gordon E. Moore described the trend in his 1965 paper that the number of components in integrated circuits had doubled every year from the invention of the integrated circuit in 1958 until 1965. He predicted that the trend would continue for at least ten years Moore (1965). His prediction has proved to be uncannily accurate and is now used in semiconductor industry to guide long-term planning and to set targets for research and development ITRS (2009). This long-term trend is also known as Moore's law.

The digital design process uses highly automated CAD tools that significantly reduce the time necessary to design a prototype when the design is closely specified by full set of parameters. The original specification is usually general and many parameters have to be defined later. The process specifying missing parameters is an inseparable part of feasibility study or the first design phases. It is very important to set suitable parameters because they determine the resulting prototype. More complex integrated circuits require more parameters to be defined

and therefore more analysis and simulation work is necessary. Usually this process requires more time in comparison to the physical design that is mostly done by modern CAD tools.
The modern CAD tools include models that closely represent the resulting prototype; however, their calculation is too much time consuming. The feasibility study and first design stages require different models. The models have to be relatively simple with the acceptable correlation to reality. An excellent software environment to create and evaluate these models is MATLAB.
Users can create new functions and procedures in MATLAB similarly as in a common programing language environment. An advantage is that MATLAB has a more simple and intuitive syntax and thorough help with many practical examples. Additionally, it has an extensive database of internal functions that can be used directly in the source code of new functions. Models usually describe a simulated behavior by mathematical equations or some iterative process. Both types of model description can be written in MATLAB intuitively with the short source code. However, the biggest advantage is very fast and simple work with calculated results and the ability to visualize the resulted data with internal MATLAB functions.
A hardware implementation of an algorithm or computation scheme implies new challenges that have to be overcome. The general aim is to design an integrated circuit with minimal area that produces results with a pre-specified overall precision. Deep analysis and lot of simulations are needed to attain this. MATLAB can be used as a part of this optimization process. The advantages are extensive mathematical support with the option to create own optimization code and easy way to sort and evaluate excessive calculated results. A simple software computational model can be created in MATLAB and later used within optimization routines. An example of this optimization process is presented in the following sections with an integrated circuit for acceleration of time consuming computations of forward and backward modified discrete cosine transform (MDCT) that is used in audio compression and coding. MDCT is used in many audio coding standards for time-to-frequency transformation of digital signals. It is perfect reconstruction cosine-modulated filter bank based on the concept of time domain aliasing cancellation Princen et al. (1987). MDCT is the basic processing block for high-quality audio compression in the international audio coding standards and commercial audio compression algorithms in consumers electronics Bosi & Goldberg (2003); Spanias et al. (2007). Forward and backward MDCT computations are the most computationally intensive operations in the well-known audio coding standards. Thus, an efficient implementation of the MDCT has become the key technology to realize real-time and low-cost audio decoders.

2. Computational models in digital design process

A computational model is a set of equations that describe some selected attribute or feature in a more simplified way compared to its true physical nature. The idea behind creating a computational model is to be able to run complex analyses and simulations of specific scenarios on PC. The fast computational speed and the ability to calculate with a wide spectrum of input data variations is very important. A computation power of common PC has been increased significantly in recent years, which enabled to run usual simulations on ordinary PC that required big server clusters in the past. The ability to create a computational model easily and in short time accelerates the design process in general. The result is that a designer can verify his/her designs in early developing phases and use the saved time to further improvements.

Digital circuit design is composed of many steps that continually shape the final integrated circuit. The technology advancement paved the road to very complex integrated circuits called system on a chip (SoC). The more complex the chip is the more design steps it requires. Considerable interest has been oriented toward computational models with lower abstraction levels. This resulted in the development of precise computational models that are integrated in modern professional design tools. Easy access to these models is a great advantage. They are utilized mostly during moderate and finalizing design steps. This chapter is oriented to computational models on higher abstraction levels that are used in feasibility studies and first design steps.

At the beginning of design process, the computational models depend considerably on a selection of design characteristics. In this design phase, the designers create more detailed specifications and develop the guidelines to reach them. The original specification is usually too general and its parametric goals can be reached by different means. Each selected solution has to be analyzed and evaluated. Simple computational models are very important in this process. These models can differ from each other significantly. They use limited amount of parameters and many unnecessary features are omitted. Models are used in many situations, e.g., a comparison analysis of hardware implementation with software solution, analysis of different algorithms, evaluation of different hardware architectures or components. These are few examples with the most significant influence on the following design steps.

Each hardware implementation performs one or several algorithms. An algorithm is evaluated by a number of operations that has to be carried out to produce a set of output data. Long computation process can produce a significant computation error that distorts the results. Each algorithm has different sensitivity to accumulation of errors. Therefore, an analysis of computation errors by models is very important and the results usually dictate minimal precision requirements of hardware components. These models are usually very simple with minimal utilization of hardware parameters. In general, all operations are transformed to basic mathematical operations that the selected hardware is capable to perform. Each transformed operation uses rounding or truncating operation to model limited precision of hardware components.

Mathematical operations can be performed with data in different data representations. Selecting an appropriate data representation can improve the resulted parameters. This effect can be modeled and analyzed. Data can be represented by floating point or fixed point. Fixed point representations are also called integer representations. Floating point has better precision for wide data intervals but if data variations are small an integer is a better solution Inacio & Ombres (1996). An integer does not contain information about decimal point placement and therefore it has to be managed externally. Floating point computation components are more complex and thus larger area of a final chip is taken Kwon et al. (2005). The increased area means higher cost of integrated circuits; hence, integer components utilization can reduce the price of a final chip. These are universal guidelines but the optimal results can be achieved only by deep analyses.

Models that analyze data variations are the good starting point to decide between floating point and integer. The level of specialization is the key information in the decision process. More universal applications usually use floating point due to more difficult estimation of data variations. In highly specialized applications integer can be a better solution because internal data structure can be highly optimized. The optimization results depend considerably on the quality of used models and on the optimization extent.

Integer is more susceptible to data overflow than floating point Beauchamp et al. (2008). Data overflow significantly distorts actual values and therefore digital designs have to include some form of protection mechanism. Before it can be designed, the knowledge where and

when the data overflow happens is necessary. This is usually analyzed with simple models that work with wide data intervals and the point of interest is oriented to data that produce highest peak values. It is also important to monitor the highest negative peaks, because they can cause data underflow which has similar effect as data overflow. The data producing these peaks are later used as test vectors for verifications of created designs.

An important part of modern integrated circuit is memory and memory controller. The cost of memory has been falling in last years, which changed the focus from memory size to memory controller bandwidth. The bandwidth depends on the speed of access to memory, on memory delays and on memory interface bit-width. Access to memory can be improved by rearranging of data in the memory. Reading data from memory that are placed next to each other is much faster than reading data that are randomly distributed through the whole memory. Grouping data that are often accessed at the same time and placing them in the shared memory block as close as possible can significantly reduce the access time and increase the bandwidth. Optimal data rearranging can be modeled and analyzed.

Another improvement of the memory bandwidth can be achieved by analysis of data values. If the data values are from a narrow data range, the universal exponent can be transferred separately and following data are represented only with a mantissa. This methodology is mostly used in serial data transfers and with integer representations. In this case, the better bandwidth utilization is increasing complexity of memory controller and therefore it has to be evaluated more thoroughly. A similar principle can be applied to optimization of integer representation itself.

Modern algorithms often require large memory. Lot of data are stored for a short time only. If these time periods are not overlapped, the data can be rewritten and memory size can be reduced. In many cases, the time periods are overlapped only partially and small changes within an order of operations can eliminate overlapping. Of course, this requires more complex models and deeper analysis. The result is represented by a significant reduction of memory size.

Digital designs can utilize different computation flows. A design with few universal computation blocks depends more on a memory subsystem. The advantage is a simple design and support for many different algorithms. The optimization objectives are mostly oriented to a memory subsystem and to the number of universal computational blocks. Deeper analysis can evaluate the necessary complexity level of used computational blocks. An exchange of several computational blocks with simplified blocks can reduce the final area of a chip.

A highly specialized digital design has usually a custom hardware architecture that gone through a long optimization process. Its significant part is done at a higher abstraction level. The important subject of analysis is the effect of increased parallel structures or utilization of computational pipeline. Precision and overflow analysis is more the necessity than an option. Each computational block can use unique interconnection, special features and attached latch or register. The aim is to utilize these options efficiently. Then, the resulting design has excellent parameters. The time consuming optimizations are costly and still economical in large production quantities.

During the computational model design, it is important to select an area of interest and to be able to verify precision of results. A model that produces results that differ from reality too significantly has to be corrected immediately. In many situations, the simple model composed of few mathematical equations can be sufficient, e.g., to verify an algorithm or some independent feature. Mutually dependent features have to be modeled by more robust models. The models that mimic hardware architectures with high precision give designers a chance to analyze the hardware behavior before it is actually produced. This saves time and reduces the overall cost.

The simple model is an advantage when used under optimization routines with many computation loops. Many mutually dependent parameters can easily rise the number of optimization cycles over the acceptable level. In these cases, the very simple models can produce more data on parameters' dependency that is later used to reduce the number of optimization dimensions and widths of input data intervals. This procedure can be repeated later with more robust and more precise models.

3. MDCT computational models in MATLAB

Hardware implementation has special preferences for computational algorithms. The number and complexity of computation operations are most important. An algorithm with smaller number of operations computes results in shorter time which represents improved computational speed. The number of computation operations reduced only by single operation can significantly improve overall performance if the computation is run many times under a loop or in more complex algorithm. Not all operations are equal. Addition and subtraction operations are similar in their complexity. Multiplication operation is much more complex and therefore consumes much more area on a chip. There is an exception. Multiplication by a constant that equals to power of 2 (2, 4, 8, 16, 32, 64, ...) is the same as shifting the original binary number to the left by the number of bits that equals to base 2 logarithm of the constant. It is similar to a constant that equals to negative power of 2 (1/2, 1/4, 1/8, ...) with the difference that shifting is to the right. Hence, one removed multiplication is more significant than a removed addition or subtraction, and multiplication by power of 2 can be neglected.

The algorithm selected for hardware implementation is presented in Britanak & Rao (2002). The algorithm computes MDCT that is used in audio compression. It has low number of mathematical operations. Additionally, it has a symmetric computational structure that is an advantage in digital design. The symmetric structure can be implemented more efficiently. The resulting hardware is dedicated to accelerate the time consuming MDCT computations in audio compression.

Correct verifications during the whole design process save time. Frequent verifications facilitate a process of locating errors, because only few modifications were made between two following inspections. An algorithm verification can be done by another algorithm that is widely accepted and well proven. The best choice for verification of MDCT algorithm is the original MDCT algorithm Princen & Bradley (1986). Its mathematical equations are:

$$c_k = \sum_{n=0}^{N-1} x_n cos\left[\frac{\pi}{2N}\left(2n+1+\frac{N}{2}\right)(2k+1)\right], \quad k = 0,1,\ldots,\frac{N}{2}-1, \tag{1}$$

$$\hat{x}_n = \frac{2}{N}\sum_{k=0}^{\frac{N}{2}-1} c_k cos\left[\frac{\pi}{2N}\left(2n+1+\frac{N}{2}\right)(2k+1)\right], \quad n = 0,1,\ldots,N-1, \tag{2}$$

where $\{c_k\}$ are MDCT coefficients and $\{\hat{x}_n\}$ represents the time domain aliased data sequence recovered by backward MDCT. The verification model was designed straightforward by simple computation loops. The source code is shown in the following MATLAB example.

```
function Z = MDCT_analytic(X, N)
Z = zeros(1, N/2);
for k = 0:((N/2) - 1)
    sum = 0.0;
    for n = 0:(N - 1)
```

```
        arg = pi / (2 * N) * ((2 * n + 1 + N/2) * (2 * k + 1));
        sum = sum + (X(n + 1) * cos(arg));
    end
Z(k + 1) = sum;
```

Complete formulas constituting the selected MDCT algorithm are as follows:

$$
\begin{aligned}
z_{2k} &= (-1)^k \frac{\sqrt{2}}{2} \sum_{n=0}^{\frac{N}{4}-1} \left(a_n cos\left[\frac{\pi(2n+1)k}{2(N/4)}\right] - b_n sin\left[\frac{\pi(2n+1)k}{2(N/4)}\right]\right), \\
z_{2k+\frac{N}{2}} &= (-1)^{k+\frac{N}{4}} \frac{\sqrt{2}}{2} \sum_{n=0}^{\frac{N}{4}-1} (-1)^{n+1} \left(a_n sin\left[\frac{\pi(2n+1)k}{2(N/4)}\right] + b_n cos\left[\frac{\pi(2n+1)k}{2(N/4)}\right]\right), \\
& k = 0, 1, \ldots, \frac{N}{4} - 1,
\end{aligned} \tag{3}
$$

where

$$
\begin{aligned}
a_n &= \left(x_n^{'} - x_{\frac{N}{2}-1-n}^{''}\right) cos\frac{\pi(2n+1)}{2N} - \left(x_n^{''} - x_{\frac{N}{2}-1-n}^{'}\right) sin\frac{\pi(2n+1)}{2N}, \\
b_n &= \left(x_n^{'} - x_{\frac{N}{2}-1-n}^{''}\right) sin\frac{\pi(2n+1)}{2N} + \left(x_n^{''} - x_{\frac{N}{2}-1-n}^{'}\right) cos\frac{\pi(2n+1)}{2N}, \quad n = 0, 1, \ldots, \frac{N}{4} - 1,
\end{aligned} \tag{4}
$$

and

$$
x_n^{'} = x_n - x_{N-1-n}, \quad x_n^{''} = x_n + x_{N-1-n}, \quad n = 0, 1, \ldots, \frac{N}{2} - 1. \tag{5}
$$

Final MDCT coefficients are obtained as

$$
c_{2k} = z_{2k}, \quad c_{2k+1} = -z_{N-2-2k}, \quad k = 0, 1, \ldots, \frac{N}{4} - 1. \tag{6}
$$

The model of selected algorithm is used for verification and optimization of an internal computational structure and it is also used under optimization routines with many loops. Therefore, the model has to be designed more efficiently in comparison to the algorithm verification model. MATLAB can calculate matrix equations more efficiently than equations written in linear code. A model described by matrix equations can reduce computation time significantly. This is very important when the model is a part of optimization process with many loops, because the saved computation time is multiplied by a number of loops.
Equations (5) can be transformed to matrix form

$$
\mathbf{x}^{'} = \mathbf{X}_{N/2\times N}^{'}\mathbf{x}, \quad \mathbf{x}^{''} = \mathbf{X}_{N/2\times N}^{''}\mathbf{x}, \tag{7}
$$

where matrices $\mathbf{X}_{N/2\times N}^{'}$ and $\mathbf{X}_{N/2\times N}^{''}$ are spare matrices that compute addition or subtraction of two input data. They can be merged to single matrix $\mathbf{X1}_{N\times N}$. These matrices model a set of adders and subtracters and can be used as a part of hardware architecture model. They are given by

$$
\mathbf{X}_{N/2\times N}^{'} = \left[\mathbf{I}_{N/2} \; -\mathbf{J}_{N/2}\right], \quad \mathbf{X}_{N/2\times N}^{''} = \left[\mathbf{I}_{N/2} \;\; \mathbf{J}_{N/2}\right], \quad \mathbf{X1}_{N\times N} = \begin{bmatrix} \mathbf{I}_{N/2} & \mathbf{J}_{N/2} \\ \mathbf{J}_{N/2} & -\mathbf{I}_{N/2} \end{bmatrix}, \tag{8}
$$

where $\mathbf{I}_{N/2}$ is the identity matrix, $\mathbf{J}_{N/2}$ is the reflection matrix, all of order $N/2$. The matrix $\mathbf{X1}_{N\times N}$ written in MATLAB is shown in the following example.

```
X1 = [   eye(N/2)     fliplr(eye(N/2));
      fliplr(eye(N/2))     -eye(N/2)  ];
```

Equation (4) can be decomposed into several stand-alone operations that are easily transformed to spare matrices. The first matrix $\mathbf{X2}_{N/2\times N}$ represents subtraction in the brackets of equation (4). The second matrix $\mathbf{G}_{N\times N/2}$ represents multiplication by cosines and sines constants. The third matrix $\mathbf{G2}_{N/2\times N}$ represents the remaining subtraction and addition. The matrix $\mathbf{X2}_{N/2\times N}$ models a set of subtracters, matrix $\mathbf{G}_{N\times N/2}$ models a set of multipliers and $\mathbf{G2}_{N/2\times N}$ models a set of adders and subtracters. They are given by

$$\mathbf{X2}_{N/2\times N} = \begin{bmatrix} \mathbf{0}_{N/4} & -\mathbf{I}_{N/4} & \mathbf{0}_{N/4} & \mathbf{I}_{N/4} \\ \mathbf{I}_{N/4} & \mathbf{0}_{N/4} & -\mathbf{I}_{N/4} & \mathbf{0}_{N/4} \end{bmatrix}, \quad \mathbf{G2}_{N/2\times N} = \begin{bmatrix} \mathbf{I}_{N/4} & \mathbf{0}_{N/4} & -\mathbf{I}_{N/4} & \mathbf{0}_{N/4} \\ \mathbf{0}_{N/4} & \mathbf{I}_{N/4} & \mathbf{0}_{N/4} & \mathbf{I}_{N/4} \end{bmatrix}, \tag{9}$$

$$\mathbf{G}_{N\times N/2} = \begin{bmatrix} cos\frac{(N/2-1)\pi}{2N} & & & & & & & & \\ & \ddots & & & & & & & \\ & & cos\frac{3\pi}{2N} & & & & & & \\ & & & cos\frac{\pi}{2N} & & & & & \\ & & & & cos\frac{\pi}{2N} & & & & \\ & & & & & cos\frac{3\pi}{2N} & & & \\ & & & & & & \ddots & & \\ & & & & & & & cos\frac{(N/2-1)\pi}{2N} & \\ & & & & & & & sin\frac{(N/2-1)\pi}{2N} & \\ & & & & & & \iddots & & \\ & & & & & sin\frac{3\pi}{2N} & & & \\ & & & & sin\frac{\pi}{2N} & & & & \\ & & & sin\frac{\pi}{2N} & & & & & \\ & & sin\frac{3\pi}{2N} & & & & & & \\ & \iddots & & & & & & & \\ sin\frac{(N/2-1)\pi}{2N} & & & & & & & & \end{bmatrix}, \tag{10}$$

where $\mathbf{0}_{N/4}$ is the zero matrix, $\mathbf{I}_{N/4}$ is the identity matrix, $\mathbf{J}_{N/4}$ is the reflection matrix, all of order $N/4$. The matrices $\mathbf{X2}_{N/2\times N}$, $\mathbf{G}_{N\times N/2}$ and $\mathbf{G2}_{N/2\times N}$ written in MATLAB are shown in the following example.

```
X2 = [zeros(N/4) -eye(N/4)  zeros(N/4)   eye(N/4);
       eye(N/4)  zeros(N/4)  -eye(N/4)  zeros(N/4)];
G = [    diag(cos(((N/2)-(1:2:N/2))*pi)/(2*N)))      zeros(N/4);
     zeros(N/4)    diag(fliplr(cos(((N/2)-(1:2:N/2))*pi)/(2*N))));
     zeros(N/4)    fliplr(diag(sin(((N/2)-(1:2:N/2))*pi)/(2*N))));
    flipud(diag(sin(((N/2)-(1:2:N/2))*pi)/(2*N))))   zeros(N/4) ];
G2 = [ eye(N/4)  zeros(N/4) -eye(N/4)  zeros(N/4);
      zeros(N/4)  eye(N/4)  zeros(N/4)  eye(N/4) ];
```

Equation (6) represents permutation operations that change signs and order of data. It can be transformed to the matrix form. The source code of the resulting matrix $\mathbf{Q}_{N/2\times N/2}$ is shown in the following MATLAB example.

```
function Q = permatrix(N);
Q = zeros(N/2);
for k = 1:(N/4)
Q(2*k-1, k) = 1;
Q(2*k, N/2-k+1) = -1;
end
```

The most complex part of the selected algorithm is equation (3). It represents N/4-point DCT-II and N/4-point DST-II combined by a butterfly operation Britanak & Rao (2002). DCT-II and DST-II can be represented by a matrix but it cannot be generated universally for any N/4 due to their not symmetrical internal computational structure. The MDCT computational model is optimized for audio standard MP3 that uses 36-point MDCT for slowly changing audio signal and 12-point MDCT for rapidly changing audio signal. Hence, the model requires 9-point DCT-II/DST-II and 3-point DCT-II/DST-II matrices. The matrices DCT-II and DST-II are similar and it is possible to transform one to the other with simple modifications. The procedure to transforming DST-II to DCT-II is described in Britanak (2002). The transformation is composed of extra operations that invert each odd input data and reverse order of all input data. It results in one DCT-II that is used two times with different sets of data. This is advantageous when dedicated hardware architecture is to be designed. The single hardware block can be optimized within shorter time and overall hardware architecture can be reduced easily by multiplexing input data from two datasets to single hardware block. The 9-point DCT-II matrix can be derived directly from the mathematic equation. The result is a matrix that represents many operations of addition and multiplication. The number of these operations can be reduced significantly by matrix decomposition to several spare matrices or by optimized linear code. The both principles utilize computation with partial results that influence several output data. It is more efficient to use already computed partial results during the following computations than to calculate all results from input data only. The 9-point DCT-II matrix $\mathbf{DC}_{9\times 9}$ written in MATLAB is shown in the following example.

```
u = 2 * pi / 9;
d1 = sqrt(3)/2;
d2 = 0.5;
d3 = cos(4*u);
d4 = cos(2*u);
d5 = cos(u);
d6 = sin(4*u);
d7 = sin(2*u);
d8 = sin(u);
DC = [ 1      1    1        1     1     1       1      1      1;
       d7     d1   d8     (d7-d8) 0   (d8-d7)  -d8    -d1    -d7;
      -d3     d2 (d3+d5)   -d5   -1    -d5    (d3+d5)  d2    -d3;
       d1     0   -d1      -d1    0     d1      d1     0     -d1;
   -(d3+d4) -d2   d3        d4    1     d4      d3    -d2  -(d3+d4);
    (d7-d6) -d1  -d6        d7    0    -d7     d6      d1    (d6-d7);
      d2    -1    d2        d2   -1     d2      d2    -1      d2;
      d6    -d1 (d6+d8)    -d8    0     d8   -(d6+d8) d1     -d6;
```

```
      d4    -d2    d5    -(d4+d5) 1  -(d4+d5)   d5   -d2      d4];
```

It can be seen from the 9-point DCT-II example that the matrix $\mathbf{DC}_{9\times9}$ is not a spare matrix. It is composed of many nonzero constants. Most of them are not equal to power of 2; hence, shift operations are used rarely. When the computation of this matrix is carried out directly, the number of multiplications is equal to the number of matrix elements that are not equal to ±1 or 0. The number of multiplications is equal to 60. 12 of them are multiplication by 0.5 and therefore the number of nontrivial multiplication is 48.

Many scientific papers oriented to reduction of mathematical operations within specific algorithms were published. The research in this area of interest is highly valued. The paper Britanak & Rao (2001) presents linear code that requires 10 multiplications for a computation of 9-point DCT-II and 2 of them are multiplication by 0.5. The optimized linear code needs 6-times less multiplications. This code can be transformed to a set of spare matrices that calculate results with minimized number of mathematical operations when multiplied in correct order. One version of these spare matrices is shown in the following MATLAB example.

```
DC1 = [  0  0  0  1  0  1  0  0  0;
         0  0  0  1  0 -1  0  0  0;
         0  0  1  0  0  0  1  0  0;
         0  0 -1  0  0  0  1  0  0;
         0  1  0  0  0  0  0  1  0;
         0  1  0  0  0  0  0 -1  0;
         1  0  0  0  0  0  0  0  1;
        -1  0  0  0  0  0  0  0  1;
         0  0  0  0  1  0  0  0  0];

DC2 = [  0   0   0   0   1   0   0   0   1;
         1   0   1   0   0   0   0   0   0;
         0   0   1   0   0   0  -1   0   0;
         1   0   0   0   0   0  -1   0   0;
         1   0  -1   0   0   0   0   0   0;
         0   1   0  -1   0   0   0   0   0;
         0   0   0   1   0   0   0   1   0;
         0   1   0   0   0   0   0  -1   0;
         0   1   0   1   0   0   0   0   0;
         0   0   0   0   0  -d1  0   0   0;
         0   0   0   0   d2  0   0   0   0;
         0   0   0   0   0   0   1   0   0;
         0   0   0   0   0   0   0   1   0;
         0   0   0   0   0   0   0   0   1];

DC3 = [  0  1  0  0  0  0  0  0  0  0  0  1  0  0;
         0  0  0  0  0  1  0  0  0  0  0  0  1  0;
         0  0 -d3 0  0  0  0  0  0  0  0  0  0  0;
         0  0  0 -d4 0  0  0  0  0  0  0  0  0  0;
         0  0  0  0 -d5 0  0  0  0  0  0  0  0  0;
         0  0  0  0  0  0 -d6 0  0  0  0  0  0  0;
         0  0  0  0  0  0  0 -d7 0  0  0  0  0  0;
         0  0  0  0  0  0  0  0 -d8 0  0  0  0  0;
```

```
          0  0  0  0  0  0  0  0  0  0 -1  0  0  1;
          1  0  0  0  0  0  0  0  0  0  0  0  0  0;
          0  0  0  0  0  0  0  0  0  1  0  0  0  0];

DC4 = [ d2 0  0  0  0  0  0  0  0  0  0;
        0 -d1 0  0  0  0  0  0  0  0  0;
        0  0  1  0  0  0  0  0  1  0  0;
        0  0 -1  0  0  0  0  0  1  0  0;
        0  0  0  1  0  0  0  0  1  0  0;
        0  0  0  0  0  1  0  0  0  0  1;
        0  0  0  0  0 -1  0  0  0  0  1;
        0  0  0  0  0  0  1  0  0  0  1;
        1  0  0  0  0  0  0  0  0  1  0;
        0  0  0  0  0  0  0  0  0  1  0;
        0  0  0  1  0  0  0  0  0  0  0;
        0  0  0  0  1  0  0  0  0  0  0;
        0  0  0  0  0  0  1  0  0  0  0;
        0  0  0  0  0  0  0  1  0  0  0];

DC5 = [ 0  0  0  0  0  0  0  0  1  0  0  0  0  0;
        0  0  0  0  0  0  0 -1  0  0  0  0  0  1;
        0  0 -1  0  0  0  0  0  0  0  0  1  0  0;
        0  1  0  0  0  0  0  0  0  0  0  0  0  0;
        0  0  0  1  0  0  0  0  0  0 -1  0  0  0;
        0  0  0  0  0  0  1  0  0  0  0  0 -1  0;
        1  0  0  0  0  0  0  0  0 -1  0  0  0  0;
        0  0  0  0  0  1  0  0  0  0  0  0  0  1;
        0  0  0  0  1  0  0  0  0  0  0  1  0  0];
```

The matrices $\mathbf{DC1}_{9\times9}$, $\mathbf{DC2}_{14\times9}$, $\mathbf{DC3}_{11\times14}$, $\mathbf{DC4}_{14\times11}$ and $\mathbf{DC5}_{9\times14}$ use the same constants as the matrix $\mathbf{DC}_{9\times9}$ from the previous example. They are designed to model hardware architecture. Addition or subtraction operations have only two inputs which is characteristic for hardware implementations. Mutual addition of more input data is distributed through several matrices. The last few lines of the several matrices perform no mathematical operation. These lines provide access to input data or partial results for the following matrices.

Computation of 9-point DST-II with 9-point DCT-II requires two small modifications. Inversion of odd input data can be modeled by a modified identity matrix that has all odd diagonal elements equal to -1. Reverse order of input data can be modeled by a reflection matrix. These two matrices are used together with $\mathbf{DC1}_{9\times9}$ during computation of 9-point DST-II. The computations of 9-point DCT-II and 9-point DST-II are done independently with clearly separated input data and results. The model with both 9-point DCT-II and 9-point DST-II is shown in the following MATLAB example.

```
M1 = [ fliplr(eye(9)), zeros(9);
          zeros(9),     eye(9)];
M2 = diag([ones(1,9), repmat([1 -1],1,4), 1 ]);
DCC1 = [ DC1    zeros(9);
        zeros(9)  DC1   ];
DCS1 = DCC1*M2*M1;
DCS2 = [     DC2    zeros(14,9);
```

```
        zeros(14,9)      DC2    ];
DCS3 = [     DC3    zeros(11,14);
        zeros(11,14)     DC3    ];
DCS4 = [     DC4    zeros(14,11);
        zeros(14,11)     DC4    ];
DCS5 = [     DC5    zeros(9,14);
        zeros(9,14)     DC5     ];
```

The results from DCT-II and DST-II are combined by a butterfly operation. It is a symmetrical operation and therefore can be described universally for any $N/4$ by a matrix $\mathbf{D}_{N/2\times N/2}$. The matrix written in MATLAB is shown in the following example.

```
D = [       1,                          zeros(1,N/2-1);
    zeros(N/4-1,1)  eye(N/4-1)  zeros(N/4-1,1) -fliplr(eye(N/4-1));
                     zeros(1,N/4)            -1     zeros(1,N/4-1);
    zeros(N/4-1,1) -fliplr(eye(N/4-1)) zeros(N/4-1,1) -eye(N/4-1)];
```

The presented matrices are the main elements of the MDCT model. The model computation core is composed of multiplications of these matrices in a correct order. The MDCT model is shown in the following MATLAB example.

```
function C = MDCT36(x);
N = 36;
nc = sqrt(2)/2;
u = 2 * pi / 9;
d1 = sqrt(3)/2;
d2 = 0.5;
d3 = cos(4*u);
d4 = cos(2*u);
d5 = cos(u);
d6 = sin(4*u);
d7 = sin(2*u);
d8 = sin(u);
C = nc*Q*D*DCS5*DCS4*DCS3*DCS2*DCS1*G2*G*X2*X1*x;
```

All matrices used in the model are spare matrices with low number of mathematical operations which increases computational speed in MATLAB. They also correspond to basic hardware computational blocks as an adder, subtracter or multiplier. Further optimization of these matrices also improves the resulting hardware architecture.
The MDCT computational model is created. The next step is analysis of data variations. Its results represent actual intervals of data inside the modeled computational structure. The positive and negative extreme input values can cause data overflow or data underflow and therefore it is important to understand how the extreme values are propagated through the computational structure. This knowledge facilitates a selection of appropriate data representations and is necessary when integer representations are selected. Extreme values of partial results can be paired with input data that caused them. These input data can be included into a verification data set and used later to test if overflowing or underflowing can happen. A computational model described by a matrix equation simplifies the analysis process which is based on solving these mathematical equations. The equation

$$\mathbf{y}^{X1} = \mathbf{X1}_{N\times N}\mathbf{x}, \tag{11}$$

can be transformed to a set of equations

$$\begin{aligned} y_1^{X1} &= \mathbf{X1}(1,:)\mathbf{x} = \mathbf{X1}(1,1)x_1 + \mathbf{X1}(1,2)x_2 + \ldots \mathbf{X1}(1,N)x_N, \\ y_2^{X1} &= \mathbf{X1}(2,:)\mathbf{x} = \mathbf{X1}(2,1)x_1 + \mathbf{X1}(2,2)x_2 + \ldots \mathbf{X1}(2,N)x_N, \\ &\vdots \\ y_N^{X1} &= \mathbf{X1}(N,:)\mathbf{x} = \mathbf{X1}(N,1)x_1 + \mathbf{X1}(N,2)x_2 + \ldots \mathbf{X1}(N,N)x_N, \end{aligned} \tag{12}$$

where $\mathbf{X1}(i,:)$ is the i-th line of the matrix $\mathbf{X1}_{N\times N}$. The positive extreme value of y_i^{X1} is produced by addition of partial results $\mathbf{X1}(i,j)x_i$ with maximal values that are all positive. This means that if $\mathbf{X1}(i,j)$ is positive then x_i should be maximal value within the interval of input data. If it is negative then x_i should be maximal negative value within the interval of input data. Similarly, the negative extreme value of y_i^{X1} is produced by addition of partial results $\mathbf{X1}(i,j)x_i$ with maximal values that are all negative. The matrix $\mathbf{X1}_{N\times N}$ is very simple with only additions and subtractions and therefore it is intuitive that maximal value of results is double the maximal input value. This simple conclusion is valid only at the beginning of computation. In the middle, it is influenced by the internal constants and computational structure.

The computation of extreme values that are produced by other matrices is done by the same principle. The matrix $\mathbf{X1}_{N\times N}$ in equation (11) is substituted by product of the matrix under investigation and other matrices that precede it. This principle is shown in equation (13). The product of necessary matrices can be calculated easily in MATLAB. The whole process can be also automated by new function created in MATLAB.

$$\begin{aligned} \mathbf{y}^{X1} &= \mathbf{MX2} = \mathbf{X2} * \mathbf{X1} * \mathbf{x} \\ \mathbf{y}^{G} &= \mathbf{MG} = \mathbf{G} * \mathbf{X2} * \mathbf{X1} * \mathbf{x} \\ \mathbf{y}^{G2} &= \mathbf{MG2} = \mathbf{G2} * \mathbf{G} * \mathbf{X2} * \mathbf{X1} * \mathbf{x} \\ \mathbf{y}^{DCS1} &= \mathbf{MDCS1} = \mathbf{DCS1} * \mathbf{G2} * \mathbf{G} * \mathbf{X2} * \mathbf{X1} * \mathbf{x} \\ \mathbf{y}^{DCS2} &= \mathbf{MDCS2} = \mathbf{DCS2} * \mathbf{DCS1} * \mathbf{G2} * \mathbf{G} * \mathbf{X2} * \mathbf{X1} * \mathbf{x} \\ \mathbf{y}^{DCS3} &= \mathbf{MDCS3} = \mathbf{DCS3} * \mathbf{DCS2} * \mathbf{DCS1} * \mathbf{G2} * \mathbf{G} * \mathbf{X2} * \mathbf{X1} * \mathbf{x} \\ \mathbf{y}^{DCS4} &= \mathbf{MDCS4} = \mathbf{DCS4} * \mathbf{DCS3} * \mathbf{DCS2} * \mathbf{DCS1} * \mathbf{G2} * \mathbf{G} * \mathbf{X2} * \mathbf{X1} * \mathbf{x} \\ \mathbf{y}^{DCS5} &= \mathbf{MDCS5} = \mathbf{DCS5} * \mathbf{DCS4} * \mathbf{DCS3} * \mathbf{DCS2} * \mathbf{DCS1} * \mathbf{G2} * \mathbf{G} * \mathbf{X2} * \mathbf{X1} * \mathbf{x} \\ \mathbf{y}^{D} &= \mathbf{MD} = \mathbf{D} * \mathbf{DCS5} * \mathbf{DCS4} * \mathbf{DCS3} * \mathbf{DCS2} * \mathbf{DCS1} * \mathbf{G2} * \mathbf{G} * \mathbf{X2} * \mathbf{X1} * \mathbf{x} \\ \mathbf{y}^{Q} &= \mathbf{MQ} = \mathbf{Q} * \mathbf{D} * \mathbf{DCS5} * \mathbf{DCS4} * \mathbf{DCS3} * \mathbf{DCS2} * \mathbf{DCS1} * \mathbf{G2} * \mathbf{G} * \mathbf{X2} * \mathbf{X1} * \mathbf{x} \end{aligned} \tag{13}$$

The next step is optimization of all data representations within the whole computational structure. Integer representations are selected for the MDCT model. The extreme values are known, therefore data overflow is no longer a problem. The aim of the following optimization is to set an optimal ratio between overall computation error, input data precision and output data precision. The MDCT algorithm is composed of many mathematical operations and most of them increase the interval of their results. This means widening of integer interface. An addition usually increases the interface by one bit and a multiplication increases it to double.

The value of the least significant bit (LSB) decreases exponentially with linear increasing of integer interface bit-width. When the LSB value is too small the information that it carries can be discarded without increasing the overall computation error. This can be modeled by introduction of rounding or truncating operations into the model. MATLAB contains internal functions that transform data to binary integer and back. This is shown in the following MATLAB example where b1 is an integer variable and a2 is recovered value of a1 with finite precision set by q.

```
q = quantizer([16,15]);
a1 = 0.54321;
b1 = num2bin(q, a1);
a2 = bin2num(q, b1);
```

The functions used in the previous example take too much time to compute. A faster method to model finite precision of data is the utilization of truncating operations that directly represents discarding of several LSBs. The rounding operation is more complex and requires additional computations. The rounding operation is usually transformed to a combination of addition and truncating operation where the added value is equal to the half of new LSB after rounding. Data in hardware are usually represented in two's-complement arithmetic and discarding of several LSBs in this system represents rounding toward zero. The function that performs rounding toward zero in MATLAB is called **fix()**. The operation of the **fix()** function is confined to the placement of decimal point. The shift of decimal point modifies the amount of data that is going to be discarded. The new data has to be in original form and this implies a second shift of the decimal point in opposite direction. The combination of shift, rounding toward zero and opposite shift can model the truncating operation in hardware. A shift is represented by multiplication of a constant. A constant in the binary shift is equal to power of 2. The positive power represents data shift to the left and negative power stands for data shift to the right. This is shown in the following MATLAB example that is a modification of the MDCT computational model.

```
yX1 = X1*x;
fyX1 = fix(2^s1 * yX1) * 2^(-s1);
yX2 = X2*fyX1;
fyX2 = fix(2^s2 * yX2) * 2^(-s2);
yG = G*fyX2;
fyG = fix(2^s3 * yG) * 2^(-s3);
yG2 = G2*fyG;
fyG2 = fix(2^s4 * yG2) * 2^(-s4);
yDCS1 = DCS1*fyG2;
fyDCS1 = fix(2^s5 * yDCS1) * 2^(-s5);
yDCS2 = DCS2*fyDCS1;
fyDCS2 = fix(2^s6 * yDCS2) * 2^(-s6);
yDCS3 = DCS3*fyDCS2;
fyDCS3 = fix(2^s7 * yDCS3) * 2^(-s7);
yDCS4 = DCS4*fyDCS3;
fyDCS4 = fix(2^s8 * yDCS4) * 2^(-s8);
yDCS5 = DCS5*fyDCS4;
fyDCS5 = fix(2^s9 * yDCS5) * 2^(-s9);
yD = D*fyDCS5;
fyD = fix(2^s10 * yD) * 2^(-s10);
```

```
yQ = Q*fyD;
fyQ = fix(2^s11 * yQ) * 2^(-s11);
C = nc*fyQ;
```

The variables s1, s2, ..., s11 from the previous example represent the precision of partial results. The values of variables are natural numbers and they depend on the input data precision, input data interval and intervals of partial results. They all can be used as inputs of an optimization process where low values of the variables mean low overall computation precision and high values of the variables imply large area of a resulted hardware implementation. Some of them are more important than others. Multiplication doubles the output precision. It means that a multiplier output has two time more bits in comparison to a single multiplier input and therefore the precision of multiplication results has to be modified. Overall computation precision is influenced significantly by precision of internal constants. The MDCT algorithm uses constants three times during the whole computation. The first set of constants is represented by the matrix $\mathbf{G}_{N\times N/2}$. The second set of constants is used during computation of DCT-II and DST-II and the last multiplications by constant represent the normalization of all results by the scaling constant nc. Precision of the internal constants can be optimized together with other variables.
Suitable values of the variables and precision of constants can be set by a simple analysis. Several variables can be set intuitively. One of them is s1 that stands for the precision of matrix $\mathbf{X1}_{N\times N}$. This matrix performs addition and subtraction operations and adding one extra bit is required. The truncating operation at the beginning of computation affects overall computation error more significantly. This truncating operation can be omitted. A similar principle can be done also for matrix $\mathbf{X2}_{N/2\times N}$. The other variables can be set to the output precision that is increased by one or two bits. The effect of higher precision can be easily visualized up to three dimensions in MATLAB. In direct approach, more dimensions are not possible to show clearly. Therefore, it is useful to modify only two variables at a time when the visualization is the main optimization tool. An example of the described visualization is shown in Figures 1 and 2. As can be seen, the multiplication results with bit-width wider than 20 bits does not improve overall precision in the specific situation shown in Figures 1 and 2. The combination of visualizations and intuitive approach usually result in a set of suitable variable and parameter values within short time. However, complex optimization routines are necessary to calculate the best set of variables and parameters.

4. Refinement of computational models and their application

Computational models are used in analyses, simulations and optimization processes. They are the key elements to further improve the parameters of future products. Optimized computational models represent hardware architectures that are later implemented as hardware prototypes. Verification of hardware prototypes and their following modifications are necessary. Hardware prototypes are designed in different platforms; therefore, direct verification is not possible. Verification models have been created to test designed computational models. The mechanism to generate input stimuli and pair them with correct results can be utilized. The necessary modifications have to be made to transform generated data to the specific data representations that are used in hardware prototypes. The models can be upgraded to write the transformed data to an external file that is directly used as input of a verification process. The modification of the final computational model allows to include not only input output pairs but also data that represent partial results of the internal computational structure.

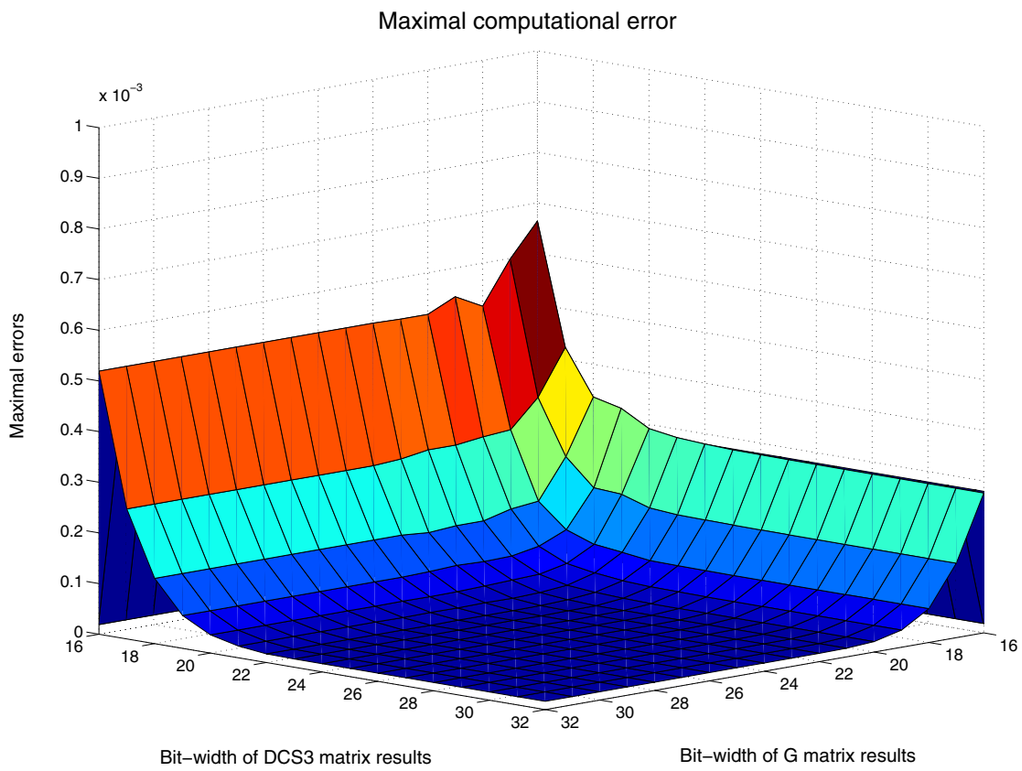

Fig. 1. Maximal computational error dependence on the bit-width of multiplication results.

The most important parameters of the hardware chip are the area, speed or frequency and delay. These parameters depend significantly on implementation technology. They cannot be calculated directly or at least without extremely complex models. Low abstraction level models are capable to compute them with high precision but they are too time demanding. They are used usually only for verification or fine-tuning during the final steps of the design process. The problem is that these parameters are necessary within the optimization process. They are important factors of final product specification. At the same time, they represent feasibility boundaries. The chip with the area larger than few hundreds square millimeters is not possible to produce and it is the same with too high frequency. Many other parameters influence their values, e.g., higher precision stands for larger area. Precision can be easily modeled and accurately calculated. However, this is not true for the area. An optimization process has to be able to evaluate the effect of increased precision to the area. Otherwise, a cost of increased precision cannot be assessed.

Time consuming calculations can be replaced by an estimation. The advantage is in significantly reduced computation complexity. This allows to use an estimation as a part of the optimization process and to optimize other parameters more efficiently. A disadvantage is in limited accuracy. In general, estimation of some complex parameter, e.g., the area, can be implemented into computational models with any abstraction levels. Estimation capabilities of higher abstraction models produce more coarse results. However, the accuracy can be improved by increasing the amount of data that the estimation is based on. The estimation data can be taken from many diverse situations and can be represented by many different forms. The most common procedure is to take data from well known previous situations or

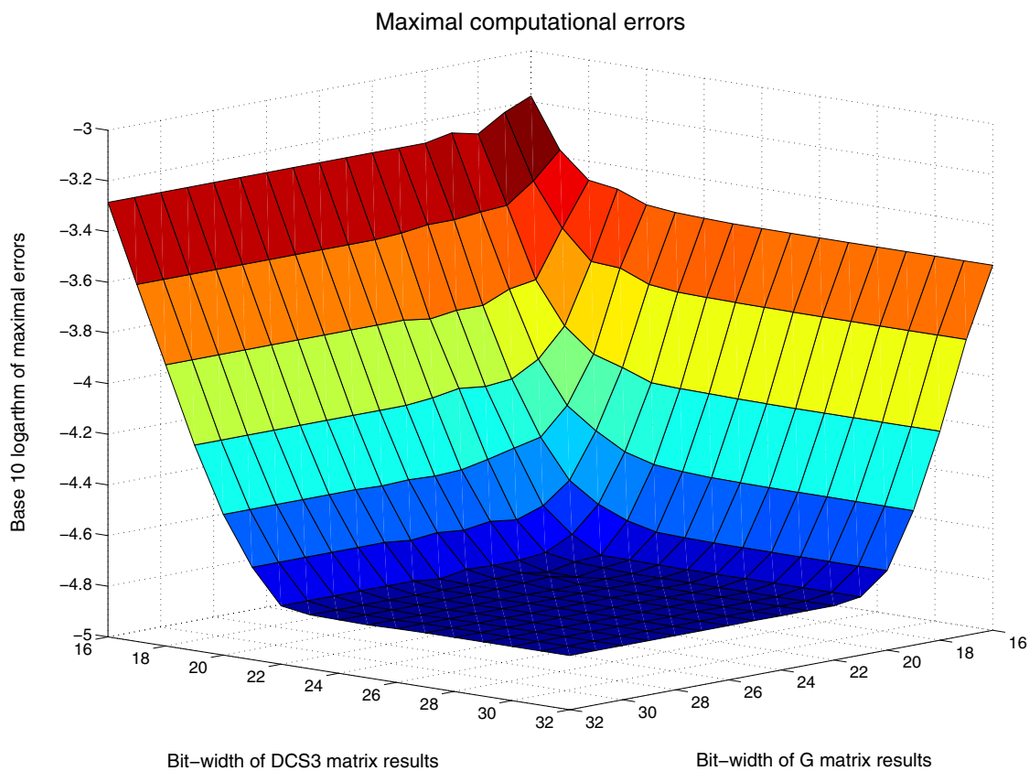

Fig. 2. Maximal computational error dependence on the bit-width of multiplication results displayed in base 10 logarithmic scale.

product versions and to interpolate between them. Higher number of situations or different versions imply a better interpolation and also more accurate estimation.
An estimated parameter is dependent on many other parameters. Each parameter included into estimation function increases the number of dimensions that the interpolation has to work with. An estimation can be simplified by reducing this number with selecting only the most important parameters. A simplified estimation can be used at the beginning of optimization process together with higher abstraction computational models. When more data are accumulated, additional parameters can be used to increase accuracy. Later, the new data can be taken from early prototypes which further improves accuracy.
More refined computational models can be transformed to independent applications. This can be useful when a product is a part of a more complex unit. Teams that design other parts can test mutual features and integration capability much sooner. Another example is the hardware-software co-design. The created applications can allow to test new software before the hardware is actually produced. Computational models with improved estimation capabilities can be transformed to applications that are offered together with final products as an extra tool. This is useful for universal products with several setting values or many parameters that can be modified by customers.
An independent application requires to design an input/output interface for users. The best choice is a graphical interface that is easy to understand and can be operated intuitively. MATLAB is a great tool to design computational models. The models can also be upgraded

with a new graphical interface. MATLAB is also able to transform their source code to independent application that can be run on a common PC.

5. Modifications of the MDCT computational model to improve hardware architecture parameters

The most important goal of computational models is to improve the resulting hardware implementation or some parameters of final chip. The original MDCT computational model had been continually upgraded. Modifications were oriented to simplify the computational structure, reduce mathematical operations and improve generality. Changes were done to the algorithm, order, type and number of mathematical operations, computational pipeline and level of parallel computations.

The first major modification is oriented to a simplification of the computational structure located between inputs and first multiplications. This part is described by mathematical equations (4) and (5) or by matrices $\mathbf{X1}_{N\times N}$, $\mathbf{X2}_{N/2\times N}$, $\mathbf{G}_{N\times N/2}$, $\mathbf{G2}_{N/2\times N}$ of the MDCT computational model. Basic algebraic manipulations applied to these equations transform several additions of input data to additions of internal constants. The constants are not changed during the whole computation process and therefore the addition of internal constants can be calculated in advance. This produces new constants that are used as replacement of the original constants and the overall number of addition operations is reduced. This process is presented in Šimlaštík et al. (2006).

Another modification is associated with a scaling factor. The MDCT algorithm uses the scaling constant nc that is applied to all results. Algebraic manipulation can change the position of this multiplication and combine it with any matrix of the MDCT computational model. The matrix $\mathbf{DCS3}_{22\times 28}$ represents multiplication by constants; however, few of them are equal to ± 0.5 that can be implemented by shift operations. Combination of the scaling factor with these constants nullifies this advantage and then a standard multiplier has to be used. The best option is to use the matrix $\mathbf{G}_{N\times N/2}$ again. In this situation, the new constants are calculated as a product of scaling factor and original constants. This results in a reduced number of multipliers used within MDCT computational structure Malík et al. (2006).

The original MDCT computational structure does not differentiate between addition of both positive, combination of positive and negative and both negative input variables. Addition of both positive input variables is implemented by an adder. Addition of one positive and one negative is implemented by a subtracter. Addition of two negative input variables is implemented by a combination of inverter and subtracter. The inverter changes the sign of the first input and then the subtracter calculates the result. The problem is that this operation is not usually implemented into standard computation blocks and extra inverters increase the final area. The solution lies in further optimization of the MDCT computational structure. The inversion can be separated from the subtraction and combined with preceding mathematical operations by applying basic algebraic manipulations. The separated operation of inversion can be completely absorbed and then the extra inverter is omitted. An example of this situation is described by equations (14) and (15).

$$e = a - b, \quad f = c + d, \quad g = -e - f, \tag{14}$$

$$e = -(a - b) = -a + b = b - a, \quad f = c + d, \quad g = e - f. \tag{15}$$

The sign change of the variable e in equation (15) is done by exchanging the input variables a and b before the variable e is calculated. Internal constants can also be used to completely

absorbed extra inversions. In this case, new constants are equal to product of -1 and original constants which is the same as inversion of the original constants. The MDCT computational model described by matrix equations is a great advantage, because the inversion separation can be written by matrix decomposition. The new matrix that depicts separated inversions is spare diagonal matrix with most diagonal elements equal to 1 and only the elements that represent inversions under investigation equal to -1. MATLAB can easily calculate the product of this matrix with a preceding matrix. If the result does not represent a situation when extra inversion can be absorbed completely, MATLAB can calculate new matrix decomposition and then the process is repeated. Optimized MDCT hardware architecture with reduced extra inverters is presented in Malík et al. (2006).

The next major modification allows computation of backward MDCT by the computational structure optimized for forward MDCT. This means that computation of forward and backward MDCT can be calculated by the same MDCT kernel. Complexity of the required changes within MDCT computational structure is not high which was presented in Cho et al. (2004) with a different MDCT algorithm. The changes are related to the inversion of several signals and modification of the input and output parts of the MDCT computational structure. Backward MDCT has twice as many outputs than inputs which is exactly opposite in comparison to forward MDCT. The matrices at the beginning and at the end of the MDCT computational structure were modified to include this. Backward MDCT uses extra scaling factor equal to $2/N$ that was absorbed to internal constants. Modified MDCT computational structure and resulted hardware architecture is presented in Malík (2007).

The MDCT computational structure is composed of many mathematical operations. When they are implemented directly the resulted hardware consumes a large area. The advantage is in high computational speed. However, higher computational speed can be achieved by increasing operational frequency and therefore the large area is a disadvantage. The area can be scaled down by separating mathematical operations to different time periods. It is actually a reduction of parallel computational level. This process requires to add memory elements into architecture to store partial results and to keep them unchanged for several computational cycles. The symmetrical computational structure, e.g., the butterfly structure, can be scaled down intuitively. In this case, all results are described by the same equation with different input variables. This means that computational structure of a single result is able to compute other results if a mechanism that selects different input variables is attached to it. The situation is more difficult when a computational structure is not symmetrical, because the universal computational structure dedicated for several results has to be designed manually. This is usually a challenging and time consuming process.

The memory elements added into computational structure can be used to introduce a pipeline computation. A pipelined computational structure represents a computational structure that is divided into several parts and all these parts compute their partial results at the same time. The aim is to design these parts so the whole computation process is continual and none part has to wait for its input data. This results in significantly increased computational speed without increasing operational frequency. A minor disadvantage is in induced computational delay. The MDCT computational structure with reduced parallel computational level and pipelined computation is presented in Malík (2007).

The MDCT computational model was upgraded with estimation capabilities. The estimated parameter is the area described in 4-input Look-Up Tables (LUTs) which are basic computational blocks used in FPGA technology. The FPGA chips were used for hardware prototyping. Estimation data were taken from elementary prototypes. The estimation uses a linear interpolation to calculate estimated results. The estimated results of this simple model had been used within the optimization process. However, later evaluation showed that the

Complete	Computation	4-input LUTs	Flip flops	Frequency	Latency
No optimization	MDCT	18661	8174	32.26 MHz	341 ns
After optimization	MDCT / IMDCT	15536	1269	41.67 MHz	192 ns

Table 1. 36-point MDCT prototypes implemented into FPGA

Complete	4-input LUTs	Flip flops	Frequency	Latency
No optimization	100 %	100 %	100 %	100 %
After optimization	83.3 %	15.5 %	129.2 %	56.3 %
Difference	-16.7 %	-84.5 %	29.2 %	-43.7 %

Table 2. Comparison of 36-point MDCT prototypes implemented into FPGA

estimated results were significantly different in comparison to prototypes. A further analysis showed that the generated set of input parameters represents hardware architecture with acceptable overall parameters and the estimation was not upgraded further.
The MDCT computational model was used to calculate optimal input parameters. These parameters represent the precision of internal constants and several partial results. The necessary input variables are input precision and overall computation precision. The optimization process is presented in Malík et al. (2006).

6. Evaluation of the optimization process

Several hardware prototypes were implemented into FPGA technology. The first prototype represents implementation of MDCT hardware architecture without any optimization. It is implementation of a complete MDCT computational structure based on the selected MDCT algorithm. The resulted parameters are shown in the first line of Table 1.
The second line of Table 1 represents parameters of the optimized MDCT computational structure with no reduction of parallel computational level. This prototype is able to calculate both forward and backward MDCT. The comparison of these two prototypes from Table 1 is shown in Table 2.
As can be seen in Table 2, all parameters were improved. The highest improvement is in reduction of internal memory elements, represented by flip flops, which are reduced by nearly 85 %. The latency is reduced by 44 % which is also induced by operational frequency increased by 29 %. The area of the combination logic, represented by 4-input LUTs, was reduced by nearly 17 %.
The MDCT implementations with a complete computational structure consume too large area and therefore several prototypes with a reduced level of parallel computation were implemented. The parameters of two such prototypes are shown in Table 3 and Table 4.
The first lines of Table 3 and Table 4 represent parameters of the first reduced MDCT computational structure that was designed directly from the complete MDCT computational structure shown in the first line of Table 1. The reduced implementation prototype has only minimal extra optimizations. The second lines of Table 3 and Table 4 represent parameters of the reduced MDCT computational structure that was designed from the optimized MDCT computational structure with no reduction of parallel computational level shown in the

Reduced	Computation	4-input LUTs	Flip flops	TBUFs	Block RAM
No optimization	MDCT	5472	2467	2070	0
After optimization	MDCT / IMDCT	3275	1752	0	0

Table 3. 36-point MDCT prototypes with reduced level of parallel computation implemented into FPGA

Reduced	Frequency	Clock period	Computation	Latency	Latency
No optimization	18.52 MHz	54 ns	18 x CLK	26 x CLK	1 404 ns
After optimization	35.71 MHz	28 ns	18 x CLK	42 x CLK	1 176 ns

Table 4. 36-point MDCT prototypes with reduced level of parallel computation implemented into FPGA

Reduced	4-input LUTs	Flip flops	TBUFs	Frequency	Latency
No optimization	100 %	100 %	100 %	100 %	100 %
After optimization	59.9 %	71.0 %	0.0 %	192.9 %	83.8 %
Difference	-40.1 %	-29.0 %	-100 %	92.9 %	-16.2 %

Table 5. Comparison of 36-point MDCT prototypes with reduced level of parallel computation implemented into FPGA

second line of Table 1. The comparison of the two implemented prototypes with a reduced level of parallel computation is shown in Table 5.

As can be seen in Table 5, all parameters were improved. The highest improvement is in increasing operational frequency by nearly 93 %. This is partially caused by introducing a computational pipeline with 3 stages. As a result the latency is reduced by 16 % even though the number of clock cycles that represent latency is increased (see the fifth column of Table 4). Internal memory elements are reduced by 29 %. The area of the combinational logic is reduced by 40 % and additionally no tree-state buffers (TBUFs) were used. The optimized reduced prototype is able to calculate both forward and backward MDCT.

The MDCT hardware prototypes were also implemented into two ASIC technologies. Comparison between implementation into AMS 350nm CMOS standard cell library and implementation into UMC 90nm CMOS low power digital library technology with clock gating technique is presented in Malík et al. (2009).

The all MDCT computational models were designed in MATLAB. Most optimization work and time consuming calculations were also done in MATLAB, which substantially contributed to overall improvements. Additionally, many internal MATLAB functions were used to generate random input values and to visualize results, which accelerated the whole optimization process significantly.

7. Conclusion

The technology of integrated circuit production has been significantly improved which resulted in very high integration of currently produced chips. The advantage is in lower cost and higher computational speed of produced chips. Higher functionality represents increased complexity of the internal computational structure that has to be designed. The designed hardware architecture has to be optimized. Each small improvement that reduces the chip final area, represents lower cost of the produced chips. This is very important in high volume productions where the savings can grow to high values. Computer aided design systems represent powerful tools that facilitate and speed up the design process. Most of these tools include complex low abstraction level models that are able to optimize parameters of the final chip. However, they are not suitable for a general optimization, because they are too computation power demanding. An increase of modern chip complexity resulted in more optimization steps that use models with higher abstraction level. MATLAB is an excellent software platform that can be used to design, verify and highly optimize these computational models. The optimization with higher abstraction level models is faster and therefore wider intervals of input parameters and more suitable solutions can be evaluated. This implies the final chip with better parameters and lower cost.

Many computational models have to be created during the whole design process. The feasibility study and early design steps are examples where the computational models are created anew. MATLAB includes extensive database of internal functions that can be directly used into computational models which reduces the time necessary to design them. The created models have to be verified and therefore the ability to generate input test data and to analyze and visualize the calculated results is necessary. This is supported by embedded MATLAB functions and the representing functions are easy to use. Additional manipulation with calculated data further improves the effect of visualizations.

The design process of computational models created in MATLAB was presented by MDCT computational models. The presented models are oriented to different tasks with a common goal to improve parameters of the resulting hardware architecture and its implementation to a chip. The effect of the optimization process was shown by the comparison of several hardware prototypes implemented into FPGA technology during different optimization phase. The parameters of the optimized prototypes are significantly improved. The improvements range from 17 % to 93 %.

8. References

Beauchamp, M. J., Hauck, S., Underwood, K. D. & Hemmert, K. S. (2008). Architectural Modifications to Enhance the Floating-Point Performance of FPGAs, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 16(2): 177–187.

Bosi, M. & Goldberg, R. E. (2003). *Introduction to Digital Audio Coding and Standards*, Springer Science + Business Media, Inc., New York, NY.

Britanak, V. (2002). The refined efficient implementation of the mdct in mp3 and comparison with other methods, *Technical Report IISAS-2002-2*, Institute of Informatics SAS, Bratislava.

Britanak, V. & Rao, K. (2002). A New Fast Algorithm for the Unified Forward and Inverse MDCT/MDST Computation, *Signal Processing* 82(3): 433–459.

Britanak, V. & Rao, K. R. (2001). An Efficient Implementation of the Forward and Inverse MDCT in MPEG Audio Coding, *IEEE Signal Processing Letters* 8(2): 48–51. Erratum: IEEE Signal Processing Letters, Vol. 8, No. 10, Oct. 2001, p. 279.

Cho, Y.-K., Song, T.-H. & Kim, H.-S. (2004). An optimized algorithm for computing the modified discrete cosine transform and its inverse transform, *Proc. of the IEEE Region 10 International Conference on Analog and Digital Techniques in Electrical Engineering*, Vol. A, Chiang Mai, Thailand, pp. 626–628.

Inacio, C. & Ombres, D. (1996). The DSP Decision fixed point or Floating?, *IEEE Spectrum Magazine of Technology Insiders* 33(9): 72–74.

ITRS (2009). International technology roadmap for semiconductors, 2009 edition executive summary, *Technical report*. `http://www.itrs.net/Links/2009ITRS/2009Chapters_2009Tables/2009_ExecSum.pdf` [Accessed: April 2011].

Kwon, T. J., Sondeen, J. & Draper, J. (2005). Design Trade-Offs in Floating-Point Unit Implementation for Embedded and Processing-In-Memory Systems, *Proc. of the IEEE Circuits and Systems*, USC Inf. Sci. Inst., Marina del Rey, CA, pp. 3331–3334.

Malík, P. (2007). A generic IP Core of Indetical Forward and Inverse 12/36 Point MDCT Architecture and an Architectural Model Simulation Toolbox, *Proc. of. the 14th IEEE International Conference on Electronics, Cirsuits and Systems*, Marrakech, Maroko.

Malík, P., Baláž, M., Pikula, T. & Šimlaštík, M. (2006). MDCT IP Core Generator with Architectural Model Simulation, *Proc. of the IFIP International Conference on Very Large Scale Integration*, Nice, France, pp. 18–23.

Malík, P., Ufnal, M., Łuczyk, A. W., Baláž, M. & Pleskacz, W. (2009). MDCT / IMDCT Low Power Implementations in 90 nm CMOS Technology for MP3 Audio, *Proc. of the IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, Liberec, Czech Republic, pp. 144–147.

Moore, G. E. (1965). Cramming more components onto integrated circuits, *Electronics* 38(8): 114–117.

Princen, J. P. & Bradley, A. B. (1986). Analysis/Synthesis Filter Bank Design Based on Time Domain Aliasing Cancellation, *IEEE Transaction on Acoustic, Speech and Signal Processing* 34(5): 1153–1161.

Princen, J. P., Johnson, A. W. & Bradley, A. B. (1987). Sub-band/transform coding using filter bank designs based on time-domain aliasing cancellation, *Proc. of the IEEE ICASSP'87*, Dallas, TX, pp. 2161–2164.

Spanias, A., Painer, T. & Atti, V. (2007). *Audio Signal Processing and Coding*, John Wiley & Sons, Inc., Hoboken, NJ.

Šimlaštík, M., Malík, P., Pikula, T. & Baláž, M. (2006). FPGA Implementation of a Fast MDCT Algorithm, *Proc. Of the IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, Praha, Czech Technical University Publishing House, pp. 228–233.

MATLAB for Engineers - Applications in Control, Electrical Engineering, IT and Robotics

Edited by Dr. Karel Perutka

ISBN 978-953-307-914-1
Hard cover, 512 pages
Publisher InTech
Published online 13, October, 2011
Published in print edition October, 2011

The book presents several approaches in the key areas of practice for which the MATLAB software package was used. Topics covered include applications for: -Motors -Power systems -Robots -Vehicles The rapid development of technology impacts all areas. Authors of the book chapters, who are experts in their field, present interesting solutions of their work. The book will familiarize the readers with the solutions and enable the readers to enlarge them by their own research. It will be of great interest to control and electrical engineers and students in the fields of research the book covers.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Peter Mañík (2011). Computational Models Designed in MATLAB to Improve Parameters and Cost of Modern Chips, MATLAB for Engineers - Applications in Control, Electrical Engineering, IT and Robotics, Dr. Karel Perutka (Ed.), ISBN: 978-953-307-914-1, InTech, Available from: http://www.intechopen.com/books/matlab-for-engineers-applications-in-control-electrical-engineering-it-and-robotics/computational-models-designed-in-matlab-to-improve-parameters-and-cost-of-modern-chips

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen