Elektrotehniški vestnik 77(1): 25–30, 2010
Electrotechnical Review, Ljubljana, Slovenija

An Iterative Logarithmic Multiplier

Zdenka Babić*, Aleksej Avramović*, Patricio Bulić †

* *Faculty of Electrical Engineering, University of Banja Luka, Patre 5, Banja Luka, BiH*
† *Faculty of Computer and Information Science, University of Ljubljana, Tržaška c. 25, Ljubljana, Slovenia*
Email: patricio.bulic@fri.uni-lj.si

Abstract. The paper presents a new multiplier enabling achievement of an arbitrary accuracy. It follows the same idea of number representation as the Mitchell's algorithm, but does not use logarithm approximation. The proposed iterative algorithm is simple and efficient and its error percentage is as small as required. As its hardware solution involves adders and shifters, it is not gate and power consuming. Parallel circuits are used for error correction. The error summary for operands ranging from 8-bit to 16-bit operands indicates a very low error percentage with only two parallel correction circuits.

Key words: Computer arithmetic, digital signal processing, multiplier, logarithmic number system

Iterativni logaritemski množilnik

Povzetek. V članku predstavimo izvedbo logaritemskega množilnika, ki nam omogoča nastavljivo natančnost. Množilnik je zasnovan na Mitchellovem postopku množenja in uporabi logaritemske predstavitve podatkov [1]. Množenje je časovno zahtevna operacija, poleg tega pa množilniki v primerjavi z drugimi aritmetičnimi vezji porabijo veliko moči. V kar nekaj aplikacijah pri digitalni obdelavi signalov, kjer je velika prisotnost šuma, pa ne potrebujemo natančnega rezultata množenja. Zato je v teh primerih primernejše logaritemsko množenje [1, 3]. Mitchell je v delu [1] predstavil osnovno idejo in izvedbo logaritemskega množilnika. Ideja je v tem, da se binarna števila predstavijo v t. i. logaritemskem zapisu (enačba 1) ter se produkt aproksimira po enačbi 3. Mitchellov postopek množenja je zapisan v Algoritmu 1. Mitchell je v svojem delu pokazal, da je največja relativna napaka okrog 11%, relativna napaka pa 3,8%. V literaturi zasledimo veliko poskusov izboljšanja natančnosti takega množilnika, največkrat s korekcijskimi tabelami, ki zahtevajo precej pomnilnika [2, 3, 4]. V tem delu predlagamo modificiran postopek množenja, s katerim izboljšamo natančnost, ob tem pa ne potrebujemo dodatnega pomnilnika za hranjenje korekcijskih termov. Osnovna ideja je, da ostanke, ki jih dobimo pri logaritemskem zapisu števil, še enkrat zmnožimo na enak način ter prištejemo h končnemu produktu kot v enačbi 5. Tako sproti tvorimo korekcijske terme. To lahko počnemo iterativno ali vzporedno, dokler ne dosežemo želene največje relativne napake (enačbi 14 in 15). Predlagani postopek množenja je prikazan v algoritmu 2. Predlagani postopek množenja smo implementirali v FPGA čipu. Osnovni blok množilnika je prikazan na sliki 1. Posamezne korekcijske terme nato izračunavamo z dodajanjem osnovnih blokov, kot je to prikazano na sliki 2. Poraba sredstev in moči v FPGA čipu brez, oz. z dodanimi korekcijskimi termi, je prikazana v tabelah 1 in 2. Rezultati analize relativne napake (tabeli 3 in 4) pokažejo, da je že s samo tremi korekcijskimi termi, največja relativna napaka pod 0,5%.

Ključne besede: Računalniška aritmetika, digitalna obdelava signalov, množilniki, logaritemski sistem

Received 16. september, 2009
Accepted 13. januar, 2010

1 Introduction

Multiplication has always been hardware, time and power consuming arithmetic operation, especially for large value operands. This bottleneck is even more emphasized in digital signal processing applications that involve a huge number of multiplications. In many signal processing applications the results of arithmetic operations do not have to be exactly accurate. For example, in signal compression techniques, quantization is usually performed after cosine or some other transform. Therefore, calculation of true transform coefficients values is not necessary and rounded products after multiplication by signal transformation are acceptable. Also, many digital signal processing systems can deal with an extra noise introduced. In these signal processing applications, where speed of calculation is more important than accuracy, logarithm number system (LNS) for multiplication seems to be a suitable method. The main advantage of this method is substitution of multiplication with addition, after conversion operands into logarithms. But this simple idea has a significant weakness: a necessity for approximation of logarithm and antilogarithm. Therefore, logarithmic-based solutions are trade-off between time consumption and accuracy. In the well known Mitchell's algorithm (MA) [1] for multiplication in LNS, a higher error is caused due to the piecewise straight line approximation of the logarithm and antilogarithm curve. Later, many methods for MA error correction are introduced with more or less success [2], [3], [4], [5], [6]. LNS multipliers can be divided into two categories, one based on methods that use lookup tables and interpolations and the other based on Mitchell's algorithm, even there is a lookup-table approach in some

of the MA-based methods. MA-based solutions suppressed lookup tables due to hardware area savings. This paper presents a new iterative solution for multiplication, where error correction is realized with parallel correction circuits. It is is organized as follows: Section 2 presents the basic Mitchell's algorithm and its modifications with their benefits and weaknesses, Section 3 describes the proposed iterative solution, in Section 4 hardware implementations of the proposed algorithm are discussed, Section 5 gives an experimental results overview, and Section 6 is a conclusion.

2 MA-Based Multipliers

A logarithmic number system is introduced to simplify multiplication, especially in cases when accuracy requirements are not rigorous. One of the most significant multiplication methods in LNS is the Mitchell's algorithm, which approximates the logarithm with a piecewise straight line function. MA multiplies two operands by finding their logarithms, adding them and looking for the antilogarithm of the sum.

Approximation of the logarithm and antilogarithm is essential. It is derived from binary representation of numbers:

$$N = 2^k(1 + \sum_{i=j}^{k-1} 2^{i-k} Z_i) = 2^k(1 + x) \qquad (1)$$

where k is the characteristic number or place of the most significant bit with the value of '1', Z_i is the bit value at i-th position, x is a fraction or mantissa and j depends on the number precision. By logarithm of the product computation,

$$\log_2(N_1 \cdot N_2) = k_1 + k_2 + \log_2(1+x_1) + \log_2(1+x_2) \quad (2)$$

$\log_2(1 + x_1)$ is approximated with x_1 and the logarithm of the two number product is expressed as a sum of their characteristic numbers and mantissas:

$$\log_2(N_1 \cdot N_2) \approx k_1 + k_2 + x_1 + x_2 \qquad (3)$$

The antilogarithm uses the similar approximation.

The final MA approximation for multiplication, depending on the carry bit from sum of mantissas is given by:

$$(N_1 \cdot N_2)_{MA} = \begin{cases} 2^{k_1+k_2}(1 + x_1 + x_2), & x_1 + x_2 < 1 \\ 2^{k_1+k_2+1}(x_1 + x_2), & x_1 + x_2 \geq 1 \end{cases} \qquad (4)$$

The sum of the characteristic numbers determines MSB of the product. The sum of mantissas is added to complete the final result. The proposed MA-based multiplication for the case $x_1 + x_2 < 1$ is given in Algorithm 1. MA produces a significant error percentage. The relative error increases with the number of bits with the value of '1' in the mantissas. The maximum possible relative error for MA multiplication is some 11% and the average error is some 3.8%. Mitchell analyzed this error and proposed analytical expressions for error correction. To sum up, the most significant advantage of MA is simplicity, efficiency, i.e. non power-consuming. The most significant disadvantage is a high error percentage.

Algorithm 1 Mitchell's Algorithm for the case $x_1 + x_2 < 1$

1. N_1, N_2: n-bits binary multiplicands, $P_{MA} = 0$: $2n$-bits approximate product
2. Calculate k_1: leading one position of N_1
3. Calculate k_2: leading one position of N_2
4. Calculate x_1: shift N_1 to the left by $n - k_1$ bits
5. Calculate x_2: shift N_2 to the left by $n - k_2$ bits
6. Calculate $k_{12} = k_1 + k_2$
7. Calculate $x_{12} = x_1 + x_2$
8. Decode k_{12} and insert '1' in that position of P_{approx}
9. Append x_{12} immediately after this one in P_{approx}
10. $N_1 \cdot N_2 = P_{MA}$

Numerous attempts have been made to improve the MA accuracy. Hall [4] derived different equations for error correction in the logarithm and antilogarithm approximation in four separate regions, depending on the mantissa value, reducing the average error to 2%, but increasing complexity of realization. Abed and Siferd [5], [6] derived correction equations with coefficients that are power of two, reducing the error and keeping the simplicity of solution. Among many methods that use look-up tables for error correction in the MA algorithm, McLaren's method [2], which uses a lookup table with 64 correction coefficients calculated in dependence on the mantissas values, can be selected as one with a satisfactory accuracy and complexity. A recent approach to MA error correction reducing the number of bits with the value of '1' in mantissas by operand decomposition was presented by Mahalingam and Rangantathan [3]. The proposed method decreases the error percentage of the MA by 44.7% on the average.

3 Proposed Solution

As already mentioned above, the basic disadvantage in the Mitchell's algorithm and similar solutions comes from

logarithm approximation. Therefore, the proposed solution avoids logarithm approximation and introduces an iterative algorithm with various possibilities for maximally minimizing the errorand getting an exact result. As the proposed solution is an iterative but not a recursive algorithm, it can be realized with parallel circuits for error reduction.

From the binary representation of numbers in (1), we can derive a correct expression for multiplication:

$$\begin{aligned} P_{true} &= N_1 \cdot N_2 = 2^{k_1}(1+x_1) \cdot 2^{k_2}(1+x_2) \\ &= 2^{k_1+k_2}(1+x_1+x_2) + 2^{k_1+k_2}(x_1 x_2) \end{aligned} \quad (5)$$

The similarity with MA is evident. The error in MA is caused by neglecting the second term in (5). To avoid the approximation error, we have to take into account the bellow relation:

$$x \cdot 2^k = N - 2^k \quad (6)$$

The combination of (5) and (6) gives:

$$\begin{aligned} P_{true} &= (N_1 \cdot N_2) = 2^{(k_1+k_2)} + \\ &+(N_1 - 2^{k_1})2^{k_2} + (N_2 - 2^{k_2})2^{k_1} + \\ &+(N_1 - 2^{k_1}) \cdot (N_2 - 2^{k_2}) \end{aligned} \quad (7)$$

Let

$$P_0 = 2^{(k_1+k_2)} + (N_1 - 2^{k_1})2^{k_2} + (N_2 - 2^{k_2})2^{k_1} \quad (8)$$

be the first approximation of the product. It is evident that

$$P_{true} = P_0 + (N_1 - 2^{k_1}) \cdot (N_2 - 2^{k_2}) \quad (9)$$

If we approximate the product with

$$P^{(0)}_{approx} = P_0 \quad (10)$$

then the proposed method is very similar to MA. Actually, P_0 is equal to the first case in the MA approximation (4). Mitchell proposed an exact correction term as in (9), but the logarithm approximation-based multiplying of the given residues was not sufficient to achieve significant error decreasing. Avoiding the logarithm approximation enables parallel error corrections and more accurate product. For this reason, an iterative calculation of correction terms is proposed as follows.

An absolute error after the first approximation is

$$\begin{aligned} E^{(0)} &= P - P^{(0)}_{approx} = P - P_0 = \\ &= (N_1 - 2^{k_1}) \cdot (N_2 - 2^{k_2}) \end{aligned} \quad (11)$$

Note that $E^{(0)} \geq 0$. The two multiplicands in (11) are binary numbers that can be obtained simply by removing the leading '1' in numbers N_1 and N_2 so we can repeat the proposed multiplication procedure with these new multiplicands

$$E^{(0)} = C^{(1)} + E^{(1)} \quad (12)$$

where $C^{(1)}$ is the approximate value of $E^{(0)}$ and $E^{(1)}$ is an absolute error when approximating $E^{(0)}$. The combination of (9) and (12) gives

$$P_{true} = P_0 + C^{(1)} + E^{(1)} \quad (13)$$

We can now add the approximate value of $E^{(0)}$ to approximate product P_{approx} as an correction term with which we decrease the error of approximation

$$P^{(1)}_{approx} = P_0 + C^{(1)} \quad (14)$$

If we repeat this multiplication procedure with i correction terms, we can approximate the product as

$$\begin{aligned} P^{(i)}_{approx} &= P_0 + C^{(1)} + C^{(2)} + \ldots + C^{(i)} = \\ &= P_0 + \sum_{j=1}^{i} C^{(j)} \end{aligned} \quad (15)$$

The procedure can be repeated in order to achieve the smallest possible, or until at least one of the mantissas becomes a zero. Then the final result is exact: $P_{approx} = P_{true}$. The number of iterations required for exact result is equal to the number of bits with the value of '1' in the operand with a smaller number of bits with the value of '1'. The proposed iterative MA-based multiplication is given in Algorithm 2.

One of the advantages of the proposed solution is the possibility to achieve an arbitrary accuracy by selecting a number of iterations, i.e. a number of parallel correction circuits.

4 Hardware Implementation

In order to evaluate the device utilization and performance of the proposed multiplier, we implemented different multipliers on Xilinx xc3s500e-4fg320 FPGA [7]. We implemented four 16-bits multipliers: a multiplier with no correction terms, and three multipliers with two, three and four correction terms, respectively.

4.1 Basic Block

The basic block is the proposed multiplier with no correction terms. The task of the basic block is to calculate one approximate product according to Equation 8. The 16-bit basic block is presented in Figure 1. The 16-bit basic block consists of two leading-one detector units (LOD), two encoders, two 32-bit barrel shifters, a decoder unit and two 32-bit adders. Two input operands are given to LODs and the encoders. The LOD units are used to remove the leading one from the operands, which are then passed to the barrel shifters. The LOD units include zero detectors, too. They are used to detect zero operands. The LOD units and zero detectors are implemented as in [5].

Algorithm 2 Iterative MA Based Algorithm with i correction terms

1. N_1, N_2: n-bits binary multiplicands, $P_0 = 0 : 2n$-bits first approximation, $C^{(i)} = 0 : 2n$-bits i correction terms, $P_{approx} = 0 : 2n$-bits product
2. Calculate k_1: leading one position of N_1
3. Calculate k_2: leading one position of N_2
4. Calculate $(N_1 - 2^{k_1})2^{k_2}$: shift $(N_1 - 2^{k_1})$ to the left by k_2 bits
5. Calculate $(N_2 - 2^{k_2})2^{k_1}$: shift $(N_2 - 2^{k_2})$ to the left by k_1 bits
6. Calculate $k_{12} = k_1 + k_2$
7. Calculate $2^{(k_1+k_2)}$: decode k_{12}
8. Calculate P_0 : add $2^{(k_1+k_2)}$, $(N_1 - 2^{k_1})2^{k_2}$ and $(N_2 - 2^{k_2})2^{k_1}$
9. Repeat i-times:
   (a) Set: $N_1 = N_1 - 2^{k_1}$, $N_1 = N_2 - 2^{k_2}$
   (b) Calculate k_1: leading one position of N_1
   (c) Calculate k_2: leading one position of N_2
   (d) Calculate $(N_1 - 2^{k_1})2^{k_2}$: shift $(N_1 - 2^{k_1})$ to the left by k_2 bits
   (e) Calculate $(N_2 - 2^{k_2})2^{k_1}$: shift $(N_2 - 2^{k_2})$ to the left by k_1 bits
   (f) Calculate $k_{12} = k_1 + k_2$
   (g) Calculate $2^{(k_1+k_2)}$: decode k_{12}
   (h) Calculate $C^{(i)}$: add $2^{(k_1+k_2)}$, $(N_1 - 2^{k_1})2^{k_2}$ and $(N_2 - 2^{k_2})2^{k_1}$
10. $P^{(i)}_{approx} = P_0 + \sum_i C^{(i)}$

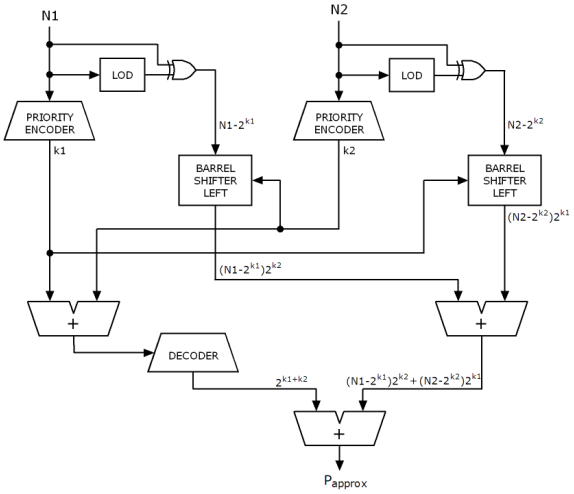

Figure 1. Block diagram of a basic block of the proposed iterative multiplier.

The barrel shifters are used to shift residues according to Equation 8. The decode unit decodes $k_1 + k_2$, i.e. puts the leading one in the product. The leading one and two shifted residues are then added to form the approximate product. The basic block is used in further implemenations to calculate P_0 and $C^{(i)}$.

4.2 Parallel Implementation

We implemented multipliers with parallel correction circuits. For this purpose, we used the cascade of the basic blocks. The block diagram of the proposed logarithmic multiplier with a parallel error-correction circuit is shown in Figure 2. The multiplier is composed of two basic blocks of which the first one calculates the first approximation of the product (P_0) while the second one calculates the error correction term $C^{(1)}$.

4.3 Device Utilization

For design entry we used Xilinx ISE 10.1.02 - WebPACK and design with VHDL. The design was synthetised with Xilinx Xst Release 10.1.02 for Linux.

Device utilization (the number of slices, number of 4-input LUTs and number of input-output blocks) for all the four implemented multipliers is given in Table 1.

Table 1. Device utilization.

Multiplier	4-LUTs	Slices	IOBs
Basic Block	353	182	96
Basic Block + 1CT	736	381	96
Basic Block + 2CT	1088	577	96
Basic Block + 3CT	1438	751	96

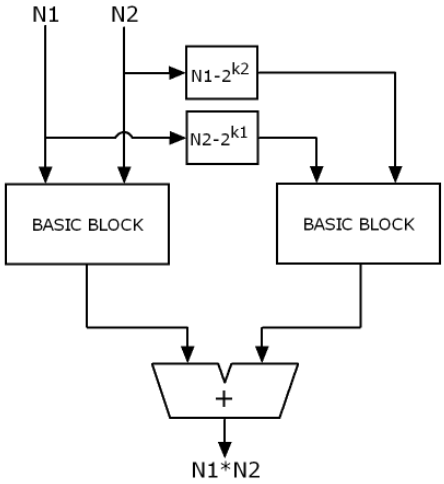

Figure 2. Block diagram of the proposed multiplier with one parallel error-correction circuit. The multiplier is composed of two basic blocks of which the first one calculates the approximate product while the second one calculates the error-correction term.

The maximum combinational path delays along with the total power consumptions for the basic block and the three parallel implementations are given in Table 2.

5 Error Analysis

In order to evaluate the proposed algorithm, it is is applied to all combinations of the *n*-bit non-negative numbers. The error percentage is calculated from the well-known equation:

$$E_i = \frac{P_{true} - P_{approx}}{P_{true}} \cdot 100\% \tag{16}$$

For evaluation, the average error percentage value is used:

$$AE = \frac{1}{N} \sum_{i=1}^{N} E_i \tag{17}$$

where N is the number of multiplications performed. For example, for the 12-bits numbers, all combinations of the numbers from 1 to 4095 are multiplied and the average error percentage is calculated. Error calculation is made in four cases: without error-correction parallel circuit, with one parallel correction, with two parallel corrections and with three parallel corrections. Results from Table 3 present the error percentage rate for various cases. Results from Table 4 give a more precise view on how the algorithm can be modified to wanted error percentage.

The obtained results are compared with results from [3], for neing the latest paper providing a complete overview of various solutions. Comparing these solutions with other solutions, a similar error table is obtained. Comparing 8-bit and 16-bit average error percentages, we can notice that our solution with three iterations outperforms others logarithm based-multipliers.

6 Conclusions

In this paper, we investgate and propose a new approach to improve the accuracy of the Mitchell algorithm-based multiplication. The approach is based on the iterative calculating of the correction terms. We show that the calculation of correction terms can be performed parallelly in hardware.

The iterative approach improves the average error percentage and the error rate compared to the basic MA multiplication. By using only three correction terms, the error of any multiplication result is less than a 0.5%.

The parallel implementation of the iterative MA multiplier with only one correction circuit almost doubles the area required compared to the original MA multiplier, but the power consumption increases only from 2% (one correction term) to 16% (three correction terms). This is still significantly less than the area and power required for a standard multiplier.

The maximum combinational delay increases by 30-45% with each added correction circuit. This can be further significantly improved by pipelining the three main stages in the basic block and the correction circuits.

Acknowledgments

This work was funded by Slovenian Research Agency (ARRS) through grants P2-0359 and BA/10-11-026.

7 References

[1] J.N. Mitchell, Computer multiplication and division using binary logarithms, *IRE Transactions on Electronic Computers*, vol. EC-11, pp. 512-517, August 1962.

[2] D.J. Mclaren, Improved Mitchell-based logarithmic multiplier for low-power DSP applications, *Proceedings of IEEE International SOC Conference 2003* pp. 53-56, 17-20 September 2003.

[3] V. Mahalingam, N. Rangantathan, Improving Accuracy in Mitchells Logarithmic Multiplication Using Operand Decomposition, *IEEE Transactions on Computers*, Vol. 55, No. 2, pp. 1523-1535, December 2006.

[4] E.L. Hall, D.D. Lynch, S. J. Dwyer III, Generation of Products and Quotients Using Approximate Binary Logarithms for Digital Filtering Applications, *IEEE Transactions on Computers*, Vol. C-19, No. 2, pp. 97-105. February 1970.

[5] K.H. Abed, R.E. Sifred, CMOS VLSI Implementation of a Low-Power Logarithmic Converter, *IEEE Transactions on Computers*, Vol. 52, No. 11, pp. 1421-1433, November 2003.

[6] K.H. Abed, R.E. Sifred, VLSI Implementation of a Low-Power Antilogarithmic Converter, *IEEE Transactions on Computers*, Vol. 52, No. 9, pp. 1221-1228, September 2003.

Table 2. Synthesis results.

Multiplier	Max. combinational delay (ns)	Total power (W)
Basic Block	24.818	0.13031
Basic Block + 1CT	32.214	0.13855
Basic Block + 2CT	37.261	0.14743
Basic Block + 3CT	41.555	0.15687

Table 3. Error percentage rate [%].

Error	8 bits			12 bits			16 bits		
	1CT	2CT	3CT	1CT	2CT	3CT	1CT	2CT	3CT
$< 0{,}1$ %	32.9	79.9	99.0	20.6	71.6	98,2	19.3	70.6	98.0
$< 0{,}5$ %	54.8	96.9	100	48.1	95.7	100	47.4	95.5	100
< 1 %	69.9	99.6	100	65.6	99.4	100	65.2	99.4	100

Table 4. Average relative errors [%] for 0, 1, 2 and 3 correction terms.

No. bits	Basic MA	1 C. term	2 C. terms	3 C. terms
8	8.9131	0.8337	0.0708	0.0048
9	9.1336	0.8980	0.0845	0.0069
10	9.2595	0.9369	0.0936	0.0086
11	9.3301	0.9597	0.0994	0.0098
12	9.3692	0.9726	0.1029	0.0106
13	9.3906	0.9799	0.1049	0.0111
14	9.4023	0.9840	0.1060	0.0114
15	9.4086	0.9862	0.1067	0.0116
16	9.4124	0.9874	0.1070	0.0117

[7] Xilinx Inc. Spartan-3E FPGA Family: Complete Data Sheet, DS312. *http://www.xilinx.com*, April 18, 2008.

Zdenka Babić received her B.Sc., M.Sc. and Ph.D. degrees in electrical engineering from the Faculty of Electrical Engineering, University of Banja Luka, Bosnia and Herzegovina, in 1983, 1990 and 1999 respectively. She is an Associate Professor at the same faculty. Her main research interests are digital signal processing, image processing, circuits and systems. She is a member of the IEEE Signal Processing Society and IEEE Circuits and Systems Society.

Aleksej Avramović received his B.Sc. degree in electrical engineering from the Faculty of Electrical Engineering, University of Banja Luka, Bosnia and Herzegovina, in 2007. He is a Teaching Assistant at same faculty. His research interests include digital signal processing and digital image processing. He is a student member of IEEE.

Patricio Bulić received his B.Sc. degree in electrical engineering, and M.Sc. and Ph.D. degrees in computer science from the University of Ljubljana, Slovenia, in 1998, 2001 and 2004, respectively. He is an Assistant Professor at the Faculty of Computer and Information Science, University of Ljubljana. His main research interests include computer architecture, digital design, parallel processing and vectorization techniques. He is a member of the IEEE Computer Society and ACM.