Computing and Informatics, Vol. 33, 2014, 721–734

ISOMORPHISM BETWEEN LINEAR CODES AND ARITHMETIC CODES FOR SAFE DATA PROCESSING IN EMBEDDED SOFTWARE SYSTEMS

Peter Raab, Stefan Krämer, Jürgen Mottok

Laboratory for Safe and Secure Systems
Regensburg University of Applied Sciences
Faculty of Electronics and Information Technology
Seybothstr. 2, D-93053 Regensburg, Germany
e-mail: peter.raab@extern.oth-regensburg.de,
{stefan.kraemer, juergen.mottok}@oth-regensburg.de

Vlastimil Vavřička

University of West Bohemia
Faculty of Applied Sciences
Univerzitní 8, 306 14 Plzeň, Czech Republic
e-mail: vavricka@kiv.zcu.cz

Abstract. We present a transformation rule to convert linear codes into arithmetic codes. Linear codes are usually used for error detection and correction in broadcast and storage systems. In contrast, arithmetic codes are very suitable for protection of software processing in computer systems. This paper shows how to transform linear codes protecting the data stored in a computer system into arithmetic codes safeguarding the operations built on this data. Combination of the advantages of both coding mechanisms will increase the error detection capability in safety critical applications for embedded systems by detection and correction of arbitrary hardware faults.

Keywords: Coding theory, linear codes, arithmetic codes, code transformation, residue error probability, Safely Embedded Software (SES)

Mathematics Subject Classification 2010: 94B05, 94B40, 11T71, 14G50

1 INTRODUCTION

The complexity and functionality of electronic controlled units have increased ever more in several sectors of industry during recent years (e.g. automotive, aeronautics). In addition, the requirements of these systems have become more demanding in terms of safety, reliability and availability. In contrast to this progress, industry demands a decrease in costs for electronics, while at the same time remaining competitive. The use of inexpensive commodity hardware is the result. However, the development of present micro-controllers follows the trend of decreasing feature size that leads to less reliability; arbitrary hardware faults are more likely [1]. Increasing the fault tolerance of unreliable hardware is often a requirement in safety critical applications. The consequence is the use of redundant hardware or of diverse data [2, 3, 4]. The Laboratory for Safe and Secure Systems at the University of Applied Sciences in Regensburg developed, in collaboration with the TU Munich, the Safely Embedded Software technique for the C programming language to safeguard the execution of code on microprocessors [5, 6, 7, 8].

Modern broadcast and storage systems follow similar strict requirements for reliability and safety. Disturbing influences can corrupt transmitted or stored data in the same way they do during computation in a computer system. Techniques of error detection and correction have been studied since Hamming researched codes for increasing fault tolerance in storage systems for the first time [12]. Based on this, a lot of improvements for error detection and correction were made: Cyclic Linear Block Codes [13], Bose-Chaudhuri-Hocquenghem (BCH) codes [14] and Reed-Solomon codes [15], just to mention the most important ones; but, linear codes are not the optimal solution for coded data processing because they do not preserve the code after arithmetic operations [11]. For an optimal error detection capability, code transformations can be a possible solution for further improvements also in coded data processing systems with memories and bus which have similar characteristics as storage and transmission systems. For this reason, this article presents the required background (Section 2) for a method to transform linear codes into arithmetic codes (Section 3) and the performance in Section 4.

2 BACKGROUND

The ISO 26262 norm, “Road vehicles – Functional Safety”, recommends several diagnostic techniques to detect possible occurring errors which are the state-of-the-art. In Table ISO 26262-5, D.4 [16], only two techniques with high diagnostic coverage (DC) are enumerated. Due to this norm, the highest DC is achievable only with coded processing (like the SES framework) and reciprocal comparison by software. The former executes the code in a transformed domain. Permanent errors and transient errors are detected by a check of the validity of the code words. Based on the normative approach, a closer look at coding techniques is valuable to get deeper insights.

An important metric for comparison of different codes is the residual error probability. This is the probability that a received code word is corrupted but no errors are detected by the decoding algorithm. This is the case when the erroneous bits in the received code word itself form a valid code word [20]. Based on a binary symmetric channel model (BSC, see Figure 1), the analysis of linear block codes and arithmetic codes (so-called AN code) shows that the probability for undetected errors is greater for AN codes than for linear codes with the same code rate [11].

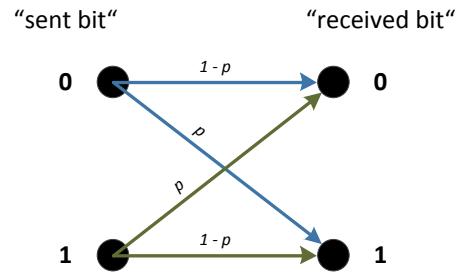


Figure 1. The binary symmetric channel model describes the probability p that a single bit changes its value or remains unchanged $(1 - p)$. There is no dependency between two nearby bits. In contrast, there are other channels with memory described in [21].

The BSC model is valid for channels where the single bits have no influence on others. Thus, linear codes are excellent for protecting single-bit errors in data storage and transmission, whereas the underlying channel model for arithmetic operations has a kind of memory. The carry bit propagation of an addition has a direct influence on nearby bits of a code word. Figure 2 shows a simple model of a computing system. It consists of a data storage unit and a transmission bus for which the BSC model is valid. Otherwise the BSC model cannot be used for the *arithmetic logical unit* (ALU). An applicable channel model for the ALU must be developed in future work.

Remark. IBM developed a code, which is also called *Arithmetic Code*, for lossless data compression [17]; but not to be confused, this arithmetic encoding is part of entropy encoding whereas the described AN codes are an example of channel encoding and error detecting codes.

2.1 Algebra of Codes

Algebraic structures are the background for most error detecting and correcting codes. An algebraic structure consists of a set of objects (e.g. numbers) and at least one operation applied to this set [22]. Two important codes are the linear codes and arithmetic codes, which have different algebraic structures [11]. An arithmetic code

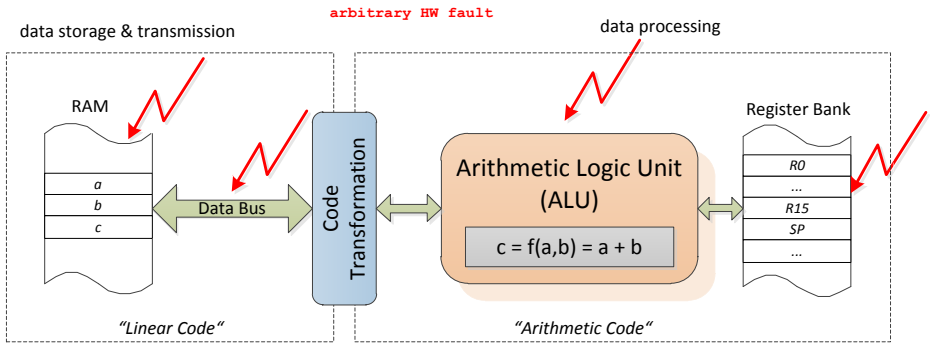


Figure 2. Simplified CPU model that shows the hardware components propagating arbitrary hardware faults. Using an adequate code for the different channel models, a transformation of the code is required.

is a set of code words which are the product of two integer numbers X (= original number) and A (= constant generator).

$$C_{AN} := \{A \cdot X \mid A, X \in \mathbb{Z}\} \tag{1}$$

A finite set of numbers is more important for computer arithmetic because of the limited register width of k bits in a micro-controller. The code words are out of the set of all multiples of A, but smaller than the possible range of $M = 2^k$. Such a subset of integer numbers is called an *ideal* $A\mathbb{Z}_M$ in algebra. This finite subset forms a ring under the two operations, addition and multiplication $(A\mathbb{Z}_M, +_M, \cdot_M)$. The sum and the product of two code words are divisible by the generator A and the result is therefore a valid code word (see axiom of closure [22]). Linear codes use other algebraic structure than arithmetic codes [19]. Instead of integer numbers, the structure of linear codes consists of a more complex set of polynomials. A polynomial is a different representation of a vector in a k-dimensional vector space [22]. The coefficients of a polynomial are the digits of a number described by a positional notation system. In a computer system, the number system is based on the finite field of order two, the so-called Galois Field *GF(2)* or $\mathbb{Z}_2 := \{0, 1\}$. Thus an integer number X is represented by the set of k binary digits in a computer system

$$\vec{x} = (x_{k-1}, x_{k-2}, \ldots x_1, x_0)$$

with $x_i \in \mathbb{Z}_2$, or in the polynomial representation:

$$\begin{aligned} x(z) &= x_{k-1} \cdot z^{k-1} + x_{k-2} \cdot z^{k-2} \ldots + x_1 \cdot z^1 + x_0 \\ x(z) &= \sum_{i=0}^{k-1} x_i \cdot z^i. \end{aligned} \tag{2}$$

The set of polynomials with coefficients out of the finite field $\mathbb{Z}_2$

$$\mathbb{Z}_2[z] := \left\{ p(z) = \sum_{i=0}^{k-1} x_i \cdot z^i \,\middle|\, x_i \in GF(2) \right\} \tag{3}$$

forms a ring of polynomials and under the two operations:

$$\begin{aligned} a(z) \oplus b(z) \;&:=\; c(z) \\ &\quad \text{with } c_j = (a_j + b_j) \mod 2 \qquad \text{and } 0 \le j < k \end{aligned} \tag{4}$$

$$\begin{aligned} a(z) \odot b(z) \;&:=\; c(z) \\ &\quad \text{with } c_j = \left(\sum_{i=0}^{j} a_i \cdot b_{j-i} \right) \mod 2 \qquad \text{and } 0 \le j < k. \end{aligned} \tag{5}$$

Linear codes have a coding rule similar to that of arithmetic codes. Both codes are generated by the multiplication based on their algebraic structure. This is the ordinary integer multiplication for arithmetic codes and the polynomial multiplication for linear codes.

$$C_{CRC} := \{ g(z) \odot x(z) \mid g(z), x(z) \in \mathbb{Z}_2[z] \} \tag{6}$$

2.2 Systematic Codes

Systematic codes are known from linear codes in communication systems. They consist of k bits of information that is separated into the $n-k$ bits of parity in their binary representation [19]. Arithmetic codes can also be in a systematic form (see Figure 3).

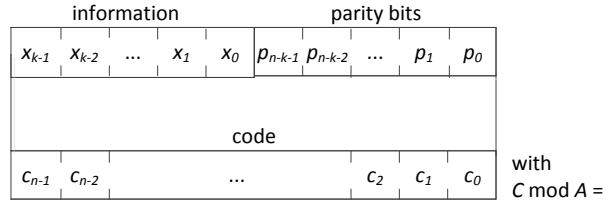


Figure 3. The information of a systematic code word is separated into two segments. The original number can be read directly from the code word. The parity bits are added afterwards. The code word itself remains a multiple of A.

For systematic arithmetic codes, the code is not separable. The addition of two systematic code words propagates possible carry bits into the information part. The separated information of the result does not match the sum of both information words (see Section 2.3). Systematic encoding is the basic principle for the

code transformation described in Section 3. Rao already showed parallels between arithmetic codes and linear block codes (see Table 1) [18].

Linear Block Codes	**Arithmetic Codes**
$c(z) = g(z) \odot x_2(z) = x_1(z) \odot z^{n-k} \oplus r(z)$	$C = A \cdot X_2 = X_1 \cdot 2^{n-k} + R$
with:	*with:*
$c(z)$: code polynomial	C : coded integer
$g(z)$: generator polynomial	A : generator integer
$x_1(z)$: information polynomial	X_1 : information integer
$r(z)$: remainder polynomial	R : remainder integer
n : length of code word in bits	
k : length of information word in bits	
$n - k$: number of redundant bits	

Table 1. Comparison between systematic linear and arithmetic codes

When analyzing the coding rules (Table 1) for both codes, we can see that $X_1 \neq X_2$ and $x_1(z) \neq x_2(z)$. It is clear that X_2 must be greater than X_1 to fulfill the equation. When coding a number, the terms X_1 and $x_1(z)$ represent the original values. Therefore, X_2 and $x_2(z)$ are not important and need not be considered.

2.3 Coded Operations

The transformation of one algebraic structure into another one is called *homomorphism*. In general, a homomorphism maps one algebraic structure into the other. Let $(G, +)$ and $(H, \oplus)$ be two algebraic structures and φ the map function of the sets $G \to H$, then it must be $\varphi(x + y) = \varphi(x) \oplus \varphi(y)$. If there is a bijective homomorphism between $(G, +)$ and $(H, \oplus)$, then both structures are isomorphic [22]. The addition of polynomials differs from that of an ordinary addition. A polynomial addition must be enhanced in a way that the above rule for a homomorphism is satisfied. The coefficients of the polynomial are elements out of the Galois Field *GF(2)* and possible carry bits are ignored. This difference between the two operations results in different outcomes and makes a correction $c(z)$ necessary. For the addition of two integers, it is

$$\varphi_+ : \quad X + Y \to x(z) \oplus y(z) \oplus c(z). \tag{7}$$

In [11], it was shown that the coded operation which is used for coded software processing [6, 4] is a homomorphism. It defines operations of code words in such a way that the result of this operation matches the coded result of the original information word.

Example 1. For the addition of two arithmetic coded numbers,

$$+_c : \varphi_+(C_1, C_2) := C_1 + C_2 \tag{8}$$

for all $C \in C_{AN}$. This means that no correction is necessary in the case of the addition of two code words (in contrast to coded multiplication [11]). For systematic encoded numbers, the sum of two coded numbers is also a systematic code word.

$$\begin{aligned} C_1 + C_2 &= X_1 \cdot 2^{n-k} + R_1 + X_2 \cdot 2^{n-k} + R_2 \\ &= (X_1 + X_2) \cdot 2^{n-k} + (R_1 + R_2). \end{aligned} \tag{9}$$

With an ordinary adder unit in an ALU, there is the problem of an overflow of the sum of the remainders R_1 and R_2. The carry bit will be propagated into the information X of a systematic code. The result will be a valid code word, because the result remains a multiple of A, but this carry bit propagation changes the value of the information X. A special version of an adder must be used. Both parts, the information and the remainder, must be handled separately (Figure 4).

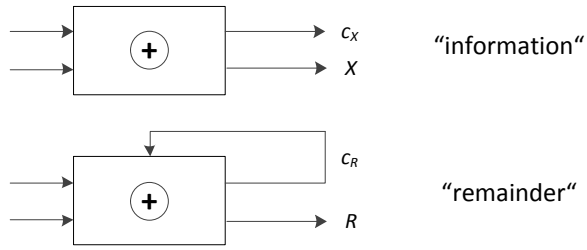


Figure 4. Special adder unit for addition of systematic arithmetic codes

When an overflow is detected in the remainder unit, the carry bit must not be propagated to the information part; but then the remainder does not correspond with the resulting information. The information number is not increased by the propagated carry-bit.

Remark. Rao and previously Garner have described another form of systematic codes. The so-called gAN code does not require special adders. For detailed information see [18].

3 CODE TRANSFORMATION

In Section 2, we saw a close similarity between linear and arithmetic codes. Obviously, there is a simple rule to transform them to each other. This section shows what the generator integer A and the polynomial $g(z)$ must look like for a transformation. Let us begin with one important theorem from Rao [18].

Theorem 1. A systematic AN code has the form $C = X_1 \cdot 2^{n-k} + R = A \cdot X_2$ if and only if $R = (-X_1 \cdot 2^{n-k}) \bmod A$ with $2^{n-k-1} < A < 2^{n-k}$.

Following Theorem 1, the remainder of a systematic AN code can be generally expressed as follows:

$$\begin{aligned} C &= X_1 \cdot 2^{n-k} + R = A \cdot X_2 \\ \Rightarrow R &= A \cdot X_2 - X_1 \cdot 2^{n-k}. \end{aligned} \tag{10}$$

With $X_2 = X_1 + 1$, the remainder R of a systematic code is

$$R = A - X_1 \cdot (2^{n-k} - A). \tag{11}$$

The term $2^{n-k} - A$ in Equation (11) is 1, if it is $A = 2^{n-k} - 1$. The remainder can be simplified to

$$R = A - X_1. \tag{12}$$

The remainder is always positive ($R \geqq 0$). Consequently, $X_1 \leqq A$ and the range of X_1 depends on A. If the generator A is of the form $2^{n-k} - 1$ (all bits are 1), the remainder of the systematic code words are decreasing numbers beginning with A (see example in Table 3).

Because of the homomorphism between the algebra of arithmetic codes and the algebra of linear codes (see Section 2), the integer numbers and the operations of Equation (11) are substituted as seen in Table 2:

R	$\rightarrow$	$r(z):$	polynomial of remainder
X_1	$\rightarrow$	$x_1(z):$	polynomial of information word
A	$\rightarrow$	$a(z):$	polynomial of generator
$X_1 - X_2$	$\rightarrow$	$x_1(z) \oplus x_2(z) \oplus c(z):$	subtraction of polynomials with $c(z)$ describes the borrow bits
$X_1 \cdot X_2$	$\rightarrow$	$x_1(z) \odot x_2(z) \oplus c(z):$	multiplication of polynomials with $c(z)$ describes the carry bits

Table 2. Transformation rules from integer numbers to polynomials

The three terms of Equation (11) are substituted step by step now:

1. The subtraction $(2^{n-k} - A)$ causes a borrow bit in every consecutive digit. The borrow bits $c(z)$ are the same as $a(z)$ but shifted by one to the left. The term $c(z)$ can be replaced by $a(z) \odot z$.
2. The multiplication $X_1 \cdot (..)$ can be simplified. The subtraction within the brackets always results in 1. A multiplication of a polynomial with 1 does not generate any carry bits. The term $c(z)$ for this multiplication can be ignored.
3. The last subtraction $A - X_1 \cdot (..)$ is also a special case. If $A = 2^{n-k} - 1$, then all $n - k$ coefficients of the polynomial $a(z)$ are equal to 1. The subtraction of any polynomials of the same or smaller order (that is, $X \leqq A$, see definition above) does not result in any borrow bits. Consequently, $c(z)$ can be ignored, too.

It follows

$$\Rightarrow r(z) = a(z) \oplus x_1(z) \odot \{z^{n-k} \oplus a(z) \oplus a(z) \odot z\}. \tag{13}$$

Expanding the brackets and reordering Equation (13) leads to:

$$\begin{aligned} r(z) &= a(z) \oplus x_1(z) \odot z^{n-k} \oplus x_1(z) \odot a(z) \oplus x_1(z) \odot a(z) \odot z \\ r(z) &= a(z) \oplus x_1(z) \odot z^{n-k} \oplus a(z) \odot z \odot \{x_1(z) \odot z^{-1} \oplus x_1(z)\}. \end{aligned} \tag{14}$$

The term $x_1(z) \odot z^{-1} \oplus x_1(z)$ of Equation (14) is the binary XOR operation of $x_1(z)$ with itself but shifted by one bit to the right. If $x_2(z) = x_1(z) \odot z^{-1} \oplus x_1(z)$ and x_{1_0} is the least significant bit of $x_1(z)$, then the term in brackets of Equation (14) can be replaced by

$$x_1(z) \odot z^{-1} \oplus x_1(z) = x_2(z) \oplus x_{1_0} \odot z^{-1}. \tag{15}$$

The right shift of $x_1(z)$ removes the least significant bit out of the integer number; but it must not be ignored and then

$$\begin{aligned} r(z) &= a(z) \oplus x_1(z) \odot z^{n-k} \oplus a(z) \odot z \odot x_2(z) \oplus a(z) \odot x_{1_0} \\ \text{with} \quad g(z) &= a(z) \odot z \\ \Rightarrow r(z) &= a(z)\{1 \oplus x_{1_0}\} \oplus x_1(z) \odot z^{n-k} \oplus g(z) \odot x_2(z) \\ \Rightarrow c(z) &= g(z) \odot x_2(z) \\ &= x_1(z) \odot z^{n-k} \oplus a(z) \odot \underbrace{\{1 \oplus x_{1_0}\}}_{=1, \text{ if } x_{1_0}=0} \oplus r(z). \end{aligned} \tag{16}$$

If the least significant bit $x_{1_0} = 0$, then $a(z)$ is not reduced and it inverts the remainder polynomial $r(z)$ compared to R. Table 3 shows an example of this effect of the isomorphism between linear block codes and arithmetic codes.

	X_1	R	$r(z)$
1	00001	11110	11110
2	00010	11101	00010
3	00011	11100	11100
4	00100	11011	00100
5	00101	11010	11010
6	00110	11001	00110
...	...	...	...

Table 3. Isomorphism between an arithmetic and linear (15, 10) code with $A = 31 = 2^5 - 1$ and $g(z) = a(z) \odot z = z^5 + z^4 + z^3 + z^2 + z$. Every second remainder R is inverted compared to $r(z)$. With both codes, the information is extended by $n - k = 5$ bits.

4 PERFORMANCE ESTIMATION

In the following part of the paper, an estimation for resource consumption and error performance of the given approach is presented.

4.1 Error Detection Capability

The error detection capability describes the performance of the code to detect bit flips. The presented linear block code that is generated by the polynomial of the form $g(z) = a(z) \odot z = \sum_{i=0}^{n-k-1} 1 \cdot z^{i+1}$ results in a code with a minimum hamming distance of only two. This means that only single bit errors can be detected but not corrected. The polynomial of the described form is indeed an irreducible polynomial, but not primitive [19]. Only primitive polynomials generate codes with a minimum hamming distance greater than two.

The range of the information X is limited to A. The required bits for the remainder R are the same as for the information part of the code. The code rate, which is a metric for the redundancy of a code [23], is

$$R = \frac{k}{n} = \frac{5}{10} = 0.5 \tag{17}$$

for the example in Table 3. This means that only half of every code word represents any information and the rest is redundancy. A comparison with rates of other codes shows that the efficiency of this code is not ideal. There are codes with a rate of 0.5 which have a minimum hamming distance of more than two, and an error correction is possible in this case.

4.2 Runtime Evaluation of Check

Verification of a code word is the evaluation of the remainder

$$R = C \mod A$$

which must be zero in the case of no errors. The division operation in a microcontroller consumes a lot of runtime. However, Rao [18] described a class of useful codes for error detection when selecting $A = 2^{n-k} - 1$. In this case, the evaluation of the remainder can be simplified and the modulo operation is substituted by a more simple addition.
Let $\vec{c} = \{c_{n-1}, c_{n-2}, \dots c_1, c_0\}$ be the binary representation of a code word. These bits are partitioned into l segments of length $n-k$. If these segments are $B_{l-1} \dots B_1, B_0$, then the modulo A of the sum of all segments B_j equals the modulo of the code word C with

$$C \mod A = \sum_{j=0}^{l-1} B_j \mod A. \tag{18}$$

The length of one segment B_j is $n-k$ bits. The sum is also restricted to $n-k$ bits. A possible overflow must be added to the sum.

Example 2. Let $C = 219$ be the corrupted systematic code word for $X = 6$ with $A = 2^5 - 1 = 31$. The binary representation for C is 0011011011_b and it follows

that there are two segments with $B_0 = 6 = 00110_b$ and $B_1 = 27 = 11011_b$. If b is the overflow bit of the sum of both segments $B_0 + B_1$ of size $n - k$ bits each, then $B_0 + B_1 + b = 2 = 219 \mod 31$. The remainder is not equal to 0 and an error is detected. Let C now be 217 without errors. Then $B_0 = 6 = 00110_b$ and $B_1 = 25 = 11001_b$. The sum is $B_0 + B_1 = 6 + 25 = 31$ and equals the generator A. In this case, the modulo is always zero and no errors have occurred. The evaluation for errors is reduced to the sum $\sum_{j=0}^{l-1} B_j$ and a following comparison with 0 or A.

5 CONCLUSION

We saw that the generator A is of the form $2^c - 1$, a simple transformation between linear and arithmetic codes and vice versa is possible (see Section 3). Secondly, the evaluation of error occurrence can be simplified to an addition instead of a division. This makes the evaluation for an error more efficient compared to the standard method with a modulo operation; but on the other hand, there are the disadvantages of the error correction probability and the code rate. The approach of a transformation between linear and arithmetic codes introduced here results in a code rate of 0.5 and a minimum hamming distance of only two. However, this type of code required for the isomorphism is not as efficient as other codes. An improved error performance is only possible when the same software is concurrently executed in two redundant coded channels. As described in [10], one single fault can also be corrected.

Furthermore, the transformation of codes can be useful in coded data processing. According to [11], the simplified processor model consists of several channels with different characteristics. BSC based operations like *MOV* or *XOR* match linear codes and have a better residual error probability than comparable arithmetic codes (see also [11]). In contrast, linear codes are not practical for arithmetic operations like the addition (*ADD*) because of the missing carry-bit propagation [24]. Thus, the presented code transformation has advantages for coded data processing if different underlying error models are considered.

REFERENCES

[1] Dodd, P. E.—Massengill, L. W.: Basic Mechanisms and Modeling of Single-Event Upsets in Digital Microelectronics. IEEE Transactions on Nuclear Science, Vol. 50, 2003, No. 3, pp. 583–602.

[2] Forin, P.: Vital Coded Microprocessor Principles and Application for Various Transit Systems. In IFA-GCCT 1989, pp. 79–84.

[3] Oh, N.—Mitra, S.—McCluskey, E. J.: ED4I: Error Detection by Diverse Data and Duplicated Instructions. IEEE Transactions On Computers, Vol. 51, 2002, pp. 180–199.

[4] Schiffel, U.—Schmitt, A.—Süsskraut, M.—Fetzer, C.: ANB- and ANBD-mem-encoding: Detecting Hardware Errors in Software. In Computer Safety, Reli-

ability, and Security, Volume 6351 of Lecture Notes in Computer Science, Springer 2010, pp. 169–182.

[5] Mottok, J.—Schiller, F.—Völkl, Th.—Zeitler, Th.: A Concept for a Safe Realization of a State Machine in Embedded Automotive Applications. In 26th SafeComp Conference 2007, pp. 283–288, ISBN 978-3-540-75100-7.

[6] Mottok, J.: Safely Embedded Software – A Safety Framework for C++. In Embedded Software Engineering Report, 2008.

[7] Meier, H.—Schiller, F.—Steindl, M.—Mottok, J.—Früchtl, M.: Diskussion des Einsatzes von Safely Embedded Software in FPGA-Architekturen. In Proceedings of the 2nd Embedded Software Engineering Congress 2009.

[8] Steindl, M.: Safely Embedded Software (SES) im Umfeld der Normen für Funktionale Sicherheit. Jahresrückblick 2009 des Bayerischen IT-Sicherheitsclusters, 2009.

[9] Brown, D. T.: Error Detecting and Error Correcting Binary Codes for Arithmetic Operations. In IRE Trans. Electron. Comput. 1960, pp. 333–337.

[10] Raab, P.—Krämer, S.—Mottok, J.—Meier, H.—Racek, S.: Safe Software Processing by Concurrent Execution in a Real-Time Operating System. In Proceedings of 16th International Conference on Applied Electronics 2011, pp. 315–319.

[11] Raab, P.—Krämer, S.—Mottok, J.: Cyclic Codes and Error Detection During Data Processing in Embedded Software Systems. In Proceedings of the 4th Embedded Software Engineering Congress 2011, pp. 577–590.

[12] Hamming, R. W.: Error Detecting and Error Correcting Codes. Bell System Technical Journal, No. AFCRC-TN-57-103, April 1950.

[13] Prange, E.: Cyclic Error-Correcting Codes in Two Symbols. Technical Report, Air Force Cambridge Research Center 1957.

[14] Bose, R. C.—Ray-Chaudhuri, D. K.: On a Class of Error Correcting Binary Group Codes. In Information and Control, Vol. 3, 1960, No. 1, pp. 68–79.

[15] Reed, I. S.—Solomon, G.: Polynomial Codes over Certain Finite Fields. Journal of the Society for Industrial and Applied Mathematics, Vol. 8, 1960, No. 2, pp. 300–304.

[16] ISO: ISO/DIS 26262-5: Road Vehicles – Functional Safety, Part 5: Product Development: Hardware Level. 2009.

[17] Rissanen, J. J.: Generalized Kraft Inequality and Arithmetic Coding. IBM Journal of Research and Development, Vol. 20, May 1976, No. 3, pp. 198–203.

[18] Rao, T. R. N.: Error Coding for Arithmetic Processors. Electrical Science Series, Academic Press, New York and London 1974.

[19] Peterson, W. W.—Brown, D. T.: Cyclic Codes for Error Detection. In Proceedings of the IRE, Vol. 49, 1961, No. 1, pp. 228–235.

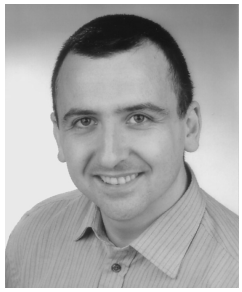
[20] Morelos-Zaragoza, R. H.: The Art of Error Correcting Coding. John Wiley & Sons 2006.

[21] Osmann, C.: Bewertung von Codierverfahren für Einen Störungssicheren Datentransfer (Evaluation of Error-Correcting Codes Used for a Reliable Data Transfer). Ph. D. thesis, Universität Duisburg-Essen, Campus Duisburg, 2001.

[22] Beutelspacher, A.: Lineare Algebra. Vieweg 1994.

[23] Bossert, M.: Kanalcodierung. Teubner 1998.

[24] Raab, P.—Krämer, S.—Mottok, J.: Reliability of Data Processing and Fault Compensation in Unreliable Arithmetic Processors. Submitted in Microprocessors and Microsystems: Embedded Hardware Design (MICPRO).



Peter Raab is experienced in the development of hardware and software for embedded systems for more than 10 years in several branches of industry. His responsibilities are the design, development and test of embedded software. From 1st of August 2010 till 31th of July 2013, he was research assistant at the Laboratory for Safe and Secure Systems (`www.las3.de`) for the research project S^{3}OP with the research interests in coded processing, real-time operating systems and reliability evaluations. In parallel, he graduated the Ph. D. study program at the University of West Bohemia Pilsen (Czech Republic). The thesis was "Model-Based Reliability Evaluation of Data Processing in HW-Fault Tolerant Processor Systems".

Stefan Krämer is research assistant at the Laboratory for Safe and Secure Systems (`www.las3.de`) working in the ZELOS^3 research project on reliable multicore scheduling. He is a Ph. D. student at the University of West Bohemia, Pilsen.

Jürgen Mottok is since more than 20 years involved in the development and assessment of safety critical automotive systems at SIEMENS VDO Automotive AG, now Continental Automotive GmbH. His responsibilities were architect and group leader for software-intensive safety critical engine control unit development. Also, he was responsible for safety assessments based on IEC 61508 and MISRA standards. As senior software and system architect he worked in the AUTOSAR consortium for architecture and safety issues, in C++ standardization gremium and in the MISRA C++ working group. Since 1st September 2004 he is Professor for computer science at Regensburg University of Applied Sciences. He is the Head of the Laboratory for Safe and Secure Systems (LaS^3, `www.las3.de`). He lectures on software engineering, programming languages, operating systems and functional safety. His research area includes automotive software engineering, real-time systems and functional safety, as well as automotive software engineering, real-time embedded systems and automotive safety. Currently he manages the research projects DynaS^3, VitaS^3, S^{3}OP, S^{3}EMO, AMALTHEA, SAGE, S^{3}CORE, ZeloS^3, FraLa and Evelin which are arranged with Research Master and Ph. D. students. The projects are performed in private public

partnership with Continental Corporation, former SIEMENS VDO Automotive AG, Timing Architects Embedded Systems GmbH and with iNTENCE Automotive Electronics GmbH. He is also involved as scientific consultant in AUTOSAR Architecture, Functional Safety and Timing Models. He is a member of the Managing Board of the Bavarian IT-Security and Safety Cluster, of the Advisory Board of the German Society of Computer Science in Eastern Bavaria, of the German Society of Physicists, of the Förderverein ADA Deutschland, of the Arbeitskreis Software-Qualität und – Fortbildung (ASQF) organisation, of Embedded4You, of the advisory board of the ASQF Safety group, head of the Executive Committee of Learning of Software Engineering – Registered Association, and organisator of the Workgroup Technical Didactics Software Engineering formed by Professors of Universities of Applied Sciences. In 2010 he became the laureate of the reward of excellent teaching appointed by the Bavarian State Ministry of Science, Research and Culture. In 2014 he was nominated for the "Lehre hoch n" initiative by "Studienstiftung des deutschen Volkes".

Vlastimil Vavřička received the master's degree in technical cybernetics in 1974 and Ph. D. degree in electronic and computer engineering in 1986 from College of Mechanical and Electrical Engineering in Pilsen. He is currently an Associate Professor in the Department of Computer Science and Engineering at the University of West Bohemia. His research interests include computer architectures, programmable logic and embedded systems design. He supervises several research projects supported by public funding in the field of embedded system design.