Reed-Muller Realization of X (mod P)

Danila A. Gorodecky
United Institute of Informatics Problems of
NAS of Belarus
Minsk, Belarus
danila.gorodecky@gmail.com

Abstract—This article provides a novel technique of X (mod P) realization. It is based on the Reed-Muller polynomial expansion. The advantage of the approach concludes in the capability to realize X (mod P) for an arbitrary P. The approach is competitive with the known realizations on the speed processing. Advantages and results of comparison with the known approaches for X [9:1] and P=7 is demonstrated.

Keywords—modular arithmetic, residue number system, X (mod P), Reed-Muller expansion

I. INTRODUCTION

The realization of the X (mod P) operation occupies a central place in cryptography; an efficiency of its realization in the residue number system (RNS) defines whether RNS will find wide implementation in practice or not.

There are two ways for hardware realization of X (mod P). The pipelining realization is implemented for transformation of a data flow to a sequence of residues. An example of a fast pipelining realization in cryptography has been proposed in [1]. Another way is based on the iteration process. An iteration produces the bits in decreasing significance with later iterations producing bits of lower significance. Variations of these techniques referring to RNS have been proposed in [2, 3]. The main goal of the realization X (mod P) is to achieve high-speed processing.

The first way is suitable for an arbitrary value of P, but the speed of the approach is limited by a block of pipeline, which includes three kind of successive operations (comparison, multiplexing, and subtraction). The second way is efficient just for some types of P (2^n, 2^n±1, 2^n±3 [2,3,4] and some variations of them [5], e.g. 2^n±2^{n/2}±1, 2^{2n+1}±1). The article proposes an approach for X (mod P) realization which is suitable and efficient for an arbitrary value of P, but it competitive with an iterative procedure.

Result of the calculation of X (mod P)=S is δ-bits binary vector, where δ=\log_2 P+1, and every digit of S=(S_0,...,S_{δ-1},...,S_1) is a Boolean function represented by Zhegalkin (or positive polarity Reed-Muller) form – polynomial XOR expansion with only incompetent variables or Zhegalkin expansion. The rest of the material describes a technique of generation of the polynomial extensions and hardware realizations of them.

A case for P=2^δ does not considered due to a simple way of realization of X (mod2^δ). In this way 
\(\left(x_0, x_{δ-1}, \ldots, x_1\right) (\text{mod} 2^δ) = (x_n, x_{n-1}, \ldots, x_1), \) i.e. 
\(X(x_0, x_{δ-1}, \ldots, x_1)\) and \(P=2^3\), then \((x_3, x_2, x_1)) (\text{mod} 8) = X .\)

II. X (MOD P) REALIZATION BY USING BOOLEAN FUNCTIONS

The idea of the approach is to consider a result of X (mod P)=S as the system of δ Boolean functions. Let’s define X=\{x_n, x_{n-1}, \ldots, x_1\}, \ P=\{p_0, p_{δ-1}, \ldots, p_1\}, and \ S=\{S_0, S_{δ-1}, \ldots, S_1\}. Hence a Boolean function \ S_i, \ i=1,δ, depends on n variables, i.e. \ S_i=S_i\left(x_n, x_{n-1}, \ldots, x_1\right).

For any other case For example, a system of functions for the case X (mod 5)=S, where X=\{x_5, x_4, x_3, x_2, x_1\}, takes the following form:
\[\begin{align*}
S_1 &= S_1(x_5, x_4, x_3, x_2, x_1); \\
S_2 &= S_2(x_5, x_4, x_3, x_2, x_1); \\
S_3 &= S_3(x_5, x_4, x_3, x_2, x_1). 
\end{align*}\]

An arbitrary Boolean function depends on n variables may be represented with the set of the truth numbers \ A(S_i) – numbers which correspond to the indexes of the truth table vector w(S_i). This set contains numbers corresponded to unities on function values. Let’s generate Boolean functions with the following way: a function \ S_i(x_n, x_{n-1}, \ldots, x_1)=1 if and only if \ X (mod P)=S(S_0, S_{δ-1}, \ldots, S_1=1,\ldots, S_1) (mod P). It is an equivalent for the set of the truth numbers \ A(S_i), when this set consists of numbers contained unity on i-th bits on mod P in the range from 0 to 2^n–1. For example, for \ X=\{x_5, x_2, x_1\}\) and \ P=3, the set of truth numbers for \ S_1 is equal \ A(S_1)=[1,4,7] and for \ S_2 is equal \ A(S_2)=[2,5].

There is a one to one correspondence between the set of the truth numbers \ A(S_i) and the truth vector w(S_i): the j-th entry of the set of truth numbers corresponds to the j-th unity of the truth vector. Thus \ A(S_1)=[1,4,7] is transformed into \ w(S_1)=[0,1,0,0,1,0,0,1], and \ A(S_2)=[2,5] is transformed into \ w(S_2)=[0,0,1,0,0,1,0,0]. Let’s recall that the truth vector w(S_i) is the binary vector whose entry corresponds to the term from
the full disjunctive normal form (FDNF) of the function $S_j$. FDNF is a disjunctive normal form with disjunctions which contained all variables of the function depends on.

The polynomial expansion of the function is the most efficient representation than others normal forms of Boolean functions for some criteria, e.g. because of a smaller number of terms and units in a circuit (in some cases is much smaller) [6].

As $i$ from the truth vector corresponds to the term from FDNF of the function $S_j$, as well as from the Zhegalkin spectrum (or Reed-Muller spectrum [7]) corresponds to the term from the Zhegalkin (Reed-Muller) expansion. This expansion is referred as $r(S_j)$. And the truth vector should be transformed to the Zhegalkin spectrum. This task may be solved with the number of methods, and to demonstrate the procedure of transformation we will use the combinatorial method [8]. The principle of the transformation of $w(S_j)$ to $r(S_j)$ (and backward) for an arbitrary Boolean function $S_j$ is represented with the following theorem.

**Theorem 1** [8]. The $i$-th entry $w_i$ of the truth vector $w(F)=[w_0,w_1,...,w_{2^n-1}]$ of the Boolean function $F$ is calculated with the following formula:

$$w_i = \begin{cases} 1, & \text{if} \left(\frac{i}{a_1}\right) \cdot \left(\frac{i}{a_2}\right) \cdot ... \cdot \left(\frac{i}{a_q}\right) \equiv 1 \pmod{2} \\ 0, & \text{otherwise} \end{cases}$$

where $a_i = a_1 \cdot a_2 \cdot ... \cdot a_q$, $\frac{i}{a_j} = 0$ for $i < a_j$ and

$$w = \left[\frac{0}{a_1}, \frac{1}{a_1}, ..., \frac{w_{2^n-1}}{a_1}\right]_{n-q}.\text{In other words, $q$ is the number of unities of the truth vector $w(F)=[w_0,w_1,...,w_{2^n-1}]$.}$$

It is helpful to use a consequence of the Lucas theorem [9] to transform $w(S_j)$ to $r(S_j)$.

**Theorem 2** [9].

$$\binom{n}{a} \equiv 1 \pmod{2} \iff \text{each bit of } a \text{ is no more than the same bit of } n.$$

Let’s demonstrate the implementation of theorems on the transformation of $w(S_j)=[0,1,0,1,0,1,0,1]$ to $r(S_j)$. According to *Theorem 1* $w_0=r_0=0$ and $w_1=r_1=1$, hence $r(S_j)=[0,1,0,...,r_j]$ and using *Theorem 2*

$$r_2 = \begin{pmatrix} 2 \pmod{2} \equiv 0 \\ \pmod{2} \equiv 0 \end{pmatrix} \equiv 0, \quad r_3 = \begin{pmatrix} 3 \pmod{2} \equiv 1 \\ \pmod{2} \equiv 1 \end{pmatrix} \equiv 1, \quad r_4 = \begin{pmatrix} 4 \pmod{2} \equiv 0 \\ \pmod{2} \equiv 0 \end{pmatrix} \equiv 0,$$

For $S_j = \begin{pmatrix} 5 \pmod{2} \equiv 1 \\ \pmod{2} \equiv 0 \end{pmatrix} \equiv 0, \quad r_5 = \begin{pmatrix} 6 \pmod{2} \equiv 1 \\ \pmod{2} \equiv 1 \end{pmatrix} \equiv 1, \quad r_7 = \begin{pmatrix} 7 \pmod{2} \equiv 1 \\ \pmod{2} \equiv 1 \end{pmatrix} \equiv 1.$

In the result $r(S_j)=[0,1,0,1,0,1,1,1].$

As the $q$-th unity of $r(S_j)=[0,1,0,1,0,1,1,1]$ correspond to the $q$-th term of the Zhegalkin polynomial of the function $S_j$, then

$S_j(x_1,x_2,x_3) = x_1 \oplus x_2 \oplus x_3 \oplus x_2 \oplus x_2 \oplus x_3.$

The same procedure is used to generate expansions for $S_2$ and $S_3$.

### III. Software Realization of X (mod P)

The generating of the converter for the calculation of $X \pmod{P}=S$ consists of four steps: calculating of the truth numbers $A(S_j)$ and the truth vector $w(S_j)$ of function $S_j$; transformation of $A(S_j)$ or $w(S_j)$ to $B(S_j)$ and $r(S_j)$ respectively; generating of a polynomial $P(S_j)$; modeling and synthesizing (with ISE Xilinx or LeonardoSpectrum) of the resulting polynomials.

The proposed approach is realized by four software blocks: 

- **Python** → **Java** → **Python** → **VHDL**. The scheme of the software realization in step-by-step manner is pictured at the Fig.

Inputs for the first step are values of $P$ and $X$. Python realization calculates the truth vector $w(S_j)$ and the truth numbers $A(S_j)$. The calculation for $X=(x_5,x_{14},...,x_1)$ and $P=(p_4,p_3,p_2,p_1)$ is produced in 0.5 second.

The second step is realized by **Java**-block. It transforms of $w(S_j)$ and $A(S_j)$ to $r(S_j)$ and $B(S_j)$ respectively. The process of calculating of $r(S_j)$ and $B(S_j)$ takes approximately 30 seconds for $X=(x_5,x_{14},...,x_1)$.

The third step is dedicated to generating of all polynomials $P(S_j)$ from $r(S_j)$ ($B(S_j)$), where $i=\overline{1,2}$ and $S=S_{5,5,5,5,5,...,S_j}$. The developed **Python** realization produces the step in 10 seconds.

The last fourth step generalizes previous steps. It joins **VHDL** descriptions of all polynomials $P(S_j)$, $P(S_2),...,P(S_{10})$ in one file. The resulting description is synthesized.

The next section represents the procedure of generating $X \pmod{P}=S$ in details.
\[ S_3 \text{. The truth numbers with the unity on the 3rd bit of numbers for modulo 5 is } A(S_3) = [4, 9, 14, 19, 24, 29] \text{ and the truth vector is } w(S_3) = (0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0). \]

**B. The second step: transformation \( A(S_i) \) to \( B(S_i) \) (or \( w(S_i) \) to \( r(S_i) \))**

As \( A(S_i) \) is analogue to \( w(S_i) \), and \( B(S_i) \) is analogue to \( r(S_i) \), thus we demonstrate this step producing on the transformation of the truth numbers \( A(S_i) \) to the Zhegalkin spectrum \( r(S_i) \).

Illustration of the transformation is unwieldy. Therefore, we will illustrate by transforming only \( A(S_3) \) to \( B(S_3) \). From the previous step \( A(S_3) = [4, 9, 14, 19, 24, 29] \) and according to 

\[ \text{Theorem 2} \]

we deduce that the truth numbers with the unity on the 2nd bit of numbers for modulo 5 is \( A(S_2) = [2, 3, 7, 8, 12, 13, 17, 18, 22, 23, 27, 28] \) and the truth vector is \( w(S_2) = (0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0). \]
The step aims to generate \(P(S_1)\), \(P(S_2)\), and \(P(S_3)\).

The fourth step: modeling and synthesizing of the resulting polynomials

This step dedicated to the modeling and synthesis of the VHDL polynomial have been got on the previous step. Synthesis is produced in with a computer-aided-design system (e.g. ISE Xilinx or LeonardoSpectrum).

V. DISCUSSION AND HARDWARE REALIZATION

This section provides results of comparison of area and the speed of processing between proposed and known approaches. The comparison produced for \(X = (x_9, x_8, ..., x_1)\) and \(P = 7\).

Modeling and synthesis is performed in ISE 13.1 and in LeonardoSpectrum2010a_7. The best results in the speed processing (in ns) and in the area (in LUTs) from ISE and Leonardo were chosen, and they are depicted in two tables.

Table 1 includes results of the synthesis of

- Pipelining approach [1] – PA. It is suitable for an arbitrary value of the modulo \(P\);
- Iterative approach [2,3] – IA. It is suitable for \(P = 2^n-1\);
- Polynomial expansion approach (proposed approach) – PEA. It is suitable for an arbitrary value of the modulo \(P\);
- Polynomial expansion approach (proposed approach) after BDD-optimization – PEA (BDD). The optimization [10] of number of terms for the proposed approach was applied. This optimization based on BDD-optimization. It is suitable for an arbitrary value of the modulo \(P\).

The synthesis was performed on

- FPGA Xilinx Spartan 3 XC3S1000 FG456 – Spartan_3;
- FPGA Xilinx Virtex 7 XC7V285t 3FFG1157 – Virtex_7;
- ASIC of the library POWER [11], which is used for design of ASIC circuits on hi-tech factory Integral (Minsk, Belarus) – POWER, where \(UNIT\) is an elementary measure of area.

The best indices are highlighted with bold.
VI. Conclusions and Further Work

There are two main advantages of the proposed approach for $X \pmod{P}$ realization: flexibility of $P$, because $P$ can be an arbitrary number, and no memory hardware realization, because it is used only XOR and AND operators.

The Table I provides the results of comparison for the case $P=2^3-1$. As we see, the proposed technique has the speed processing advantage over the known realizations.

Theoretically the proposed approach of hardware realization of $X \pmod{P}$ goals to calculate the operation for an arbitrary $X$ and $P$. But the bottleneck is in the realization $X \pmod{P}$ for a big range of $X$, e.g. if $X \pmod{30:1}$ and $P=7$ the realization process takes more than 24 hours. The process of the synthesis on FPGA takes most of this time.

In this way, the proposed approach has two directions for improving: area optimization and expanding the range of $P$.

Primarily further work will be directed to getting as short as possible polynomials and, as a consequence, reducing of the hardware complexity of the scheme of converter. The progress could be achieved at the expense of expanding of the range of $X$.

## REFERENCES

[1] J.T. Butler, T. Sasao, “Fast hardware computation of $x \pmod{z}$”, Proc. of the 25th IEEE International Parallel and Distributed Processing Symposium, Anchorage, pp. 289-292, AK, May, 2011.

[2] N.I. Cherviakov, P.A. Sahnik, A.V. Shaposhnikov, S.A. Riadnov, “Modular parallel computing systems of microprocessors systems”, Moscow, Fizmatlit, 2003, (in Russian)

[3] A. Omondi and B. Premkumar, “Residue number systems: Theory and implementation”, Imperial College Press, Singapore, 2007.

[4] K. Navi, A.S. Molahosseini, and M. Emamidoust, “How to Teach Residue Number System to Computer Scientists and Engineers”, IEEE Transactions on Education, V. 54, № 1, pp. 156–163, February, 2011.

[5] A.A. Zarandi, A.S. Molahosseini, and M. Hosseinzadeh, “Modern Residue Number System Moduli Sets: Efficiency vs. Complexity”, Neurocomputers, No. 9, pp. 7–12, 2014.

[6] A. Zakrevskij, Yu. Pottosin, and L. Cheremisinova, “Optimization in Boolean Space”, TUT Press, Tallinn, 2008.

[7] T. Sasao and J.T. Butler, “The eigenfunction of the Reed-Muller transformation”, Proc. of the Reed-Muller Workshop, Novoray, 2007.

[8] Danila A. Gorodecky, “Combinatorial Method of Polynomial Expansion of Symmetric Boolean Functions”, Proc. of the 11th International Workshop on Boolean Problems, Germany, pp. 211-218, September, 2014.

[9] A. Granville, “Arithmetic Properties of Binomial Coefficients I: Binomial coefficients modulo prime powers”, Canadian Mathematical Society Conf. Proc., Vol. 20, pp. 253-275, 1997.

[10] P.N. Bibilo and U.I. Romanov, “Logical design of discrete circuits with representation of production-frame model”, Minsk: Belarusskaia navuka, 2011.

[11] P.N. Bibilo and N.A. Kirienko, “Evaluation of energy consumption of CMOS circuits by switching their activity”, Mikroelektronika, No. 1, pp. 65-77, 2012.