

# Some transforms in cyclic redundancy check (CRC) computation

Zhanqi XU, Aijun WEN, and Zengji LIU

State Key Lab on Integrated Service Network

Xidian University

Xi'an, P. R. China

zqxu@mail.xidian.edu.cn, {ajwen, zjliu}@xidian.edu.cn

**Abstract**—Some transforms in cyclic redundancy check (CRC) computation, such as the multiplication of two messages' polynomials, shift, complement, and different initial (defaulted) remainder of a message, are studied. The relationships of the CRCs between the transformed message and the original one are also addressed from the mathematical point of view. Some examples are given to validate such transforms. We find that these transforms are useful in CRC computation implementations of both software and hardware.

**Keywords**—cyclic redundancy check (CRC); CRC computation; transform

## I. INTRODUCTION

Due to superior features in error correction and detection *Cyclic Redundancy Check* (CRC) has been widely employed to guarantee the data integrity in both transmission and storage, such as Ethernet link and file system. The traditional computation of CRC encoding and decoding has been studied extremely, which addresses various software and/or hardware implementations calculating a-bit or multi-bit (e.g. a-byte, and a-word) at a time [1,2,3]. Recent innovations include the increase of the computation speed [3,4], reduction of computation complexities [5], parallel computation by a single calculator[6], and unified mathematical expressions to open out the secret behind the parallel computation with multiple calculators or channels [7], and so on. To facilitate the CRC computation implementation to some extent, several new transforms, from the mathematical viewpoint, are presented in this paper, which include the multiplication of two messages, shift or complement of a message, and different (defaulted) initial CRC values of the message. We try to find out how the CRC of a specially transformed message is relating to that of its original message.

## II. TRANSFORMS

### A. Assumption and Notation

Let a bit sequence of K bits long, and  $\{m_i, i=K-1, K-2, \dots, 2, 1, 0\}$  be the message to be protected, then it can be denoted by a binary polynomial as follows

$$m(x) = \sum_{i=0}^{K-1} m_i x^i \quad (1)$$

Where the highest term of the polynomial is related to the leftmost bit of the bit sequence, i.e. the most significant bit (MSB).

Let  $r(x)$  be the polynomial of the check bits and  $R$  be the (maximum) number of coefficients in  $r(x)$ , then  $r(x)$  is given by

$$r(x) = \sum_{i=0}^{R-1} r_i x^i \quad (2)$$

According to the principles of CRC, we have

$$x^R m(x) = a(x)g(x) + r(x) \quad (3)$$

where  $g(x)$  is *generator polynomial*,  $a(x)$  and  $r(x)$  represent the *quotient* and *remainder* resulted from  $x^R m(x)$  divided by  $g(x)$ , respectively.

For the description simplification, we denote (1) by  $r(x) = R_{g(x)}[x^R m(x)]$ .

### B. Multiplication

Let the polynomials of two messages be  $m_1(x)$  and  $m_2(x)$ , and their remainders correspond to  $r_1(x)$  and  $r_2(x)$ , respectively. Therefore, the remainder of  $m_1(x)$  multiplied by  $m_2(x)$  is given by

$$R_{g(x)}[x^R m_1(x).m_2(x)] = R_{g(x)}[r_1(x).m_2(x)] \quad (4)$$

Or

$$R_{g(x)}[x^R m_1(x).m_2(x)] = R_{g(x)}[r_2(x).m_1(x)] \quad (5)$$

**Proof:** If we assume  $a_1(x)$  represents the quotient of  $x^R m_1(x)$  divided by  $g(x)$ , and then the left expression in (4) can be written as

$$\begin{aligned} & R_{g(x)}[x^R m_1(x).m_2(x)] \\ &= R_{g(x)}\{[a_1(x).g(x) + r_1(x)].m_2(x)\} \end{aligned}$$

$$=R_{g(x)}[r_1(x).m_2(x)]$$

So (4) holds true, it is also easy to verify (5).

### C. Shift

If we let a message in (1) shift  $W$  bits to the MSB direction, forming a new message denoted by  $m(x)'$ , and assume  $r(x)'$  represents the remainder of the division of  $x^R m(x)'$  by  $g(x)$ , the resulted polynomial  $m(x)'$  is given by

$$m(x)' = x^W m(x) \quad (6)$$

Then let us analyze how  $r(x)'$  relates to  $W$  and  $r(x)$ . Using (2) to (6) we have

$$\begin{aligned} r(x)' &= R_{g(x)}[x^R m(x)'] \\ &= R_{g(x)}[x^W [x^R m(x)]] \\ &= R_{g(x)}[x^W r(x)] \end{aligned} \quad (7)$$

Further we assume that  $W$  can be represented as follows

$$W = R * L + N \quad (8)$$

where  $L$  is a positive integer, and  $N$  is non-negative integer and less than  $R$ , i.e.  $0 \leq N < R$ .

In the case when  $N$  is equal to zero we can rewrite (7) as follows

$$\begin{aligned} r(x)' &= R_{g(x)}[x^{L*R} r(x)] \\ &= R_{g(x)}[x^R R_{g(x)}[x^R R_{g(x)}[... \\ &\quad x^R R_{g(x)}[x^R r(x)] \underbrace{...}_{L}]]] \end{aligned} \quad (9)$$

Here there are totally  $L$  square brackets in (9).

Let  $m_2(x)$  in (4) be  $x^R$ , we find that  $R_{g(x)}[x^R m_1(x). x^R]$  is equal to  $R_{g(x)}[r_1(x). x^R]$  which can further be represented by  $R_{g(x)}[x^R . R_{g(x)}[x^R m_1(x)]]$ . Equation (9) is derived by applying such recursive operations  $L$  times.

If  $N$  is not equal to zero, we can first calculate  $r(x)'$  by using (9), then denote the result as  $r(x)''$ , we get  $r(x)'$  from (7)

$$r(x)' = R_{g(x)}[x^N r(x)'] \quad (10)$$

How the  $r(x)'$  is calculated from (9)? We first construct a lookup table for regular  $R$ -bit parallel computation [1,2,7], take  $r(x)$  as input of that table to get  $r_1(x)$ , and put  $r_1(x)$  to that table to get  $r_2(x)$ , and so forth. Repeat this operation more times until  $r_L(x)$  is gotten from table output for  $r_{L-1}(x)$ .

### D. Complement

We assume that all bits in (1) are complemented, i.e. the bit equaling to 1 is changed to 0 and 0 is converted to 1,

respectively. How can we get the CRC for such a complemented message denoted by  $\overline{m(x)}$ ? From (1) the  $\overline{m(x)}$  is given by

$$\overline{m(x)} = \sum_{i=0}^{K-1} \overline{m_i} x^i \quad (11)$$

It appears difficult to directly compute the remainder  $r(x)'$  of  $\overline{m(x)}$  from (11). Fortunately, we note that adding  $m(x)$  to

$\overline{m(x)}$  is just equal to  $\sum_{i=0}^{K-1} x^i$ , so (11) can be expressed as

$$\overline{m(x)} = m(x) + \sum_{i=0}^{K-1} x^i \quad (12)$$

So

$$r(x)' = r(x) + R_{g(x)} \left[ \sum_{i=0}^{K-1} x^{i+R} \right] \quad (13)$$

This expression implies the CRC of a complemented message is just equal to EXOR (EXclusively OR) that of its original message with the remainder of  $R$ -bit 1's, the later is comparatively easy to get if the generator and the length of a message are given.

### E. Different CRC initials

The different CRC applications may have different CRC (defaulted) initials, such as 0's in CRC-16 SDLC (Synchronous Data Link Control), and 1's in early Ethernet with CRC-16.

If the initial CRC values for  $m(x)$  are  $R$ -bit 0's, let us see another message,  $m(x)'$ , which is the same as  $m(x)$  except that initial remainder for  $m(x)'$  is assumed to be  $D(x) = \sum_{i=0}^{R-1} d_i x^i$ ,

usually all  $d_i$  here being 1. In terms of the CRC computation for  $m(x)'$  and its corresponding remainder are given by

$$m(x)' = m(x) + x^{K-R} D(x) \quad (14)$$

$$r(x)' = r(x) + R_{g(x)}[x^K D(x)] \quad (15)$$

## III. EXAMPLES AND VALIATIONS

Let us take an example to validate the properties described in section II. We assume the message to be protected is [0,1,2, ..., 23], 24 bytes long, with the generator polynomial  $g(x) = x^{16} + x^{12} + x^5 + 1$ . Two CRC codes, 7EA9h (hexadecimal) and 1D8Bh, are obtained under two initial check bits, all 0's and 1's, respectively.

First we complement the message  $m(x)$  and get a new one, [255, 254, ..., 232]. Assuming a zero initial check bits, it is not difficult to get the remainder of the complemented one, B436h, and the result of  $R_{g(x)} \left[ \sum_{i=0}^{K-1} x^{i+R} \right]$  in (13) will be CA9Fh. The result of EXORing 7EA9h with CA9Fh, B436h, suggests the correctness of (13).

If the  $K$  is set to  $24 \times 8$ , the remainder,  $R_{g(x)}[x^K D(x)]$  in (15), is 6322h when sixteen terms in  $D(x)$  are all set to 1's. The EXOR of 6322h and 7EA9h, 1D8Bh, is exactly the same as what we computed directly.

For the shift of a message, the message is first segmented to two parts,  $[0,1,2\dots15]$  and  $[16,17\dots23]$ , denoted by  $m_1(x)$  to  $m_2(x)$ , respectively. If a zero initial check-bit is assumed, it is easily to get the remainder of  $m_2(x)$ , 5B58h. With the initial check bits of sixteen 1's, we can get that the remainder for  $m_1(x)$  will be 0BAEh. We notice that the  $m(x)$  can be expressed by  $x^{64} m_1(x) + m_2(x)$ . Putting the remainder of the first segment, 0BAEh, to  $r(x)$ , we calculate  $R_{g(x)}[x^R r(x)]$  one time and  $R_{g(x)}[x^R \cdot]$  three times as described above since  $L$  is equal to 4 according to (8), and get that the remainder for  $x^{64} m_1(x)$  is 4D63h. The result of EXORing 4D63h with 5B58h, which is the same as stated at the beginning of this paragraph, i.e., 1D8Bh, demonstrated the validity of (13).

More generally, if a message is divided into more than one segments, say  $m_1(x)$  through  $m_K(x)$ , we conclude initial remainder of the first segment (i.e., the most significant part) must be the same as the that of whole message while each initial remainder of the rest segments must be R-bit 0's. A detailed example could be found in [7].

#### IV. CONCLUSIONS

We have shown some transforms and how these transforms are used in CRC computation. These transforms could facilitate the CRC computation implementations in both software and hardware. For example, for the further reduction of CRC computation time at as high as multi-ten Gigabit per second, a

message can be segmented a lot of parts, and the remainder of each part could be computed dependently, we can get the CRC of the original message from all CRCs of the segmented parts. This resulted in multiple channels parallel CRC computation algorithm, in which multiple calculators, say  $M$ , run at same time, thus making the computation speed nearly  $M$  times faster comparing to one channel.

#### ACKNOWLEDGMENT

Grateful acknowledgment is made to Wuhan Post & Telecomm Research Institute, China, for the support in part.

#### REFERENCES

- [1] Tenkasi V. Ramabadran, Sunil S. Gaitonde, "A tutorial on CRC computations", IEEE Micro, Vol.8, No.4, pp.62~75, August, 1988
- [2] A. Perez, "Byte-wise CRC calculations", IEEE Micro, Vol.3, No.3, pp.40~50, June, 1983
- [3] Jesper Birch "A programmable 800 Mbit/s CRC check/generator unit for LANs and MANs", Computer networks and ISDN systems, Vol.24, pp.109-118, April, 1992
- [4] Derby, Jeff H, "High-speed CRC computation using state-space transformations", IEEE Global Telecommunications Conference GLOBECOM'01, San Antonio, TX, United States, V1, pp.166-170, 2001
- [5] R.J. Glaise, "A two-step computation of cyclic redundancy code CRC-32 for ATM networks", IBM Journal Research Development, Vol.41, No.6, pp.705-709, Nov., 1997
- [6] Xu Zhanqi, Yi Kechu, Liu Zengji, A universal algorithm for parallel crc computation and its implementation, Journal of Electronics (China), 2006, Vol.23, No.4, pp528-531
- [7] XU Zhan-qi, PEI Chang-xing, DONG Huai-nan, Generalized CRC Computation Algorithm with Multiple Channels and Its Implementation, Journal of Nanjing University of Posts and Telecommunications (Natural Science), Vol.28, No.2, pp 53-57, Apr. 2008