# On the computation of cyclic redundancy checks by program

P. L. Higginson and P. T. Kirstein

*University of London, Institute of Computer Science, 44 Gordon Square, London WC1H OPD*

**A method of computing cyclic redundancy checks by program is described which is substantially quicker than methods published previously. The method is particularly convenient for the generating polynomial appropriate to Binary Synchronous Communication in the form used with IBM 360/370 computers. The method is described in detail, and a comparison of program times is given by different methods.**

(Received March 1972)

## 1. Introduction

Cyclic Redundancy Checks (CRC) have been used to provide error detection on blocks of data for many years. The theory of the codes used has been developed in many papers (e.g. Peterson (1961)). Many of the publications describe hardware for generating and decoding the CRC sum bytes on a serial bit by bit basis. The advent of low cost computers as part of remote entry stations and as intelligent terminals make it important to provide minimum cost data transmission facilities. In these systems it is often necessary to generate the CRC bytes by software.

For software generation of CRC bytes, the algorithms which operate on a bit at a time are very inefficient. For this reason, there has been a recent development of algorithms which work on a byte by byte basis (Boudreau and Steen, 1971). These algorithms are a considerable improvement on those which work on a bit by bit basis, but either require substantial amounts of core, or are still somewhat slow. A typical calculation based on two bytes at a time for the PDP-9 with the specific CRC bytes required in binary synchronous communication with an IBM computer takes 55 $\mu$s per pair of 8 bit bytes. The main reason for this long time is the need to ascertain the parity of the pair of bytes. It is shown in this paper that it is possible to split up the computation into two parts. One part must be done with each byte pair, but does not require computation of parity. The second part operates on the parity of the whole message. This form of computation is much faster in any machine which does not have hardware for generating the parity of a byte or word.

In Section 2 the binary and polynomial notation is described briefly. A fuller description is given by Peterson (1961). The general formulae for generating the cyclic redundancy check bytes are given in Section 3. There it is shown how the check sum changes from one bit to the next. This form of the formulae is rather slow, and in practice the data has a certain number of 8 bit bytes. Since for IBM BSC purposes the residue is based on a 16 bit counter, and most machines at the moment have 16 bits or more, we present relations in Section 4 based on accumulating pairs of 8 bit bytes or units of 16 bits of data. The formulae derived would be fairly fast to evaluate if the parity of the message was known or a single instruction existed to obtain it; if this quantity is not fast to obtain, an alternate way of expressing the formulae is developed in Section 5 which is much faster. Finally, in Section 6, we extend the method of Section 5 for the case when the computer used has an 8 bit word length. In this case formulae based on accumulating single bytes are required.

## 2. Binary arithmetic and our notation

In all our arithmetic with binary numbers for the cyclic redundancy check we use a polynomial notation

$$F(x) = f_n + f_{n-1}\,x \ldots f_o\,x^n \qquad (1)$$

where the coefficients $f_i$ can be only 0 or 1. The polynomial $F$ will be represented in a computer by a bit stream in which the $i$th bit is $f_i$. Thus multiplying by $x^j$ is shifting to the left by $j$ bits. Addition of two polynomials is the *Exclusive Or* of the corresponding coefficients. In this paper all variables in capitals denote polynomials, while lower case letters denote scalars or coefficients of polynomials.

In the cyclic redundancy check we will operate on an accumulator with $n$ positions (in practice $n = 16$). In addition we assume a link bit for convenience; this is the most significant bit or coefficient of $x^n$. Any operation leading to a polynomial greater than of order $n$ can be ignored; it is shifted beyond our counter and lost. It is the aim of this paper to find the $n$ bit redundancy check sum in the most expeditious way.

## 3. Single bit cyclic redundancy check

The method of generating a cyclic redundancy check sum is to commence with a generating polynomial

$$G(x) = x^n + g_1\,x^{n-1} \ldots + g_{n-1}\,x + g_n \qquad (2)$$

and to take a Message polynomial $M_i$ of the form, after $i$ bits have been sent

$$M_i(X) = (m_o\,x^{i-1} + m_1\,x^{i-2} \ldots + m_{i-1})\,x^n \qquad (3)$$

The extra multiplication by $x^n$ denotes an initial shift of $n$ bits.

Then if $Q_i(x)$ and $R_i(x)$ are the quotient and remainder of dividing $M_i$ by $G(x)$, then

$$M_i(x) + R_i(x) = G(x)\,Q_i(x) \qquad (4)$$

$R_i(x)$ is denoted as the *Residue* of $M_i$ with respect to $G$.

We will now relate $M_i(x)$ and $R_i(x)$ to $M_{i+1}(x)$ and $R_{i+1}(x)$. First from the definition of $M_i$,

$$M_{i+1}(x) = x\,M_i(x) + m_i\,x^n \qquad (5)$$

Let us define

$$R_i(x) = r_{i,0}\,x^{n-1} + \ldots + r_{i,n-1} \qquad (6)$$

Then comparing Equations (4) and (5) we see that

$$\begin{aligned} G(x)\,Q_{i+1}(x) &= M_{i+1}(x) + R_{i+1}(x) \\ &= x\,Q_i(x)\,G(x) + x\,R_i(x) + m_i\,x^n + R_{i+1}(x) \end{aligned}$$

Remembering that $R_i(x)$ is of degree $(n - 1)$ or less, we see that

$$R_{i+1}(x) = x\,R_i(x) + (r_{i,o} + m_i)\,G(x) + m_i\,x^n \qquad (7)$$

In Equation (7) the coefficient of $x^n$ vanishes, so that the $R_{i+1}$ is the correct one. Thus to get $R_{i+1}$ from $R_i$ in a machine with a link $L$ and accumulator of $n$ bits containing $R_i$ we do as follows

(*a*) Rotate left one bit
(*b*) *XOR* $m_i$ in to Link
(*c*) Add $L.G$

This procedure is simple but has to be done each bit. In the PDP-9 this takes about 6 $\mu$s/bit. For this reason a faster method is desirable.

## 4. Binary synchronous redundancy check on a two byte basis

Equation (7) is as far as we can go in general. It is instructive to consider the particular CRC used in IBM binary synchronous communications. Here $G(x)$ has the form

$$G(x) = x^{16} + x^{15} + x^2 + 1 \tag{8}$$

Moreover the number of bits used is *always* a multiple of 8. We will assume, in the rest of the next two sections, that the number of bytes sent is $2n$, and always consider pairs of bytes. This will require an extra operation at the end if an odd number of bytes is sent.

Because we are using pairs of bytes, a new notation will be used. Instead of the message polynomial of Equation (3), we will assume the $i$th byte pair has the form

$$M_i(x) = m_{i,o}\, x^{15} + m_{i,1}\, x^{14} + \ldots m_{i,15} \tag{9}$$

and the Residue after the $i$th byte pair is $R_i$ where

$$R_i(x) = r_{i,o}\, x^{15} + \ldots r_{i,15} \tag{10}$$

From repeated application of Equation (7), it is found that if we denote $R_{i+1}$ as the polynomial resulting from sixteen applications of the formula, then

$$R_{i+1} = (M_i + R_i)D + (r_{i,o} + m_{i,o})A + (r_{i,1} + m_{i,1})B + p_i C \tag{11}$$

where $R_i$ is the residue before adding the $i$th byte pair, $M_i$ is the contents of the $i$th byte pair, and the coefficients are given by the expressions

$$\begin{aligned} M_i &= m_{i,o}\, x^{15} + \ldots + m_{i,15} \\ A &= x^3 + x \\ B &= x^{15} + x^2 + 1 \\ C &= x^{15} + x + 1 \\ D &= x^2 + x \\ p_i &= \sum_{j=0}^{15} (r_{i,j} + m_{i,j}) \end{aligned} \tag{12}$$

Equations (11) and (12) are much faster already to program than repeated application of Equation (7). It can be shown that usually by far the longest part of a program to compute $R_{i+1}$ from $R_i$ is the computation of the parity $p_i$. If $p_i$ is given by hardware, or if a simple parity was provided with each byte, then the computation is greatly speeded up. A more elegant and general version of Equation (11) is given in Boudreau and Steen (1971). The present form is more convenient, however, for our purposes.

## 5. Computation of CRC by separating off parity calculation

It can be shown that the first three terms of Equation (11) can be performed fairly quickly. The last term is usually slow, because $p_i$ requires an operation to be done on every bit—taking 25 $\mu$s on a PDP-9 for 16 bits. We therefore will try to rewrite Equation (11) in terms of another variable $W_i$, where $W_i$ is related to $R_i$, but its recurrence relation does not contain $p_i$.

First, we will relate $p_i$ to the sum of all previous message bits, i.e. the total parity of the message. We can show from the definition of $p_i$, and from summing all the terms of $R_{i+1}$ in Equation (11), that

$$p_{i+1} = \sum_{j=0}^{15} (r_{i+1,j} + m_{i+1,j}) = p_i + \sum_{j=0}^{15} m_{i+1,j} \tag{13}$$

But $p_o = 0$, hence $p_i$ is the parity of the first $i$ byte pairs of the message.

Next let us define $W_i$ by the relation

$$W_i = R_i + C p_{i-1} \tag{14}$$

Then substituting Equation (14) into Equation (11) we see that

$$\begin{aligned} W_{i+1} &= R_{i+1} + C p_i \\ &= D(W_i + M_i + C p_{i-1}) + (w_{i,o} + m_{i,o} + p_{i-1}) A \\ &\quad + (w_{i,1} + m_{i,1}) B + p_i C + C p_i \\ &= (M_i + W_i)D + (m_{i,o} + w_{i,0})A + (m_{i,1} + w_{i,1})B \\ &\quad + (x^{16} + x^{17})p_{i-1} \end{aligned} \tag{15}$$

Now if $W_{i+1}$ is computed from $W_i$ by truncating coefficients of $x^{16}$ or greater, Equation (15) can be written

$$W_{i+1} \approx (M_i + W_i)D + (m_{i,o} + w_{i,1})A + (m_{i,1} + w_{i,1})B \tag{16}$$

where $\approx$ is written because the terms in $x^{16}$ and higher are truncated. Because the recurrence relation Equation (16) does not contain $p_i$, it is much faster to calculate on most computers than that of Equation (11).

Now $p_i$, is, from the discussion after Equation (13), the total parity of the message. Hence to evaluate $p_i$ after $i$ byte pairs

$$\begin{aligned} p_i &= \sum_{k=1}^{i} \sum_{j=0}^{15} m_{k,j} \\ &= \sum_{j=0}^{15} \sum_{k=1}^{i} m_{k,j} \end{aligned} \tag{17}$$

To get $p_i$ we may do $\sum_i M_i$ on a word basis and then do $\sum_{j=0}^{15} (\sum_i M_i)_j$. This is much faster than any other way of doing it.

Thus to determine $R_n$ for a set of data, we first determine $W_n$ by repeated application of Equation (16); then $p_n$ is found from Equation (17); finally $R_n$ is found from Equation (14). Timing figures for the PDP-9 of the three algorithms by bit, by character pair and by message as developed in Sections 3, 4 and 5 are given in **Table 1.** In the same table are shown for comparison purposes the first two methods of CRC generation of cyclic redundancy checking by program (Boudreau and Steen, 1971). The third method of that reference is not considered, because we do not have hardware for generating the parity of a byte or word. The figures in the third column of Table 1 give the words of core required to generate the CRC for an even number of bytes. The second and third methods deal with byte pairs, and have additional core requirements to deal with an odd number of bytes; these requirements are indicated in the last column. The time given for the third method excludes an overhead of 37 $\mu$s/block, which is negligible for the block lengths usually transmitted (>100 bytes).



**Table 1 Performance of different methods of CRC generation on the PDP 9**

| METHOD | MESSAGE TIME PER BYTE ($\mu$s) | WORDS OF CORE USED | EXTRA WORDS OF CORE USED FOR ODD NO. OF BYTES |
|---|---|---|---|
| Hardware Simulation (Equation 7) | 55 | 18 | 4 |
| Byte Pair Equations (Equation 13) | 24 | 34 | 14 |
| Message Equations (Section 5) | 13 | 43 | 14 |
| Method by 256 word Look-Up Table of Boudreau and Steen (1971) | 22 | 269 | — |
| Method by double 32 word Look-Up Table of Boudreau and Steen (1971) | 36 | 51 | — |

**Table 1** shows that the core requirements of the method of this section are not large, and the time taken is only 60 per cent of the fastest method of Boudreau and Steen (1971), and four times faster than simple simulation of the BSC hardware. Programs of the different methods are given in the Appendix for comparison purposes.

To compute the CRC for an odd number of characters requires a special operation on the last byte; the core requirements for this are shown in the last column of Table 1. In the second and third methods only this operation takes double the time of other bytes; this time penalty is negligible for blocks of >100 bytes.

**6. Extensions of method to single bytes**

The reason for the great difference in speed between the second and the third algorithm is that one application of Equation (16) requires only 13 $\mu$s, while the parity of a pair of bytes takes 25 $\mu$s. The speed of application of Equation (16) is due to the specific generating polynomial used, and would not apply in general.

A similar, but less impressive gain in speed would arise by deriving a version of Equation (11) based on eight applications of Equation (7). The appropriate equations are best expressed in terms of half words or bytes.

If

$$R_i = x^8 R_{iH} + R_{iL} \text{ etc.} \tag{18}$$

and $R_{iH}$, $R_{iL}$, the Message $M_i$ etc. are bytes, then the equivalent of Equation (11) is

$$R_{i+1,H}\, x^8 + R_{i+1,L} = R_{i,L}\, x^8 + \{[(x^2 + x)(R_{i,H} + M_i)] + p_i C\} \tag{19}$$

where again

$$C = x^{15} + x + 1 \tag{20}$$

and

$$p_i = \sum_{j=0}^{7} (r_{i,jH} + m_{i,j}) \tag{21}$$

The portion of Equation (19) inside { } depends only on $(R_{i,H} + M_i)$. This is the reason that that part of the equation can be obtained from a 256 word table look up, and is the basis of the first method of Boudreau and Steen (1971).

The $p_i$ of Equation (21) is not the parity of the message, and relations separating out the parity term can be derived only by considering pairs of bytes. In this way the method becomes identical to that of Section 5. Equations (19)-(21) are useful, however, to deal with the extra byte if the message contains an odd number, and the method of Section 5 is used.

# Appendix

**CRC generation routines for the PDP-9**

In the main text algorithms for five methods of CRC generation have been mentioned. Three are hardware simulation (Section 3), generation by character pair (Section 4) and generation by message (Section 5). The other two are methods using Look-Up tables derived in Boudreau and Steen (1971). For comparison purposes, we present here the PDP-9 programs for the basic routines for each of these methods. The instruction code of the PDP-9 is given in the PDP-9 users Handbook (1970), and will not be discussed here. It is an 18 bit machine with a one $\mu$s cycle time; thus the time it uses in $\mu$s also gives the number of core cycle times. It is assumed in each case that the byte pairs are packed into bits 2-10 and 11-17 of the PDP word, bit 17 is sent first, bit 2 last and the numbers in the routines are all octal.

Incidentally the first three methods gain slightly from having the data packed into buffers or available in byte pairs, while it is preferable for the last two methods to have the individual bytes available.

1. *Hardware simulation*

Subroutine for one byte

```
CRC  Ø                      /Enter with byte in ACC
     XOR      CRCA
     DAC      CRCA
     LAW      -1Ø
     DAC      COUNT
     LAC      CRCA
     RCR
     SZL!CLL
     XOR      (120001
     ISZ      COUNT
     JMP      .-4
     DAC      CRCA
     JMP*     CRC
```

core used = 16 words (13 program, 2 temp, 1 constant)
time taken = 60 $\mu$s per byte.

To compute the CRC for a pair of bytes the count is set to −20; in this way the time taken can be reduced to 55 $\mu$s/byte at the expense of 6 more words.

2. *Byte pair equation*

Subroutine for a byte pair

```
CRC  Ø
     XOR      CRCA
     DAC      CRCA
     RCR
     XOR      CRCA
     RCR!SZL            /ACC = (M_i + R_i)D
     XOR      (170001   /      + (r_io + m_io) (A + B)
     SZL
     XOR      (120001   /      + (r_io + m_io + r_i1
                                        + m_i1)B
     DAC      TEMP
     LAC      CRCA      /Now parity
     LRS      1Ø        }
     XOR      CRCA      }
     DAC      CRCA      }
     RTR                }
     RTR                }
     XOR      CRCA      } Generate Parity of ACC with
     DAC      CRCA      } answer in LINK
     RTR                }
     XOR      CRCA      }
     DAC      CRCA      }
     RAR                }
     XOR      CRCA      }
     RAR                }
     SZL!CLA
     LAC      (140001   /P_i C
     XOR      TEMP
     DAC      CRCA
     JMP*     CRC
```

core used = 34 words
time taken = 48 $\mu$s per byte pair.

Note that a Parity Generation Instruction (of the ACC) would replace the 13 words of code indicated (taking 25 $\mu$s) and reduce the CRC of a message to 14 $\mu$s per byte.

3. *Message equation*

Assuming $T1$ is set up to the buffer address $T2$ = −number of complete words, and given $T4 = T1$, $T5 = T2$ for the second loop.

```
        CLA
LOOP1   XOR*    T1
        DAC     T7
        RCR
        XOR     T7
        RCR!SZL          /(Mi + Ri)D
        XOR     (170001  /Bit 17 into places 2, 3, 4, 5, 17
        SZL
        XOR     (120001  /Bits 16.17 into places 2, 4, 17
        ISZ     T1
        ISZ     T2
        JMP     LOOP1
        DAC     T1       /save
        CLA
LOOP2   XOR*    T4       /Work out word parity
        ISZ     T4
        ISZ     T5
        JMP     LOOP2
        DAC     T2
        ALS     1Ø
        XOR     T2       /Then 8 bit parity byte
        DAC     T2
        RTL
        RTL
        XOR     T2       /Reduce 8 bits to 4
        DAC     T2
        RTL
        XOR     T2       /To 2
        DAC     T2
        RAL
        XOR     T2       /To 1
        RTL
        SPA!CLA
        LAC     (140001
        XOR     T1
        DAC     T1
```

core used = 43 words
main loop time = 25 μs per byte pair.

Note that the 'once-only' parts take about 37 μs but as most blocks are at least 100 bytes long this is not important.

4. *Single* 256 *word look-up*
Subroutine for one byte

```
CRC     Ø
        XOR     CRCA
        AND     (377
        TAD     (XOR TABLE
        DAC     .+3
        LAC     CRCA
        LRSS    1Ø
        XX
        DAC     CRCA
        JMP*    CRC
```

core used = 269 words
time taken = 22 μs per byte.

5. *Double* 32 *word look-up*
This is similar to 4 but uses two 16 word tables. Subroutine for one byte

```
CRC     Ø
        XOR        CRCA
        LRS        4
        AND        (17
        TAD        (XOR TABLE 1
        DAC        .+6
        LLS!CLEA   4              /CLEA = 1000
        TAD        (XOR TABLE 2
        DAC        .+4
        LAC        CRCA
        LRSS       1Ø
        XX
        XX
        DAC        CRCA
        JMP*       CRC
```

core used = 51 words
time taken = 36 μs per byte.

**References**


PETERSON, W. W. (1961). *Error Correcting Codes*. The MIT Press, Cambridge, Mass.
BOUDREAU, P. E. and STEEN, R. F. (1971). Cyclic Redundancy Checking by Program, *AFIPS Conference Proceedings*, Vol. 39, pp. 11-15.
*PDP* 9 *User's Handbook*, Digital Equipment Corporation, Maynard, Mass., 1970.