Preprint: The International Conference on Dependable Systems and Networks, DSN-2004.

Cyclic Redundancy Code (CRC) Polynomial Selection For Embedded Networks

Philip Koopman
ECE Department & ICES
Carnegie Mellon University
Pittsburgh, PA, USA
koopman@cmu.edu

Tridib Chakravarty
Pittsburgh, PA, USA
tridib@alumni.carnegiemellon.edu

Abstract

Cyclic Redundancy Codes (CRCs) provide a first line of defense against data corruption in many networks. Unfortunately, many commonly used CRC polynomials provide significantly less error detection capability than they might. An exhaustive exploration reveals that most previously published CRC polynomials are either inferior to alternatives or are only good choices for particular message lengths. Unfortunately these shortcomings and limitations often seem to be overlooked. This paper describes a polynomial selection process for embedded network applications and proposes a set of good general-purpose polynomials. A set of 35 new polynomials in addition to 13 previously published polynomials provides good performance for 3- to 16-bit CRCs for data word lengths up to 2048 bits.

1. Introduction

Cyclic Redundancy Codes (CRCs) are commonly used for error detection in embedded networks and other applications. But many applications employ CRCs that provide far less error detection capability than they might achieve for a given number of CRC bits. This is largely because there is little published guidance and less quantitative data upon which to base tradeoff decisions. To help improve this situation, this paper proposes "good" general purpose CRCs for error detection applications that encompass many current and future embedded network protocols and other uses having data words up to 2048 bits in length.

While various CRC designs can be found in standards and folklore, most of them are far from optimal for the short messages found in embedded networks. For embedded networks, the property of interest is usually the Hamming Distance (HD), which is the minimum possible number of bit inversions that must be injected into a message to create an error that is undetectable by that message's CRC-based Frame Check Sequence. For example, if a CRC polynomial has HD=6 for a given network, that means there are no possible combinations of 1-, 2-, 3-, 4-, nor 5-bit errors (where a bit error is an inversion of a bit value) that can result in an undetected error, but there is at least one combination of 6 bits that, when corrupted as a set within a message, is undetectable by that CRC. An additional property of interest is burst error detection capability, but all codes we will discuss can detect burst errors up to the size of the CRC width. Other possible evaluation criteria exist such as unidirectional bit error detection (which depends on data values) and high-noise detection. Unfortunately there does not seem to be any authoritative characterization of faults in embedded networks. Our interactions with industry indicate that HD for random independent errors on a binary symmetric channel is usually the primary factor considered in embedded network CRC design, and thus is the metric we use in this paper.

After a series of protocol evaluations for industry applications in which the question arose as to whether it would be possible to achieve a given Hamming Distance (HD) with a given CRC size, we decided to explore the design space of CRC size, message length, and attainable Hamming Distance. The results indicate that there are significant opportunities for improving CRC effectiveness because some commonly used CRCs have poor performance. Moreover, many sources used in industrial practice teach engineers to select a polynomial without taking into account the length of the data being error checked, which ignores an important engineering tradeoff. And, even if engineers want to make detailed design tradeoffs, tools and data tables on polynomial performance are scarce and often difficult to apply.

This paper presents a small set of polynomials that provides good overall performance while including message length as a key design parameter. After discussing background and previous work, a methodology for defining "good" CRC designs is proposed, and the results of applying that methodology are presented. Comparisons of published CRCs to the proposed designs reveal both strengths and serious weaknesses in the existing state of practice.

2. Background

A CRC can be thought of as a (non-secure) digest function for a data word that can be used to detect data corruption. Mathematically, a CRC can be described as treating a

binary data word as a polynomial over GF(2) (*i.e.*, with each polynomial coefficient being zero or one) and performing polynomial division by a *generator polynomial* G(x), which is commonly called a CRC polynomial. (CRC polynomials are also known as feedback polynomials, in reference to the feedback taps of hardware-based shift register implementations.) The remainder of that division operation provides an error detection value that is sent as a Frame Check Sequence (FCS) within a network message or stored as a data integrity check. Whether implemented in hardware or software, the CRC computation takes the form of a bitwise convolution of a data word against a binary version of the CRC polynomial. The data word size is the data protected by the CRC but not including the CRC itself. [Peterson72] and [Lin83] are among the commonly cited standard reference works for CRCs. [Wells99] provides a discussion for non-specialists.

Table 1. Example Hamming weights for data word size 48 bits.

CRC Size (bits)	CRC Polynomial	HD	Hamming weights for number of bits corrupted:					
			1 bit	2 bits	3 bits	4 bits	5 bits	6 bits
16	CCITT-16 0x8810	4	0	0	0	**84**	**0**	**2 430**
16	[Baicheva00] 0xC86C	6	0	0	0	0	0	**2 191**
15	CAN 0x62CC	6	0	0	0	0	0	**4 314**
12	CRC-12 0xC07	4	0	0	0	**575**	**0**	**28809**
12	0x8F8	5	0	0	0	0	**1 452**	**13 258**
8	DARC-8 0x9C	2	0	**66**	**0**	**2 039**	**13 122**	**124 248**
8	CRC-8 0xEA	4	0	0	0	**2 984**	**0**	**253 084**
7	CRC-7 0x48	3	0	0	**216**	**2 690**	**27 051**	**226 856**
7	0x5B	4	0	0	0	**5 589**	**0**	**451 125**

Error detection is performed by comparing an FCS computed on data against an FCS value originally computed and either sent or stored with the original data. An error is declared to have occurred if the stored FCS and computed FCS values are not equal. However, as with all digital signature schemes, there is a small, but finite, probability that a data corruption that inverts a sufficient number of bits in just the right pattern will occur and lead to an undetectable error. The minimum number of bit inversions required to achieve such undetected errors (*i.e.*, the HD value) is a central issue in the design of CRC polynomials.

Using the right polynomial is central to CRC-based error detection schemes. The prime factorization of the generator polynomial brings with it certain potential characteristics, and in particular gives a tradeoff between maximum number of possible detected errors *vs.* data word length for which the polynomial is effective. Many polynomials are good for short words but poor at long words, and the converse. Unfortunately, factorization of a polynomial is not sufficient to determine actual HDs. A polynomial with a promising factorization might be vulnerable to some combination of bit errors, even for short message lengths. Thus, factorization characteristics suggest potential capabilities, but specific evaluation is required of any polynomial before it is suitable for use in a CRC function.

Conventional wisdom is that the best way to select a CRC polynomial is to use one that is already commonly used. For example, [Press92] lists 16-bit polynomials and states the choice of polynomial "is only a matter of convention." This approach assumes that those polynomials were selected for optimal error detection, which in some cases is incorrect. For example, several standardized 16-bit polynomials have error detection performance inferior to available alternatives, and appear to have been chosen to minimize the number of "1" bits in the feedback value at a time when each such bit had substantial hardware implementation cost. [Lin83] states that polynomial selection is "a very difficult problem" and says that some good cyclic codes have been discovered, but provides no details. Most coding theory and practice books are similar to [Wells99] in that they give only a handful of published polynomials and little or no guidance on polynomial selection tradeoffs.

A Hamming weight *N* is the number of errors, out of all possible message corruptions, that is undetected by a CRC using a particular polynomial. A set of Hamming weights captures performance for different numbers of bits corrupted in a message at a particular data word length, with each successively longer data word length having set of Hamming weights with higher values. The first non-zero Hamming weight determines a code's Hamming Distance.

Table 1 shows some example Hamming weights for CRC polynomials at a data word size of 48 bits, which is a representative length for many embedded networks. The first polynomial shown is the ubiquitous CCITT-16 polynomial 0x8810. 0x8810 is a hexadecimal representation of the polynomial $x^{16} + x^{12} + x^5 + 1$, with x^{16} as the highest bit and an implicit +1 term, as is common in software-based CRC implementations. It has only three "feedback" bits set in the polynomial, which was advantageous for early hardware implementations. For data words that are 48 bits in length, CCITT-16 detects all 1-bit errors (as does any CRC polynomial), and all 2- and 3-bit errors. However, it only provides HD=4 at this length because, as shown by the weights in Table 1, it fails to detect 84 of all possible 4-bit errors. In comparison, the 16-bit polynomial 0xC86C [Baicheva00] attains HD=6 at this length.

We can also do better than CCITT-16 for this example using smaller CRCs. The well known CAN 15-bit polyno-

mial 0x62CC, which is optimized for data word sizes of up to 112 bits, provides HD=6 at this length, missing only 4,314 of all possible 6-bit errors while using one less bit for its 15-bit CRC. Perhaps a surprise, though, is that 12-bit polynomial 0x8F8 can achieve HD=5 at this length, while the best published 12-bit CRC, 0xC07, achieves only HD=4. The 8-bit CRC-8 polynomial 0xEA also achieves HD=4 at this length – but a designer would have to know to use that published polynomial rather than the published DARC-8 polynomial, which does not. The smallest CRC polynomial achieving HD=4 at this length is the 7-bit CRC 0x5B (albeit with a higher weight than CCITT-16), although the best published 7-bit CRC achieves only HD=3. This example points out two fundamental problems with current practice: there are gaps in the set of published polynomials, and there is a need for specific guidance on which polynomials to use when.

Proposing to make changes to decades of entrenched CRC folklore and standardization is no small task. While some might think that there is little need for new CRC polynomials because networking standards force the use of existing polynomials, this is often not the case for embedded networks. New embedded networks are continually being developed, each with unique performance and error detection tradeoffs. The Train Control Network (TCN) is a recent example for which we provided a CRC effectiveness evaluation that demonstrated an 8-bit CRC could have provided HD=4 protection compared to the 7-bit CRC plus parity bit scheme used that provided only HD=3 protection [Koopman01]. In all fairness, TCN was constrained by legacy compatibility to use a 7-bit CRC; but many protocols for new application areas are not so constrained. Moreover, proprietary embedded network development happens continually in industry. Given that there is no end in sight to the proliferation of application-specific network protocols, it makes sense to find out what the best CRC polynomials are so that they can be used by new applications and emerging standards.

3. Previous work

Previous published work on CRC effectiveness has been limited by the computational complexity of determining the weights of various polynomials. Only a few detailed surveys of polynomials have been published. Baicheva surveyed 8-bit CRC polynomials with certain factorization structures up to data word length 127 [Baicheva98]. Baicheva proposed a good polynomial that was better than the ATM Header CRC polynomial. But, as discussed below, the actual optimal polynomial for the ATM Header data word length has a factorization that was not evaluated in that survey. This illustrates the importance of an exhaustive search of polynomials when designing a CRC for an application.

Baicheva later surveyed all 16-bit CRCs for data words up to 1024 bits in length [Baicheva00], identifying various polynomials as both "good" in general and optimal for particular data word lengths. A later survey paper [Kazakov01] extended those results to longer data words, focusing on computing a point-by-point optimal bound for 16-bit CRC effectiveness. We have included their results in our evaluation of alternatives.

While 8 and 16 bit CRC sizes are often used, it is also common to use other CRC sizes in embedded network applications to conserve bandwidth while achieving a particular desired HD. As illustrated by CAN's use of a 15-bit instead of 16-bit CRC, even saving one bit on the FCS field makes a difference that matters.

Finding a comprehensive list of well known polynomials in print is difficult. [Peterson72] gives a list of irreducible polynomials of degree 16 or less, but does not give an evaluation of error detection capabilities when those polynomials are used for CRCs. A search revealed published polynomials for 3-, 4-, 5-, 6-, 7-, 8-, 10-, 12-, 14-, 15-, and 16-bit CRCs. In many cases there is more than one recommended polynomial for a given CRC size. Worse, in the case of the CRC-12 polynomial, there are three different polynomials given under the same name, with most references evenly split between two of them. In the words of one of the more complete listings of polynomial candidates, "These [polynomials] differ in bit 1 and calculations using them return different values. With citations evenly split, it is hard to know which is correct" [Jaffer03]. Given a lack of published quantitative analysis, popularity contests are a common method for selecting polynomials.

This paper seeks to publish readily usable engineering guidelines for CRC selections of polynomial sizes 3 to 16 bits for embedded networks. CRC performance was determined by performing a complete evaluation of every possible undetected error pattern in messages as described in [Koopman02], yielding an exact result rather than an approximation. The balance of this paper first presents some case studies illustrating the severity of current problems, then describes a methodology for selecting "good" general purpose polynomials, and finally presents data for choosing embedded networking polynomials.

4. Case studies of current protocols

Adopting a previously published polynomial has two potential problems. One problem is that some polynomials in use simply provide very poor error detection capabilities overall. A second problem is that even a "good" polynomial is of necessity optimized for message sizes of a particular length, and will do poorly when misused for messages of a different length. Some case studies of current polynomials in use illustrate these points.

4.1. USB vs. ITU for 5 bit CRCs

5-bit CRCs are used, among other places, for providing error detection for Universal Serial Bus (USB) tokens and by an ITU standard for telecommunication systems. Figure 1 shows the performance of these polynomials compared to the best achievable bound. The probability of undetected error P_{ud} is summed from the probability of successively higher numbers of bit errors at an assumed Bit Error Rate (BER) of 10^{-6} weighted by the percentage of errors caught per corresponding polynomial weights for each data word length. The bound line shown assumes a different, optimal, polynomial is selected for each length, and thus is a firm bound on performance. Lower numbers are better, indicating a lower probability of undetected error slipping past the CRC.

The USB 5-bit CRC standard, "USB-5," is hexadecimal value $0x12 = x^5 + x^2 + 1$ [USB00]. This polynomial is used by USB to protect data words of length 11 bits. USB-5 is optimal for 11-bit messages, and is nearly optimal for longer data word lengths. It is, however, not necessarily a good choice for data words sized 10 and lower, because it is a full bit of HD worse than the bound.

ITU G.704 [G704] uses a 5-bit polynomial CCITT-5, $0x15 = x^5 + x^3 + x + 1$. Figure 1 shows CCITT-5 as optimal at length 10. However, CCITT-5 is a full bit of HD worse that USB-5 at lengths 11-26, and more than a factor of 2 worse than the bound (and USB-5) at longer lengths. ITU G.704 uses CCITT-5 for a data word length of 3151 bits, which is clearly a length at which this is an inefficient CRC. (At 3151 bits USB-5 is optimal, and is 2.077 times better at error detection that CCITT-5.).

CCITT-5's polynomial is divisible by (x+1), which results in an ability to detect all odd numbers of bit flips and is commonly said to be desirable (e.g., [Tanenbaum96]). Indeed, at a data word length of 3151, CCITT-5 is the best polynomial out of all 5-bit polynomials divisible by (x+1). USB-5, on the other hand, is not divisible by (x+1) and performs better than CCITT-5 at all lengths above 10 bits. To understand why USB-5 does so much better in Figure 1, it is helpful to examine the weight structure at length 3151, shown in Table 2. For a BER of 10^{-6}, most messages of length 3151 suffer zero-, 1-, or possibly 2-bit errors, with each increasing number of erroneous bits less likely. As Table 2 shows, USB-5 is almost twice as effective at detecting 2-bit errors, and the 3-bit error weight for USB-5 is not high enough for it to outweigh this advantage. Thus, USB-5 is superior for this BER.

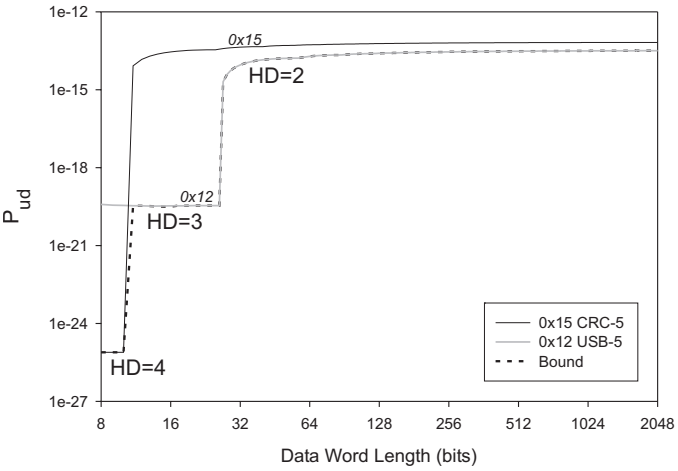

Figure 1. Performance of published 5-bit CRCs.

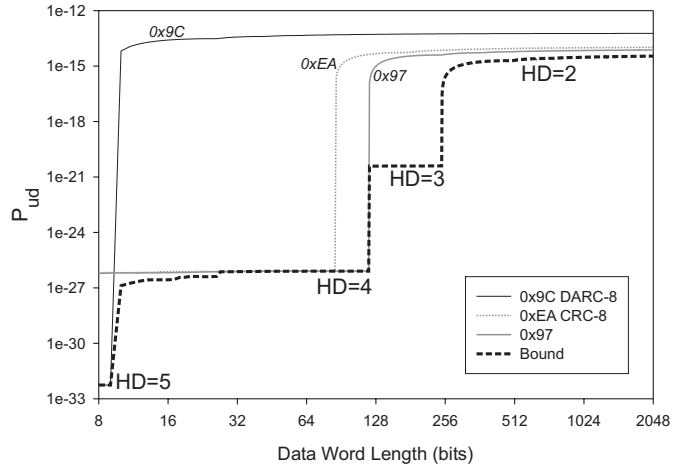

Figure 2. 8 bit, HD=4 CRC performance.

4.2. 8-bit polynomials

8-bit polynomials are commonly used because they are efficient for 8-bit microcontroller applications. Perhaps the most commonly used 8-bit polynomial is "CRC-8", which is $0xEA = x^8+x^7+x^6+x^4+x^2+1$. Figure 2 shows that this polynomial provides HD=4 up to length 85. However, at lengths 86 to 119 it provides only HD=2 compared to a bound of HD=4. At lengths above 119 it provides the same HD=2 as the bound, but with a significantly higher P_{ud}. Overall, this polynomial is adequate up to length 85, but could be improved upon. Many uses of CRC-8 in current systems are therefore questionable, because they attempt to provide error detection for a large data word (such as across a long message or a large block of memory data).

Table 2. Weights for CCITT-5 and USB-5; data word size 3151 bits.

Polynomial	HD	Hamming weights for number of bits corrupted:				
		1 bit	2 bits	3 bits	4 bits	5 bits
USB-5 0x12	2	0	159 075	163 552 409	128 929 654 767	81 278 805 135 219
CCITT-5 0x15	2	0	330 435	0	257 909 068 726	0

How can a designer do better than CRC-8? One way is to take advantage of published improved polynomials. In this particular case, [Baicheva98] has published polynomial C2, with value 0x97 = $x^8+x^5+x^3+x^2+x+1$. Figure 2 shows that 0x97 has the same or better performance as CRC-8 at every data word length (this is also true at lengths beyond 2048 bits as well). Because of this, 0x97 *dominates* CRC-8 and therefore is an unconditionally better polynomial choice given our fault assumptions. Moreover, 0x97 has optimal performance at length 119, the largest possible length at which HD=4 can be achieved with an 8-bit CRC. Unfortunately, such analysis for polynomials is difficult to find for other situations; there are no published complete surveys for other CRC sizes except those already mentioned.

The ATM-8 HEC polynomial 0x83 = $x^8 + x^2 + x + 1$ (said to be from ITU standard I.432) does not dominate CRC-8, but is close to C2. The two polynomials perform essentially the same at lengths above 112 bits, but ATM-8 degrades to 45% worse than C2 at length 8 bits. For a 32-bit ATM data word, C2 is 4.9% more effective than ATM-8 at error detection for moderate to low BERs.

A pitfall of choosing a published polynomial is that it might not be good for most data word lengths. For example, the DARC polynomial [ETSI02] 0x9C = $x^8+x^5+x^4+x^3+1$ is optimal for length 8, but provides only HD=2 at lengths 10 and above with rather poor P_{ud} performance as shown in Figure 2. The DARC application uses this polynomial for data word lengths 16 through 48 bits, where it performs poorly.

Figure 2 makes it clear that there are two missed opportunities even when choosing the best published 8-bit CRC polynomials. The first is that they miss the HD=3 "ledge" on the bound curve between lengths 120 and 247. The second is that none of them are close to optimal for lengths of 248 and higher. For example, CRC-8 is a factor of 3.3 worse at 1024 bits. Other published CRCs such as the ATM HEC do no better than a factor of 2.3 worse than the bound at 1024 bits and higher. But, there are polynomials that are far better than commonly used ones for HD=3 and HD=2 applications. Figure 3 shows that 0xA6 = $x^8 +x^6 +x^3 +x^2 +1$ maintains HD=3 up to 247 bits, and indeed is optimal at that length. Moreover, 0xA6 gives performance almost indistinguishable from the bound at lengths of 120 and up.

Two examples of real protocols help to illustrate the potential of improved CRCs: SMBus and Xmodem. SMBus is a low speed communication bus used for "smart" batteries and portable electronic device power management applications. Version 1.1 of SMBus added a CRC-8 polynomial [Smbus00], presumably inherited from the older I^2C bus standard. Many of the messages are between 16 and 40 bits in length, and are suitably protected by CRC-8 at HD=4. However, there is a data packet transfer command that results in 35-byte (280 bit) message payloads. The protection afforded by CRC-8 is only HD=2 for messages 11 bytes and longer. The ATM-8 polynomial might have been a better choice because it would have held HD=4 for longer messages, and provided better error detection for HD=2 operating regions.

One can make the case that 0xA6 would have been an even better choice for SMBus, depending on the expected message workload. Longer messages make bigger targets for random bit errors, and are therefore more likely to accumulate multiple errors. If the message workload for SMBus in a particular application makes heavy use of long messages, the HD=3 operating region of 0xA6 and the better performance for HD=2 operating regions might outweigh the penalty of increased vulnerability (HD=3 instead of HD=4) for short messages. The details depend on the weighted sum of undetected error probabilities for messages of each length, which would vary by application. But the point is that sometimes it is worth giving up a little error detection at short message lengths to gain better protection for longer messages. (An alternate strategy would be to use 0xA6 for long messages and a good HD=4 polynomial for short messages.)

Another common CRC-8 application is for the XMODEM protocol, developed by Ward Christansen in 1977. This protocol transmits packets in 128-byte (1024 bit) chunks protected by CRC-8. While polynomial evaluations were not generally available then, any of the alternate CRCs discussed here (except DARC-8) would have been a more effective choice.

5. Polynomial selection

The difference between good HD=4 polynomials and a good HD=3/HD=2 polynomial for 8-bit CRCs illustrates that a one-size-fits-all approach to CRC selection can cost a significant amount of error detection performance, including losing a bit (or more) of possible error detection capability for some message lengths. Therefore, selection of a good CRC polynomial must involve not only the size of the CRC, but also the size of the data word. Moreover, many commonly used polynomials are poorly suited to likely applications. Therefore, we propose "good" polynomial candidates and prescribe a method for selecting an appropriate candidate for each application.

5.1. Candidate polynomial selection

The selection of a "good" polynomial for generic use is of course a matter of engineering judgement. The following selection process was chosen to result in polynomials that primarily maintained high HD values to the longest data word lengths possible, secondarily achieved good performance at shorter lengths, and thirdly achieved good

performance at longer lengths than the stated maximum usage length. The prioritization of these goals keeps in mind that embedded network applications typically have a maximum message length that needs a certain HD; that short messages can benefit from improved HD protection so long as protection of long message is not materially sacrificed; and that sometimes a protocol revision adds messages longer than originally envisioned, so good performance at longer message lengths is desirable as a safety net.

The steps followed were performed for all distinct CRC polynomials of size 3 bits to 16 bits.

(1) Compute weights for all polynomials at data word lengths 8 bits through 2048 bits.

(2) Find the bounding weights by selecting the polynomial with the lowest weight at each length (i.e., generating a list of point-wise optimal polynomials). In general this means a different polynomial is selected as the bound for each length from 8 bits through 2048 bits, although in many cases a single high-performing polynomial happens to account for multiple bound values. This data was the source of our "bound" curves.

(3) Identify "break points" in the bound, in which the best achievable HD value changes. For example, the 8-bit CRC bound in Figure 2 has break points as follows: HD=5 is possible to length 9, so the HD=5 break point is at 9. The HD=4 break point is at length 119 (HD=4 is the best possible HD from length 10 to 119). The HD=3 breakpoint is at length 247. And of course HD=2 is possible at all lengths with any CRC polynomial. Follow the subsequent steps for each breakpoint in turn.

(4) Identify all polynomials that achieve the HD bound at the breakpoint. This guarantees that the polynomial selected gets all the way into the "corner" of the bound curve at the break point. If there are multiple polynomials, select the one with the lowest weights. If there is a tie for lowest weights or several low weights are within 1% of each other (a near-tie), invoke Steps (5) and (6). If there is an existing published polynomial within 1% of the bound, use that polynomial. This step excludes polynomials that might be better at lower weights at the expense of decreasing HD before the break point (the HD=4 break point of CRC-8, shown in Figure 2, is an example of a polynomial being pruned from consideration for this reason).

(5) If multiple polynomials have been identified in Step 4, select the polynomial having the longest-length break point for the next higher HD value. For example, there are 30 distinct 10-bit polynomials that achieve the break point HD=3 at length 1013, all of which have identical 3-bit Hamming weights. However, among those polynomials, the longest length for which HD=4 is possible is length 73 from polynomial 0x327, making it the choice for a "good" polynomial. Similarly, polynomial 0xBAAD is the only polynomial that provides both HD=4 at length 2048 and HD=5 up to length 108 bits (other polynomials with HD=4 at 2048 bits provide HD=5 at shorter data word lengths), and is only 0.39% worse in performance than the optimal polynomial at length 2048, which is 0xD3E9 given by [Kazakov01]. This screening step provides polynomials that not only provide good performance at break points, but also have a bonus of even better HD at smaller lengths.

Table 3. "Best" polynomials for HD at given CRC size and data word length. Underlined polynomials have been previously published as "good" polynomials.

Max length at HD / *Polynomial*	CRC Size (bits)													
	3	4	5	6	7	8	9	10	11	12	13	14	15	16
HD=2	2048+ 0x5	2048+ 0x9	2048+ 0x12	2048+ 0x21	2048+ 0x48	2048+ 0xA6	2048+ 0x167	2048+ 0x327	2048+ 0x64D	–	–	–	–	–
HD=3		11 0x9	26 0x12	57 0x21	120 0x48	247 0xA6	502 0x167	1013 0x327	2036 0x64D	2048 0xB75	–	–	–	–
HD=4			10 0x15	25 0x2C	56 0x5B	119 0x97	246 0x14B	501 0x319	1012 0x583	2035 0xC07	2048 0x102A	2048 0x21E8	2048 0x4976	2048 0xBAAD
HD=5						9 0x9C	13 0x185	21 0x2B9	25 0x5D7	53 0x8F8	none	113 0x212D	136 0x6A8D	241 0xAC9A
HD=6							8 0x13C	12 0x28E	22 0x532	27 0xB41	52 0x1909	57 0x372B	114 0x573A	135 0xC86C
HD=7									12 0x571	none	12 0x12A5	13 0x28A9	16 0x5BD5	19 0x968B
HD=8										11 0xA4F	11 0x10B7	11 0x2371	12 0x630B	15 0x8FDB

(6) If Step 5 does not apply or results in a tie, select the polynomial with best performance at 2048 bits (the maximum length computed). This yields a polynomial that has good performance for long data word lengths.

(7) If Step 6 is an approximate tie (the HD weights are within 1%), pick the polynomial with the best weights at smaller lengths, even if those smaller lengths are at the same HD as the break point.

The result of applying this selection process is shown in Table 3. Each cell in Table 3 has two numbers – a top number for the break point length at the given HD, and a bottom "good" polynomial for lengths up to the break point. Lengths above 2048 bits were not studied in detail, but for 11 bit and smaller CRCs, the given polynomials are also near-optimal for arbitrarily longer lengths. The underlined polynomials are ones that have been previously published as suitable for use in CRCs (see Table 4 for details). Table 3 can be used in the following ways:

- *Find a "good" polynomial given CRC size and length:* Select the appropriate CRC size column in Table 3. Pick the row with the smallest length greater than or equal to the desired length. The polynomial in that box will provide the best HD possible at that length and CRC size. For example, for a 9-bit CRC, a length of 246 would use polynomial 0x14B and achieve HD=4, but the best HD that can be achieved at length 247 is HD=3 using polynomial 0x167.
- *Find the minimum size CRC required to achieve a given HD at a particular length:* Pick the row of Table 3 with the desired HD. Select the furthest left column in that row with a length greater than or equal to the desired length. That is the smallest CRC that can provide the desired HD at the required length. For example, HD=6 for a data word length of 52 bits can be achieved with a 13-bit CRC using polynomial 0x1909.

Of course this selection table is not without limitations. For applications that have only a single data word length, and in which optimal performance is required even at the expense of more effort in polynomial selection, an optimal polynomial should be selected.

5.2. Performance of published polynomials

Despite the fact that Table 3 has many novel polynomials, that does not necessarily mean that a previously published polynomial will perform poorly in any specific application. In particular, the self-imposed requirement to achieve the maximum possible HD for break point values disqualified some otherwise good standard polynomials. For example the CAN polynomial 0x62CC is good for lengths up to 112 bits, but has a break point at 112 bits compared to the bound's break point at 114 bits for HD=6. For most applications, the CAN polynomial is likely to be good enough, and there is little point selecting a novel polynomial. Therefore, it is important to present an evaluation of the performance of commonly used polynomials so designers can choose between the extra potential effort of justifying a "non-standard" polynomial selection vs. the potential error detection gain.

Table 4 is a list of polynomials either in public use or proposed as "good" polynomials in the literature that we have encountered. (Only select polynomials from the published 8- and 16-bit surveys previously discussed have been included, since they are for the most part point solutions rather than generic suggested polynomials.) The best available citation for each polynomial has been given, along with the most commonly used nickname.

Table 4 gives the performance for data word lengths through 2048 bits. To keep the data manageable, performance is categorized into four columns. The first column indicates that performance is optimal or near optimal (within 1% of the best possible performance bound). The second column indicates that undetected error probabilities are within a factor of two of the bound, which is a somewhat arbitrary distinction but overall is useful in conveying which polynomials are close to being good for particular data ranges. The third column indicates where polynomials have more than twice the undetected error rate of the bound, and the fourth column indicates where each polynomial's HD is one or more bits worse than the bound. Underlined entries correspond to recommendations from Table 3. Tables of weights for these polynomials and bounds are available from the primary author, and are on the Web at http://www.ece.cmu.edu/~koopman/crc

Some CRC polynomials appear to be incorrect as a result of data transcription or similar errors that have occurred as polynomials are passed down over time. For example [Ottoson01] gives a "CRC-7" value of 0x68 instead of 0x48 as given elsewhere, which might be due to a one-bit data transcription error into a source code binary

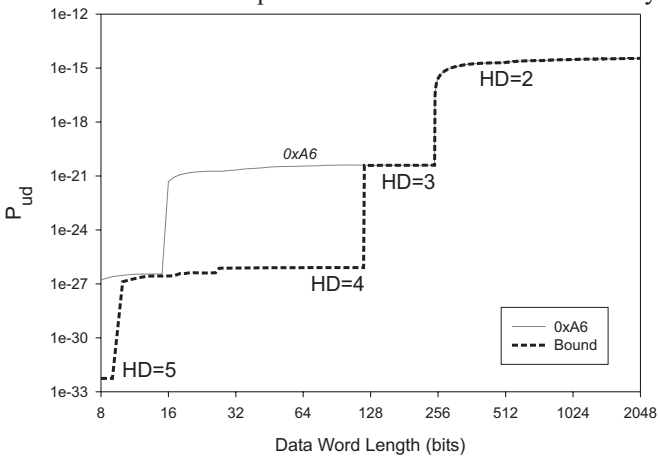

Figure 3. A good 8-bit polynomial for lengths 120 and above.

Table 4. Performance of polynomials at BER = 10^{-6}. Underlined ranges correspond to Table 3.

CRC Size (Bits)	Nickname	Polynomial	Source	Performance compared to bound at lengths given			
				Within 1% of bound	Within 2x of bound	Same HD, but more than 2x bound	Worse HD than bound
3		$0x5 = (x^3 + x + 1)^‡$	[RFC 3095]	8-2048	–	–	–
4	CCITT-4:	$0x9 = (x^4 + x + 1)^‡$	[G704]	8-2048	–	–	–
4	CRC-4:	$0xF = (x^4 + x^3 + x^2 + x + 1)$	[Ottoson01]	–	–	12-2048	8-11
5	CRC-5:	$0x15 = (x+1)(x^4 + x^3 + 1)^‡$	[G704]	8-10	–	27-2048	11-26
5		$0x12 = (x^5 + x^2 + 1)^‡$	[USB00]	11-13; 17-2048	14-16	–	8-10
6	DARC-6:	$0x2C = (x+1)(x^5 + x^4 + x^2 + x + 1)^‡$	[ETSI02]	12-25	8-11	58-2048	26-57
6	CRC-6:	$0x21 = (x^6+x+1)^‡$	[G704]	26-28; 37-2048	29-36	–	8-25
7		$0x5B = (x+1)(x^6 + x^5 + x^3 + x^2 + 1)^‡$	*new*	29-56	11-28	8-10; 121-2048	57-120
7	CRC-7:	$0x48 = (x^7+x^4+1)^‡$	[G704]	87-91; 99-2048	57-86; 92-98	–	8-56
7		$0x44 = (x^7+x^3+1)^‡$ (CRC-7 inverse)	[G832]	87-91; 99-2048	57-86; 92-98	–	8-56
	FT2:	$0x72 = (x^7+x^6+x^5+x^2+1)^‡$	[Funk88]	87-89; 99-2048	57-86; 90-98	–	8-56
7		$0x67 = (x+1)(x^3 + x + 1)^‡(x^3 + x + 1)^‡$	[RFC3095]	–	–	121-2048	8-120
7		$0x68 = (x+1)(x^2 + x + 1)^‡(x^4 +x^3 +1)^‡$	[Ottoson01]	–	8	121-2048	9-120
8	DARC-8:	$0x9C = (x^8 + x^5 + x^4 + x^3 + 1)$	[ETSI02]	8-9	–	248-2048	10-247
8	C2:	$0x97 = (x+1)(x^7 + x^6 + x^5 + x^2 + 1)^‡$	[Baicheva98]	27-50; 52; 56-119	18-26; 51; 53-55	10-17; 248-2048	8-9; 120-247
8	DOWCRC:	$0x98 = (x+1)(x^7+x^6+x^5+x^3+x^2+x+1)^‡$	[Whitfield01]	43-119	19-42	10-18; 248-2048	8-9; 120-247
8	ATM-8:	$0x83 = (x+1)(x^7+x^6+x^5+x^4+x^3+x^2+1)^‡$	[Ottoson01]	53-119	18-52	10-17; 248-2048	8-9; 120-247
8	WCDMA-8:	$0xCD = (x+1)(x^7 + x^3 + 1)^‡$	[Ottoson01]	28-31; 43-119	20-27;32-42	10-19; 248-2048	8-9; 120-247
8		$0xA6 = (x^8 +x^6 +x^3 +x^2 +1)^‡$	*new*	136-140; 210-2048	11-15; 120-135; 141-209	10	8-9; 16-119
8	CRC-8:	$0xEA = (x+1)(x^2+x+1)^‡(x^5+x^4+x^3+x^2+1)^‡$	[Ottoson01]	–	20-85	10-19; 248-2048	8-9;86-247
9		$0x13C = (x+1)(x^8 +x^7 +x^6 +x^4 +x^2+x+1)$	*new*	8	–	503-2048	9-502
9		$0x185 = (x^2 + x + 1)^‡(x^3 +x^2 + 1)^‡(x^4 +x^3 +1)^‡$	*new*	13-16	–	9-12; 503-2048	8; 17-502
9		$0x14B = (x+1)(x^8 +x^7 +x^3 +x^2 +1)^‡$	*new*	147-246	26-28; 30-146	14-25; 29; 503-2048	8-13; 247-502
9		$0x167 = (x^9 +x^7 +x^6 +x^3 +x^2 +x +1)^‡$	*new*	45-46; 48; 412-2048	18-44; 47; 247-411	14-17	8-13; 49-246
10		$0x28E = (x+1)(x^2 + x + 1)^‡(x^3 +x^2 + 1)^‡(x^4 +x^3+1)^‡$	*new*	9-12	8; 77-95	22-76; 1014-2048	13-21; 96-1013
10		$0x2B9 = (x^5 +x^2 +1)^‡(x^5 +x^3 +x^2 +x +1)^‡$	*new*	17-21	13-16	1014-2048	8-12; 22-1013
10	CRC-10:	$0x319 = (x+1)(x^9 + x^4 + 1)^‡$	[Jaffer03]	306-501	73-305	22-72; 1014-2048	8-21; 502-1013
10		$0x327 = (x^{10} +x^9 +x^6 +x^3 +x^2 +x +1)^‡$	*new*	880-2048	32-73; 502-879	22-31	8-21; 74-501
11		$0x571 = (x^{11} +x^9 +x^7 +x^6 +x^5 +x +1)$	*new*	8-12	–	2037-2048	13-2036
11		$0x532 = (x+1)(x^{10} +x^9 +x^5 +x +1)$	*new*	21-22	13-20	2037-2048	8-12; 23-2036
11		$0x5D7 = (x^{11} +x^9 +x^8 +x^7 +x^5 +x^3 +x^2 +x +1)^‡$	*new*	25-28; 1775-2048	23-24; 1013-1774	–	8-22; 29-1012
11		$0x583 = (x+1)(x^{10} +x^9 +x^7 +x^6 +x^5 +x^4 +x^3 +x^2 +1)^‡$	*new*	625-1012	13-17; 96-624	27-95; 2037-2048	8-12;18-26; 1013-2036
11		$0x64D = (x^{11} +x^{10} +x^7 +x^4 +x^3 +x +1)^‡$	*new*	111-131; 1775-2048	62-110; 96-624; 1013-1774	27-61	8-26; 132-1012
12		$0xA4F = (x+1)(x^{11} +x^{10} +x^6 +x^5 +x^4 +x^2 +1)$	*new*	8-11	–	–	12-2048
12		$0xB41 = (x+1)(x^3 +x^2 +1)^‡(x^8 +x^4 +x^3 +x^2 +1)^‡$	*new*	22-27	13-21	12	8-11
12		$0x8F8 = (x^{12} +x^8 +x^7 +x^6 +x^5 +x^4 +1)$	*new*	42-53	28-41	–	8-27; 54-2048
12		$0xC05 = (x^2+x+1)^‡(x^2+x+1)^‡(x^8+x^7+x^6+x^5+x^2+x+1)^‡$	[Press92]	–	55-160	54	8-53; 161-2048
12		$0xC06 = (x^{12}+x^{11}+x^3+x^2+1)$	[Whitfield01]	–	–	–	8-2048
12	CRC-12:	$0xC07 = (x+1)(x^{11} +x^2 +1)^‡$	[Ottoson01]	1074-2035	221-1073	54-220	8-53; 2036-2048
12		$0xB75 = (x^2+x+1)^‡(x^3+x^2+1)^‡(x^7 +x^5 +x^4 +x^3+1)^‡$	*new*	2036-2048	–	–	8-2035

Table 4 continued.

CRC Size (Bits)	Nickname	Polynomial	Source	Performance compared to bound at lengths given: Within 1% of bound	Within 2x of bound	Same HD, but more than 2x bound	Worse HD than bound
13		0x10B7 = $(x+1)(x+1)(x^{11}+x^9+x^7+x^6+x^5+x+1)$	*new*	10-11	8-9	–	12-2048
13		0x12A5 = $(x^2+x+1)^{\ddagger}(x^{11}+x^{10}+x^6+x^5+x^4+x^2+1)$	*new*	12-13	–	53-56	8-11; 14-52; 57-2048
13		0x1909 = $(x+1)(x^{12}+x^8+x^7+x^6+x^5+x^4+1)$	*new*	44-52	21-43	13, 14-20	8-12; 53-2048
13		0x102A = $(x^4+x^3+x^2+x+1)(x^9+x^8+x^4+x^3+x^2+x+1)^{\ddagger}$	*new*	363-2048	171-362	53-170	8-52
14		0x2371 = $(x+1)(x+1)(x+1)(x^{11}+x^{10}+x^6+x^5+x^4+x^2+1)$	*new*	11	10	8-9	12-2048
14		0x28A9 = $(x^{14}+x^{12}+x^8+x^6+x^4+x+1)^{\ddagger}$	*new*	13-14	12; 136-317	114-135	8-11; 15-113; 318-2048
14		0x372B = $(x+1)(x+1)(x^{12}+x^{11}+x^{10}+x^7+x^5+x^4+x^3+x+1)^{\ddagger}$	*new*	51-57	30-50; 304-2048	14-29; 114-303	8-13; 58-113;
14		0x212D = $(x^7+x^6+x^3+x+1)^{\ddagger}(x^7+x^6+x^5+x^4+1)^{\ddagger}$	*new*	101-113	20-21; 58-100	14-19	8-13; 22-57; 114-2048
14		0x21E8 = $(x^4+x^3+x^2+x+1)(x^{10}+x^9+x^4+x+1)^{\ddagger}$	*new*	1940-2048	227-1939	114-226	8-113
14	DARC-14:	0x2402 = $(x+1)(x^{13}+x^{12}+x^{11}+x+1)^{\ddagger}$	[ETSI02]	–	540-2048	114-539	8-113
15		0x630B = $(x+1)(x^{14}+x^9+x^3+x^2+1)^{\ddagger}$	*new*	12	10-11; 732-2048	8-9; 137-731	13-136;
15		0x5BD5 = $(x^3+x^2+1)^{\ddagger}(x^{12}+x^{11}+x^8+x^6+x^5+x^3+x^2+x+1)$	*new*	16	14-15	13	8-12; 17-2048
15		0x573A = $(x+1)(x^{14}+x^{13}+x^{10}+x^8+x^7+x^6+x^4+x+1)$	*new*	78; 98-114	49-77; 79-97	17-48	8-16; 115-2048
15	CAN:	0x62CC = $(x+1)(x^7+x^3+1)^{\ddagger}(x^7+x^3+x^2+x+1)^{\ddagger}$	[Bosch91]	64-112	49-63	17-48	8-16; 113-2048
15		0x6A8D = $(x^{15}+x^{14}+x^{12}+x^{10}+x^8+x^4+x^3+x+1)$	*new*	123-136	115-122	–	8-114; 137-2048
15		0x4976=$(x^4+x^3+x^2+x+1)(x^{11}+x^{10}+x^8+x^7+x^6+x^4+x^2+x+1)^{\ddagger}$	*new*	1771-2048	441-1770	17-18; 137-440	8-16; 19-136
15		0x4001 = $(x^{15}+x+1)^{\ddagger}$	[Gilmore02]	–	–	–	8-2048
15		0x740A=$(x^3+x^2+1)^{\ddagger}(x^6+x+1)^{\ddagger}(x^6+x^4+x^2+x+1)$	[MPT1327]	–	–	–	8-2048
16		0x8FDB = $(x+1)(x^{15}+x^{14}+x^{13}+x^{12}+x^{10}+x^8+x^6+x^5+x^3+x^2+1)^{\ddagger}$	*new*	14-15	12-13; 42-51; 913-2048	8-11; 20-41; 242-912	16-19; 52-241
16		0x968B = $(x^2+x+1)^{\ddagger}(x^{14}+x^{13}+x^9+x^7+x^5+x^4+1)$	*new*	19	16-18; 245-363	8; 242-244	9-15; 20-241; 364-2048
16		0xC86C = $(x+1)(x^{15}+x^{11}+x^{10}+x^9+x^8+x^7+x^5+x^4+x^2+x+1)$	[Baicheva00]	110-111; 113-135	39-109; 112	20-38	8-19; 136-2048
16	C_3	0xAC9A = $(x^{16}+x^{14}+x^{12}+x^{11}+x^8+x^5+x^4+x^2+1)$	[Castagnoli90]	219-241	33-35; 136-218	8; 20-32	9-19; 36-135; 242-2048
16		0xBAAD = $(x^3+x^2+1)^{\ddagger}(x^6+x^5+x^2+x+1)^{\ddagger}(x^7+x^3+1)^{\ddagger}$	*new*	1578-2048	242-1577	20	8-19; 21-241
16		0xD3E9 = $(x^3+x^2+1)^{\ddagger}(x^6+x^5+x^2+x+1)^{\ddagger}(x^7+x^6+x^5+x^4+1)^{\ddagger}$	[Kazakov01]	1270-2048	256-1269	242-255	8-241
16	CRC-16:	0xA001 = $(x+1)(x^{15}+x^{14}+1)^{\ddagger}$	[Press92]	–	1270-2048	242-1269	8-241
16	ANSI-16:	0xC002 = $(x+1)(x^{15}+x+1)^{\ddagger}$	[USB00]	–	1270-2048	242-1269	8-241
16		0xD04B = $(x+1)(x+1)(x^{14}+x^{13}+x^{12}+x^{10}+x^8+x^6+x^5+x^4+x^3+x+1)^{\ddagger}$	[Ottoson01]	–	39-67; 1146-2048	8; 20-38; 242-1145	9-19; 68-241
16	IEC-16:	0xADC9=$(x+1)(x+1)(x^7+x^6+x^3+x+1)^{\ddagger}(x^7+x^6+x^5+x^4+x^3+x^2+1)^{\ddagger}$	[Baicheva00]	–	39-112	8-9; 20-38	10-19; 113-2048
16	CCITT-16:	0x8810 = $(x+1)(x^{15}+x^{14}+x^{13}+x^{12}+x^4+x^3+x^2+x+1)^{\ddagger}$	[ETSI02]	–	1015-2048	242-1014	8-241

Notes: All polynomials are represented as a product of irreducible factors

All polynomial factors marked with (‡) are primitive.

data array. Unfortunately, 0x68 is dominated by six other polynomials and is generally inferior to 0x48.

Similarly [Jaffer03] indicates that various web-based sources are evenly split between 0xC06 and 0xC07 as being the "standard" CRC-12 polynomial. *Numerical Recipes* states that CRC-12 is 0xC05 [Press92]. One can speculate that these are also one-bit transcription errors.

6. Conclusions

New embedded networks that use CRCs are continually being created. Unfortunately, the usual practice of selecting a published CRC polynomial under the presumption that it is "good" runs into trouble because some published values perform quite poorly. Moreover, even if a good published polynomial is available, there is generally no published guidance on what range of data word lengths it is good for nor, for that matter, quantitative data to help distinguish that good polynomial from any competing published or standardized "bad" polynomials.

This paper presents the first exhaustive survey of all CRC polynomials from 3 bits to 15 bits, and discusses 16-bit polynomials as well. A methodology for selecting generically "good" CRC polynomials is based on achieving maximum Hamming Distance for the longest possible data word sizes and other performance considerations. Our tables of good polynomials given should enable practitioners to use quantitative information in selecting effective polynomials for embedded computing error detection applications for data word sizes up to 2048 bits.

7. Acknowledgments

Thanks to Kobey DeVale for her help in early explorations. This work was supported in part by grants from Bombardier Transportation, the General Motors Collaborative Laboratory at Carnegie Mellon University, and the Pennsylvania Infrastructure Technology Alliance.

8. References

[Baicheva98] Baicheva, T., S. Dodunekov & P. Kazakov, "On the cyclic redundancy-check codes with 8-bit redundancy," *Computer Communications*, vol. 21, 1998, pp. 1030-1033.

[Baicheva00] Baicheva, T., S. Dodunekov & P. Kazakov, "Undetected error probability performance of cyclic redundancy-check codes of 16-bit redundancy," *IEEE Proc. Comms.*, Vol. 147, No. 5, Oct. 2000, pp. 253-256.

[Bosch91] Bosch, *CAN Specification*, Version 2, 1991.

[Castagnoli93] Castagnoli, G., S. Braeuer & M. Herrman, "Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits", *IEEE Trans. on Communications*, Vol. 41, No. 6, June 1993.

[Gilmore02] Gilmore, J., "Parallel CRC Equation Generator" (source code), accessed on October 22, 2002 at: http://www.physics.ohio-state.edu/~cms/crc_generator

[ETSI02] ETSI, *Radio broadcasting systems; DAta Radio Channel (DARC); System for wireless infotainment forwarding and teledistribution, ETSI EN 300 751, V1.2.1,* October 2002.

[Funk88] Funk, G., "Performance comparison of standard frame transmission formats FT1.2 and FT2 specified by IEC TC57", *ntzArchiv Bd. 10*, 1988, pp. 217-221.

[G704] International Telecommunication Union, *General Aspects of Digital Transmission Systems: Synchronous frame structures used at 1544, 6312, 2048, 8488 and 44 736 kbit/s hierarchical levels, ITU-T Recommendation G.704 (previously "CCITT Recommendation")*, July 1995.

[G832] International Telecommunication Union, *Transport of SDH Elements on PDH Networks: Frame and Multiplexing Structures, ITU-T Recommendation G.832 (previously "CCITT Recommendation")*, November 1993.

[Jaffer03] Jaffer, A., "Mathematical packages: cyclic checksum," accessed September 27, 2003 at http://www.swiss.ai.mit.edu/~jaffer/slib_4.html#IDX522

[Kazakov01] Kazakov, P., "Fast calculation of the number of minimum-weight words of CRC codes," *IEEE Trans. Information Theory*, (47) 3, March 2001, pp. 1190-1195.

[Koopman01] Koopman, P. & T. Chakravarty, "Analysis of the Train Communication Network Protocol Error Detection Capabilities," Feb. 25, 2001, http://www.tsd.org/papers/

[Koopman02] Koopman, P., "32-bit cyclic redundancy codes for Internet applications," *Intl. Conf. Dependable Systems and Networks (DSN)*, Washington DC, July 2002, pp. 459-468.

[Lin83] Lin, Shu & D. Costello, *Error Control Coding*, Prentice-Hall, 1983.

[MPT1327] MX.com, *Error detection and correction of MPT1327 formatted messages using MX429A or MX809 devices, application note MPT1327*, 1999.

[Ottosson01] Ottosson, T., T. Eriksson, P. Frenger, T. Ringström & J. Samuelsson, "crc.cpp," version 1.8, May 22, 2003, at: http://itpp.sourceforge.net/crc_8cpp-source.html

[Peterson72] Peterson, W. & E. Weldon, *Error-Correcting Codes*, MIT Press, Second Edition, 1972.

[Press92] Press, W., S. Teukolsky, W. Vetterling, & B. Flannery, *Numerical Recipes in C (2nd ed.)*, Cambridge Press, 2002.

[RFC3095] Bormann, C. (ed.), *RFC 3095 - RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed*, 2001, accessed at: http://www.faqs.org/rfcs/rfc3095.html

[Smbus00] SBS Implementers Forum, *System Management Bus (SMBus) Specification,* Version 2.0, August 3, 2000.

[Tanenbaum96] Tanenbaum, A., *Computer Networks* (3rd. Ed.), Prentice Hall, 1996.

[USB00] *Universal Serial Bus Specification*, Rev. 2.0, 2000.

[Wells99] Wells, R., *Applied coding and information theory for engineers*, Prentice-Hall, 1999.

[Whitfield01] Whitfield, H., "XFCNs for Cyclic Redundancy Check Calculations," April 24, 2001, accessed at http://homepages.cs.ncl.ac.uk/harry.whitfield/home.formal/CRCs.html