2016

Analyzing error detection performance of checksums in embedded networks

Mirza, A. N.
Faculty of Information Technology (FIT)
Strathmore University

Analyzing Error Detection Performance of Checksums in Embedded Networks

Ali Naqi Mirza

Submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Based Information Systems at Strathmore University

Faculty of Information Technology
Strathmore University
Nairobi, Kenya

June, 2016

Declaration

I declare that this work has not been previously submitted and approved for the award of a degree by this or any other University. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.

…..Ali Naqi Mirza…….. [*Name of Candidate*]

…… [*Signature*]

….June 7, 2016……... [*Date*]

Approval

The thesis of Ali Naqi Mirza was reviewed and approved by the following:

Dr. Vitalis Ozianyi, PhD
Faculty of Information Technology
Strathmore University

Dr. Joseph Orero, PhD
Dean, Faculty of Information Technology
Strathmore University

Professor Ruth Kiraka
Dean, School of Graduate Studies

Abstract

Networks are required to transport data from one device to another with adequate precision. For a majority of applications a system is expected to ensure that received data and transmitted data is uniform and consistent. Many elements can change one or more bits of a message sent. Applications therefore need a procedure for the purpose of identification and correction of errors during transmission.

Checksums are frequently used by embedded networks for the purpose of identifying errors in data transmission. But decisions regarding which checksum to utilize for error detection in embedded networks are hard to make, since there is an absence of statistics and knowledge about the comparative usefulness of choices available.

The aim of this research was to analyze the error detection performance of these checksums frequently utilized: XOR, one's complement addition, two's complement addition, Adler checksum, Fletcher checksum, and Cyclic Redundancy Check (CRC) and to assess the error detection effectiveness of checksums for those networks which are prepared to give up error detection efficiency in order to curtail costs regarding calculations and computations, for those wanting uniformity between error identification and costs, and finally for those networks which are ready to yield elevated costs for notably enhanced error detection.

Even though there is no one size fits all method available, this research gives recommendations in order to decide as to which checksum approach to adopt. A bit flip fault design with manufactured error simulations was utilized for this research. Mathematical technique used for the proposed fault design was Monte Carlo simulations using Mersenne twister random number generator. This study concludes that the error identification performance of XOR, and Adler checksums for arbitrary and autonomous bit and burst errors is below an accepted level, rather 1's and 2's complement checksums should be utilized for networks prepared to surrender error identification efficiency in order to minimize the cost of calculations. Fletcher checksum should be utilized by networks wanting symmetry between computational costs and error identification and CRCs by networks prepared to yield greater costs of computation for notably enhanced error identification.

Table of Contents

Acknowledgements

I would like to acknowledge my gratitude to my Supervisor Dr. Vitalis Ozianyi, for his advice and guidance in making sure that I am able to submit my Research Proposal in time. I would also thank my classmates for their help and motivation. Above all, Praise to GOD who gave me the strength to accomplish my Research.

Abbreviations and Acronyms

FCS	-	Frame Check Sequence
CRC	-	Cyclic Redundancy Codes
CPU	-	Central Processing Unit
WSC	-	Weighted Sum Codes
IEEE	-	Institute of Electrical and Electronics Engineers
ITU-T	-	International Telecommunications Union
ASCII	-	American Standard Code for Information Exchange
CAN	-	Controller area Network
TCP	-	Transmission Control Network
HD	-	Hamming Distance
HW	-	Hamming Weight
HDLC	-	High Level Data Link Control
CCITT	-	Consultative Committee for International Telephony and Telegraphy
LAN	-	Local Area Network
BER	-	Bit Error Rate
MSB	-	Most Significant Bit
Pud	-	Possibility of Undetected Error
LRC	-	Longitudinal Redundancy Check
XOR	-	Logical eXclusive OR

Definition of Terms

Checksum: A sum obtained from the bits of a chunk of computer data that is computed prior to and after storage, to ensure that data is independent from errors and manipulation.

Embedded Networks: Networks built on embedded processors giving communication links for certain applications.

Micro Controllers: A small computer on a single integrated circuit comprising a processor core, memory and programmable input/output peripherals.

Hamming Distance: Distance between the two strings of equivalent length, is the number of locations at which the corresponding symbols are different.

Polynomials: A sequence of 0's and 1's that can be illustrated as a polynomial with coefficients of 0 and 1.

List of Figures

List of Equations

List of Tables

Dedication

I dedicate this work to my family. Surely without them, I could not have the completed the dissertation. They gave me all support needed to accomplish this task.

Chapter 1: Introduction

1.1 Background

A Checksum is an error detection process that is generated by adding all the bytes in a data word in order to produce a checksum value. In network applications it is commonly known as (FCS) or frame check sequence. Checksum is attached to the (Payload) data word and transferred with it. Received data word is recalculated at the receivers end and matched with the value of checksum. There is an improbability of errors in transmission in cases where recalculated and received checksum are equal. Certainly there is a possibility that structure of changed bits in the sent messages just happens to transpire in a flawed data word emulating the transferred and also possibly the flawed value of checksum. There is an incompatible balance between the power of computation utilized on calculating checksum, FCS field size, and the possibility of such unidentified errors.

Checksums that are frequently utilized typically fall in three main categories of cost/performance balance. A straight forward and less significant checksum incorporates a simple add function across all bytes in a message. XOR, one's complement and two's complement addition are mostly utilized add functions. Computation costs of these checksums are economical; however they give very moderate error detection. Even though analyzing the error detection of these checksums was not particularly problematic, an absolute and detailed handling of the capabilities of these checksums is difficult to find in the literature. Analytic differentiation of their error detection capabilities is given by Plummer (1989) however quantitative data is not provided.

Cyclic Redundancy Check (CRC) is the most costly checksum which is frequently utilized. To be precise CRC is not a sum, but more of an error detecting code utilizing polynomial division for the purpose of computation. CRC calculations can be a heavy load on CPU, particularly in various embedded systems where small processors are common sight.

Since computation of CRC is expensive, two relatively economical checksums are suggested for usage. Fletcher (Fletcher, 1982) and Adler (Deutsch and Gailly, 1996) checksums are intended to provide error detection services which are practically as robust as CRC's with

notably curtailed costs of computations. For the implementation of Fletcher checksum Sklower (1989) and Nakassis (1988) proposed capability enhancements. It can be said that though Fletcher and Adler checksum services are practically as robust as a comparatively fragile CRC, in certain significant circumstances they are notably unsatisfactory.

While Checksums analyzed in this research have been in the scope of knowledge for many years, comparatively a small amount of research is available regarding their error detection performance. Information about error detection performance publicized by Fletcher provides a comparative analysis of checksums that are analyzed in this research. Studies conducted on the subject in the early days were primarily focused on the lists of effective CRC polynomials for fixed lengths (Baicheva, Dodunekov, and Kazakov, 1998).

The efficiency of one's complement, Fletcher checksum, and CRC was evaluated by Stone, Hughes, and Partridge (1995), they established in their research that one's complement and Fletcher checksum have elevated possibilities of unidentified errors than they actually presumed. The rationale they gave for this is the inconsistency of network data.

CRC polynomials presently utilized were examined by (Koopman, 2002). He suggested substitutes which gives improved performance. For embedded networks he suggested a polynomial electing procedure and established conducive criteria for CRC 3 to CRC 16 for data words up to 2048 bits.

Weighted sum codes (WSC) was suggested by McAauley (1994) as a substitute to Fletcher checksum and CRC. A comparative analysis of WSC as opposed to one's complement addition, XOR checksum, blocks parity, Fletcher checksum and CRC. He stressed that in error detection WSC is as robust as CRC and as swift as Fletcher checksum for speed calculations. Since WSC's are not frequently utilized therefore its analysis is out of scope for this research.

An account of various algorithms for the purpose of checksum computations are analyzed in this research in sequence of raising cost of calculations from XOR to CRC. Furthermore this research evaluates the approaches taken for each checksum and its subsequent adequacy and inadequacy, stressing on specific susceptibilities to unidentified errors based on bit error

structures and data values. Research also evaluates the cost efficiency of several substitutes. In some situations frequently utilized checksum method is a better option.

1.2 Problem Statement

A normal method of securing the integrity of communicated data is to add a checksum. Even though it is widely established that Cyclic Redundancy Code (CRC) have high capacity of identifying errors, a large majority of embedded networks utilize the services of less capable checksum methods in order to minimize the costs of computations in immensely restricted systems. Even embedded applications with high capacity are commonly unable to provide for custom hardware created for CRC assistance. Occasionally this cost/performance balance is understandable. But at times designers renounce superior error detection efficiencies without obtaining any proportional advantages in enhanced computation speed or decreased memory foot print.

While checksums and CRC's have been utilized for many years, (Prange 1957) publicized research into their effectiveness is rather limited. In addition, many distinct application spheres, utilize a broad choices of data integrity methods that do not surely relate to publicized leading practices. Partly this appears to be because of calculation costs that have restricted the ability to analyze CRC performance and partly because of lack of communication between mathematical explanations of methods and what professionals require to effectively execute those methods. Frequently, it is purely because of specialists emulating challenging error identification methods already utilized under the improper presumption that broadly utilized methods must be surely better methods. Though utilization of enhanced substitute checksums can attain better error detection performance in contemporary protocols, the level of enhancement attainable is uncertain from current literature. Since there is a consistent development of contemporary network protocols for numerous applications, it is significant to comprehend knowledge about checksum methods and procedures that work best.

In response to these problems the research investigates most frequently utilized checksums in embedded networks. It analyzes the relative error detection performance of these checksums and the aspects regarding their cost/performance balance.

1.3 Research Objectives

i. To review checksums approaches/methods utilized in embedded networks.

ii. To analyze relative error detection performance of checksums, their corresponding cost/performance tradeoff points and evaluate effectiveness of error detection through analysis and simulation.

iii. To investigate the reasons of unidentified errors for several checksums and propose recommendations for mapping effectiveness of error detection to a realistic level of integrity.

1.4 Research Questions

i. What are the major checksum approaches used in embedded networks?

ii. How the relative error detection performance of checksums and their corresponding cost performance will be analyzed?

iii. What are the major reasons of undetected errors for several checksums?

1.5 Scope and Limitations

It is important to position this study around some specific areas that are examined more closely. Due to time constraints this research had limitations. It focused only on analyzing the error detection performance of the following checksums: XOR, one's complement, two's complement, Adler checksum, Fletcher checksum, and CRC. Error correction is not part of this research. We leave an assessment of that for future research. The aim of the research is to thoroughly analyze the literature available on the research area and document the analysis, which can help future researchers conducting advanced studies on the same subject.

Chapter 2: Literature Review

2.1 Introduction

Idealists for decades have anticipated that small computers will eventually be directly intertwined into everyday structure of our lives, and a world hugely inhabited with "smart objects" ascending to "global computing" (Weiser, 1991). "Things that think" Gershenfeld (1999) and "The Networking economy" (Kelly, 1997).

Power of processing has turned into something that is less expensive and abundant. There has been a dramatic increase in the speediness of micro controllers (Moravec, 1999). The total production of micro controllers in the year 2000 surpassed the global population (Tennenhouse, 2000). At a massive speed these little chips are embedded into daily devices like toys, watches, smart cards, traffic lights etc. Evidently processing power at disposal is not the restricting element. Languages utilized for micro controllers have also multiplied. JINI (sun, 2000), "Thin clients" (Emware, 2000), Mobile agents (Minar, Grey, Roup, Krikorian, and Maes, 1999) and many other computationally trivial languages have been created for the purpose of aiding committed applications in embedded objects and devices.

The consistently declining costs of micro controllers have culminated in scenarios where the price of a single connector can surpass the price of the micro controller it attaches (Abelson, 1995). Industry standards like IEEE 802.11 (IEEE, 1999), Bluetooth (Bluetooth, 1999) and IrDA (Irda, 1998) have produced wireless interconnect systems that are less costly than wired systems. Since substantial utilization of semi-conductor technologies are exercised by these wireless technologies, they relish low prices and enhanced communication rates.

2.2 Embedded Networks

Embedded networks are networks that are built on embedded processors giving communication links for certain applications.

A common embedded microcontroller functions in comparative separation, incapable to attract information or exercise any authority besides its present sphere. Though it has power of computing it is similar to an intellectual with brightness and capability but without sensory

contributions and a medium to communicate what it knows. Hence there is far reaching but senseless computing and things that think but unable to communicate or relate (Dunbar, 2001).

For little embedded microcontrollers in typical devices, the price of communication stays comparatively inflated. Digital networks were initially created for the purpose of inter connecting main frame and mini computers, but for some reason amended rather inappropriately for connecting PC's and laptops. For embedded processors these legacy networks are unsuitable: they are high priced since a lot of power is utilized and they do not blend suitably in managing thousands of connections needed in a global sphere of things that think.

The absence of beneficial networking technologies for embedded microcontrollers has somewhat hindered the rise of smart devices and objects. What's needed is a contemporary model of embedded networks particularly created for the interconnection of microcontrollers.

A network needs to accumulate crucial mass for the purpose of being beneficial. As argued by Metcalfe's law, "The value of a network rises as the square of the number of devices connected". Since the volume of embedded microcontrollers is increasing quickly across the globe, the dependable method in order to accumulate and sustain crucial mass is that the network connection should be placed directly on the chip. Simon (1989) argues that it is beneficial to evaluate technology as "an interface" between an inner environment, the substance and the organization of artifact itself and an outer environment and the surroundings in which it operates. Embedded networks are incorporated into embedded processors, and gives communication linkages for particular applications in a relationship. These situations commands the intrinsic design needed for embedded networks.

2.2.1 Restrictions Enforced by the Host

A set of restrictions enforced by the microcontroller on which the embedded network node is built its (inner environment). A single microcontroller infrastructure is appropriate for a wide variety of applications, since the strength of microcontrollers resides in their all-inclusiveness. For practical embedded networking, it should be accommodating and flexible for a wide variety of applications. Because the embedded network system itself rests on the microcontroller chip, it should not enforce crucial strain on the chip. This can give rise to following design principles.

i. **Moderate power utilization:** An embedded network node must not surpass the power utilization of the microcontroller in order to be an appealing nominee for assimilation onto a microcontroller. One of the outcomes of Moor's law the argument that the number of transistors per unit area of assimilated circuit doubles every 18 months is that of decreased power. Since small devices have moderate exploring functions resulting in decreased switching currents (Carvey, 1996).

ii. **Little Silicon foot print:** The fabrication price of silicon microcontroller is relative to size. Little chips have elevated gains and they can be crammed on an individual silicon wafer. The circuitry that executes embedded networking should balance a small percentage of total chip size for the purpose of maintaining economical prices. This aids networking algorithms with calculation and computational clarity and small routing tables.

iii. **Modern Computational Overhead:** Since computing utilizes power, networking algorithms that needs reduced computation are appropriate for broad variety of applications particularly those having restricted obtainable power.

2.2.2 Restrictions Enforced by Applications

Dunbar (2001) in his doctoral thesis argues that Embedded networking's "outer environment" enforces another kind of design restrictions. Things that think have intertwined into our daily surroundings, always willing to provide service whenever needed and dwindling away into background whenever it is not needed. The networks connecting these devices fabricate their own communication webbing in the absence of prior arrangements, deliberate planning or preservation. If assistance is required by a device it is because of the urgency that is specific to the application and not because of network shortcomings. The important design fundamentals for embedded networks can be outlined as:

i. **Swift Infrastructure:** It is unwarranted to require people to manage and configure a network of things that think. An embedded network is expected to provide its services to its users and not the opposite. This needs a network system that is built upon

requirements and instinctively reconfigures itself as devices are connecting too and disconnected from the network.

ii. **Self-Configuring Nodes:** Correctly devised things that think have networks that are assembled inside and not appended on, which shows in their utilization: devices are united into a network merely by moving them manually into the network surroundings. Network must brace active and forceful recuperation and routing for the purpose of accessible network services even if devices are shifted or go offline.

iii. **Casual Arrangement:** Traditional wireless networks are cautiously devised giving consideration to its settings, utilization structure, and density. By comparison, volume and density of an embedded network is not normally known in advance. The blueprint of an embedded network must assist a wide variety of viable device configurations, from rare to numerous and from limited to dense.

An embedded network therefore is a relatively contemporary model in networking and provides various advantages over traditional networks.

i. **Proxy Intellect:** A clock must have awareness of how to set itself. A toy must be capable of identifying its owner's speech. No specific intellect is needed in the clock or toy when an embedded network connects these devices to alternative computational services.

ii. **Data Accumulation:** Modern day computers are branded as "deaf and dumb" (Pentland, 1998), sensorial devoid, and lack capability to behave rationally. Embedded networks can be utilized to accumulate unprocessed data from a variety of sources for the purpose of processing it into valuable information.

iii. **Low Cost Links:** In various situations, the price of links can be a noteworthy element of complete system funds. For instance price of a power switch can be 5 dollars but the price of copper wires, channel, and installation and implement time frame can elevate the

installation price in excess of 50 dollars. In a variety of scenarios embedded network links can present less costly substitutes to wired links.

2.3 Error Detection

Errors are described as actions and responses that transpire in an unwanted imbalance between presumed and actual situation leading to undesirable results that may have been eluded (Zhao and Olivera, 2006).

Technopedia describes error detection in the context of networking as "techniques utilized for the detection of noise or other obstructions initiated into data while it is transferred from sender to receiver." Error detection assures the reliable transportation of data across susceptible networks. Error detection also decreases the possibility of sending erroneous frames to the receiver, called undetected error probability.

William (1993) describes the goals of an error detection approach as one which helps the receiver of transferred data via noisy channel to establish whether the data has been altered.

Forouzan (2007) argues that networks should have the capability to transmit data from one device to another with reasonable precision. For a majority of applications, a system is expected to assure that the received data and transferred data are in uniformity. Every time data is transferred from one device or node to the following, there is a possibility of alteration in transit. Various aspects can change one or more data bits. Some applications need a procedure for the purpose of detection and correction of errors.

A number of applications can handle a restricted magnitude of errors. For instance in transmitting audio or video random errors can be acceptable, however when text is transmitted a precision of high magnitude is desirable.

In transmission systems, an error transpires when a bit is changed while in transit from transmitter to receiver. This means that a binary 0 is received when a binary 1 is sent.

Analysis of Stallings (2008) suggests that regardless of the design of transmission system, there will be errors, resulting in the alteration of one or more bits in transmitted frame.

Assumption is that transferred data is a single or a number of infectious flows of bits, known as frames. Stallings describe these possibilities in regards to errors in transferred frames:

P_b: Probability of receiving a bit in error, also called bit error rate (BER)
P_1: Probability of a frame arriving with no bit errors.
P_2: Probability, of arrival of a frame with one or many undetected errors, with an error detecting algorithm in place.
P_3: Probability, of arrival of a frame with one or many detected bit errors but no undetected bit errors, with an error detecting algorithm in place.

Stallings assumes few scenarios. The first one is where no steps are taken for detection of errors. In this case the possibility of detected errors P_3 is zero. In order to describe other possibilities the assumption is that any bit is in error P_b is steady and free for each bit:

$$P_1 = (1 - P_b)^F$$
$$P_2 = (1 - P_1) \qquad \text{(Equation 2.3)}$$

F is the number of bits per frame.

Possibility of a frame received with no bit errors declines where the possibility of frame arriving with one bit error rises. Also as the length of the frame expands the possibility of frame arrival with no bit errors declines. As there are more bits in a long frame and therefore high probability of errors in bits.

2.3.1 Types of Errors

There are two common kinds of errors that transpire in data transmission: single and burst bit errors. Single bit error is a solitary constraint that corrupts one bit but has no influence on neighboring bits. In a burst error, an error of length *L* is an infectious flow of *L* bits in which first and last bit including any number of in between bits are received by the receiver in error.
IEEE std100 and ITU-T Recommendation describes Burst errors as:

"A set of bits in which two consecutive fallacious bits are always segregated by less than given number *x* of error free bits. The last fallacious bit in the burst and the first fallacious bit in the next burst are consequently segregated by *x* error free bits or more."

Therefore it can be said that in a burst error, there is a congregation of bits in which a number of errors happen, though it is not a certainty that all of the bits in the congregation experience an error.

Single bit errors can transpire in existence of white noise. A small casual distortion of signal to noise ratio can affect the receiver's judgment of a single bit. Burst errors are more frequent and more strenuous to handle. Impulse noise can also produce burst errors. At high data rate the influence of burst error is stronger. (Stallings, 2008)

Single bit error implies that during transfer of data from sender to receiver only one bit has altered (Kaisi and Kitakami, 2002). It has altered from either 0 to 1 or 1 to 0. This one bit error can't be neglected, because there is a strong possibility that alteration in one bit can alter the complete connotation of the message. For instance an 8 bit message 00001010 illustrates the beginning of text, but subsequent alteration happens and the message changes to 00000010. This altered message will be wrongly understood as line feed, which is obviously in conflict with the actual message transferred by the sender.

In serial data transmission this kind of error is less probable. For instance consider that data is transmitted at 1 Mbps. This means that every bit will remain merely for 1 / 1000000s. For single bit error to transpire noise is expected to have a period of 1us. This is highly unusual since noise typically remains in excess of 1us.

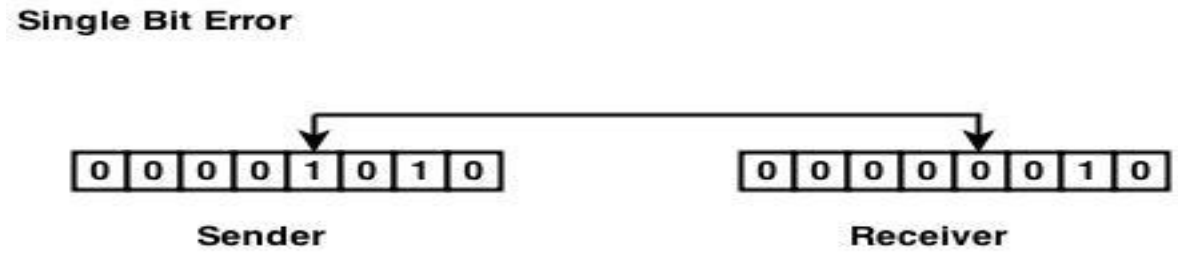


Fig: 2.3.1(a). Single bit error illustrating the transformation of 1 to 0, from sender to receiver.

Burst error means that while transferring data from sender to receiver two or more bits are in error. Due to distortion in the channel alteration can happen and bits can be changed from 0 to 1 or 1 to 0 (Kaisi and Kitakami, 2002). In a burst error the bit error length is calculated from

the first bit where the error appeared to the last bit of altered bits, even if various bits in between the errors are not altered.

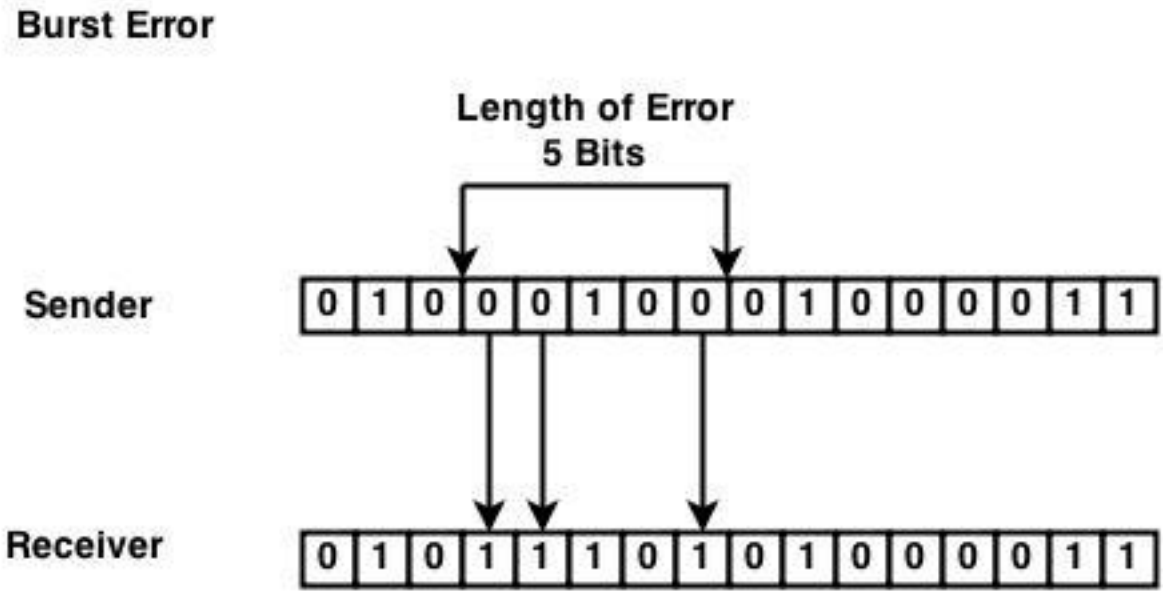


Fig: 2.3.1(b). Burst error with length of 5 bits transforming a message from sender to receiver.

2.4 Checksums

Merriam Webster dictionary define checksums as "a sum obtained from the bits of a chunk of computer data that is computed prior to and after transmission and storage, to ensure that data is independent from errors or manipulation".

Oxford dictionary define checksums as "A digit or digits depicting the sum of accurate digits in a chunk of stored or transmitted digital data against which later comparisons can be made for detecting errors in the data".

According to Technopedia a checksum is an error detection mechanism in which a sender calculates a numerical value in proportion to the number of set or unset bits in a message and transmits it along every message frame. On the other end, the receiver implements the same checksum mechanism in order to extract the numerical value. If the checksum value received is equivalent to the value that is transmitted than the transmission is deemed to be correct and independent of errors. An unequal checksum confirms that the transferred message is not complete.

The approach of extracting a checksum from messages is known as checksum function and is accomplished by utilizing a checksum algorithm. Effective checksum algorithms yield conflicting outcomes with large possibilities, if there are alterations in messages. Specific checksums appropriate for small blocks of data are parity bits and check digits. Based on checksums particular error detection codes have the competence of retrieving authentic data.

A typical method of safeguard the coherence of data message is to add a checksum. Even though it is a fact that CRC is a powerful error detecting code, a number of embedded networks utilize the services of less powerful checksum mechanisms in order to decrease computation costs in high restricted systems. In certain situations this cost/performance balance is understandable, but at times designers give up error detection benefits without acquiring corresponding advantages in calculation speed surge or memory footprint deduction.

According to Peterson and Davie (2007) Checksum approach is an uncomplicated approach build on the addition of all the words and then transferring them containing the outcome of that addition. Checksums are built on the idea of redundancy similar to parity checks and CRC. Data is subdivided into identical chunks of *n* bits by checksum generator. These chunks are summed utilizing binary arithmetic in a manner that the length of the result is also *n* bits. Result is then attached to the end of the authentic message as redundant bits, known as checksum field. Receiver takes similar approach and applies the same procedure on the data received and matches the outcome with the checksum received. In case the outcome of the computation is zero, the receiver retains the transferred data. In case the result is not zero, receiver rejects the data concluding that an error has transpired.

For the purpose of error detection, how the sum of data and checksum are compared is perhaps the only distinction between various executions of a checksum approach. In case of parity, which approach to utilize is the prerogative of the designer. The kind of checksum utilized should be decided and agreed before time by both sender and receiver devices. Some of the distinct kinds of checksum execution approaches are as follows:

i. A chunk of data is judged as independent of errors if the sum of data matches the checksum. In this scenario the checksum component is calculated by adding all the data chunks and throwing away any carries.

ii. A chunk of data is judged independent of errors if the sum of data and checksum concludes in binary value with all ones. In this scenario the checksum component is calculated by taking 1's complement of data sum. This approach is known as 1's complement checksum.

iii. A chunk of data is judged independent of errors if the sum of data and checksum concludes in a binary value with all zeros. In this scenario the checksum component is calculated by taking 2's complement of data sum. This approach is known as 2's complement checksum.

Networks that utilize CRC for assuring the coherence of data include Flex ray (flex ray consortium, 2005), Controller area networks (CAN) (Bosch, 1991), and TTP/C (TTTechcomputertechnik, AG, 2003), nevertheless the utilization of less effective checksums are in abundance. A broadly utilized substitute checksum is exclusive or (XOR) utilized by HART (The HART book), Magellan (Thales Navigation, 2002) and various networks that are purpose built (Lian, Zang, Wu, and Zhao, 2005). Two's complement addition checksum is other broadly utilized substitute utilized by XMODEM (Christensen, 1982), Modbus ASCII (Modicom Inc. 1996) and few communication protocols (Mcshane Inc.).

Besides these choices, apparently more appropriate substitute approaches to checksums are utilized by non-embedded networking. Non embedded checksums analyzed as substitutes are the one's complement addition checksum in TCP (Braden, Borman, and Partridge, 1988), Fletcher checksum utilized as a TCP substitute checksum (Zweig, and Partridge, 1990) and Adler checksum (Deutsch, and Gailly, 1996). Though these checksums seems to offer possible enhancements for embedded applications, so far they are not broadly utilized in embedded applications.

2.4.1 XOR Checksum

National Institute of Standards and Technology (NIST) defines XOR as: "Exclusive Or" or "not equal to" function: 0 XOR 0 = 0, 0 XOR 1 = 1, 1 XOR 0 = 1, 1 XOR 1 = 0.

According to Thomsen (1983), "XOR is a rational operator whose output is true when either of the two inputs is true but not both. The output differs in relation to the comparative

attributes of the inputs, and by maneuvering this the device can be made to perform the functions of all types of logic gates utilized in computer circuitry (OR, AND, NOR).

Oxford dictionary defines XOR as "A Boolean operator working on two variables that have the value one if one but not both of the variables is one". It further states that 'a circuit producing output signal when a signal is received through one and only one of its two inputs' also known as exclusive OR gate.

Exclusive Or (XOR) checksums are calculated by jointly XOR'ing chunks of data word. Checksum value is not influenced by the sequence in which chunks are processed. XOR checksum can be considered as a parity calculation executed concurrently through each bit location of data chunks (Bit *y* of the checksum is the parity of all chunk bits *y*). For instance bit 4 of the checksum is the parity of bit 4 of all chunks.

XOR is not dependent on data. Which implies that values of data word doesn't influence the performance of error detection. XOR checksum has a hamming distance (HD) of 2 which detects all one bit errors, but not some 2 bit errors. Especially it is not successful in detecting bit errors that are even and transpiring in the same bit location of the checksum computational chunk. It has the capability of detecting any bit error structure that concludes in an odd number of errors in minimum at one bit location, comprising all scenarios where the entire number of bit errors is odd. It also has the capability of detecting all burst errors with length up to *k* bit size (*k* equivalent to size of checksum), for the reason that in order to become undetected, bits must line up in the same location inside a chunk. Burst errors that are larger than *k* bits in length can be detected if they conclude in an odd number of authentic bits being reversed or if the reversed bits do not line up in the same bit location in altered chunks. (Maxino, T. 2006)

Supposing a chunk size of *k* and checksum size *k*, in each set of data chunks there are *k* probable 2 bit errors that are not detected (Single 2 bit error which is undetected, error transpires in the same bit location in two chunks) for a code word of *n* bits, multiplication is done by combining numbers of *k* size chunks in the code word taking two at once. Therefore undetected 2 bit error is:

$$k * \binom{\frac{n}{k}}{2} = \frac{k\left(\frac{n}{k}\right)\left(\frac{n-k}{k}\right)}{2} = \frac{n(n-k)}{2k}$$

For an *n* bit code the result of the probable 2 bit error is:

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

Division of undetected 2 bit errors by entire number of 2 bit errors gives the fragment of 2 bit errors that are undetected:

$$\frac{n-k}{k(n-1)} \qquad \textit{(Equation 2.4.1 a)}$$

n is the length of code words in bits, *k* is the checksum size in bits.

It can be concluded that as *n* moves towards infinity, the ratio of 2 bit errors that are undetected are roughly *1/k* (*k* is size of checksum), which is relatively inferior performance for checksums. For instance for an 8 bit checksum 12.5% undetected errors and for a 32 bit checksum 3.1% undetected errors. XOR checksum has large possibilities of undetected errors. It is not considered as beneficial as some of the other addition checksums utilized for the purpose of common use in error detection (Koopman and Maxino, 2009).

XOR TRUTH TABLE

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 0$ *(Equation 2.4.1 b)*

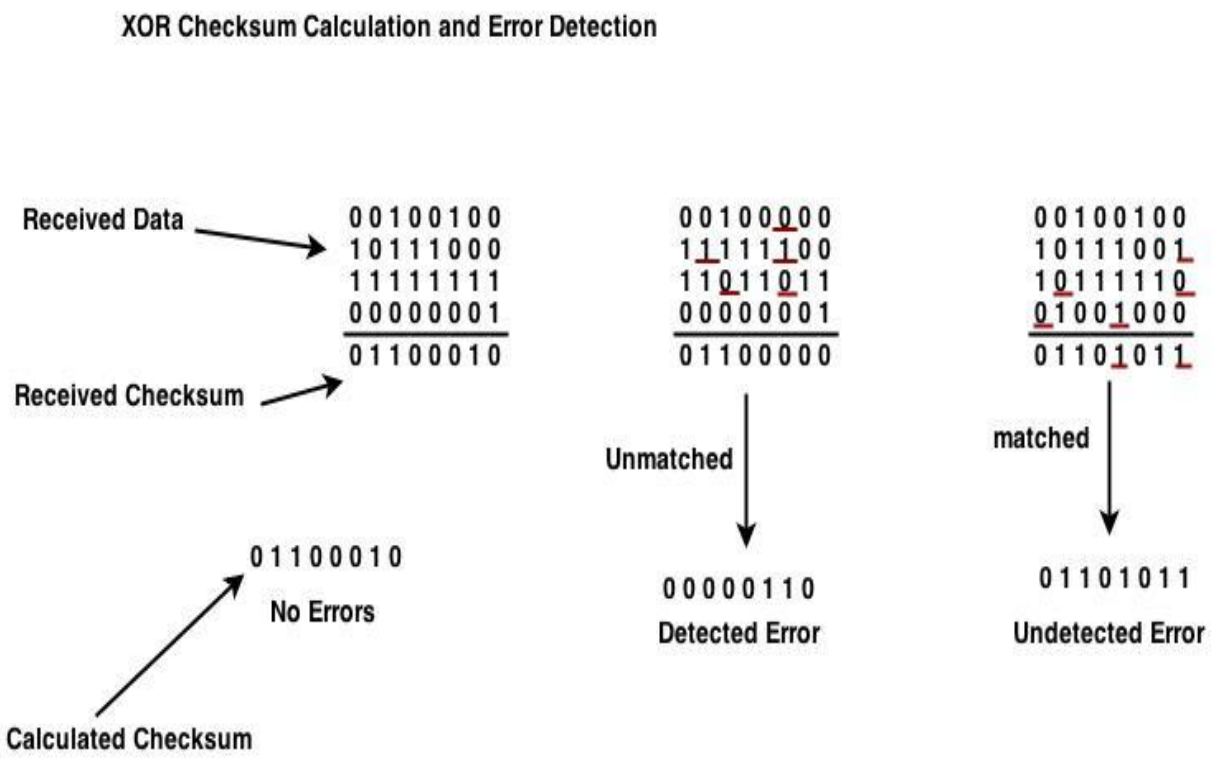


Fig: 2.4.1. Illustration of XOR calculations (Detected & Undetected errors.)

2.4.2 One's Complement Checksum

One's complement addition checksum is acquired by conducting an integer one's complement addition of total chunks of data word. One's complement addition can be applied on two's complement hardware by "wrapping" the carry out of the addition back into the checksum. The running sum is supplemented especially if adding a chunk to the total of the running sum culminates in a carry out. Effective utilization of speed is widely known for hardware that doesn't assist carry bits (Braden *et al*., 1988). The sequence of chunk processing doesn't influence the value of checksum.

Error detection competence for inverted bits influencing the most significant bit of the chunk is the primary performance distinction between one's and two's complement addition checksums. Owing to the fact that carry out data of the most significant bit is maintained and incorporated back to the least significant bit, a set of one's or a set of zero's that are influenced by inverted bits in most significant bit are detected by one's complement addition but are

undetected by two's complement addition (Plummer, 1989). Burst errors of (k – 1) bits length (k is checksum size) are detected by one's complement addition. Since the carry outs are added back in the least significant bit, some k bit burst errors remain unidentifiable. Besides the performance of error detection in the location of most significant bit, the conduct of one's and two's complement checks are more or less synonymous. Though for non-dependent random bit errors, one's complement has moderately small possibility of errors that are undetected.

For lengths that are asymptotic, the possibility of errors that are undetected for all zero's and one's moves to 2 / n (n length of codeword's).

Roughly 1 / 2k of all probable 2 bit errors are detected in cases with unsatisfactory data at asymptotic lengths (k checksum size). This is 1 / 2 the percentage of errors that are undetected in XOR checksum. The innate rationale that needs to be acknowledged is that for every bit location, error sets that are undetected will be 0-1 and 1-0. This is opposed to XOR checksum where undetectable error sets also include 0-0 and 1-1 (Schwiebert and Jiao, 2001).

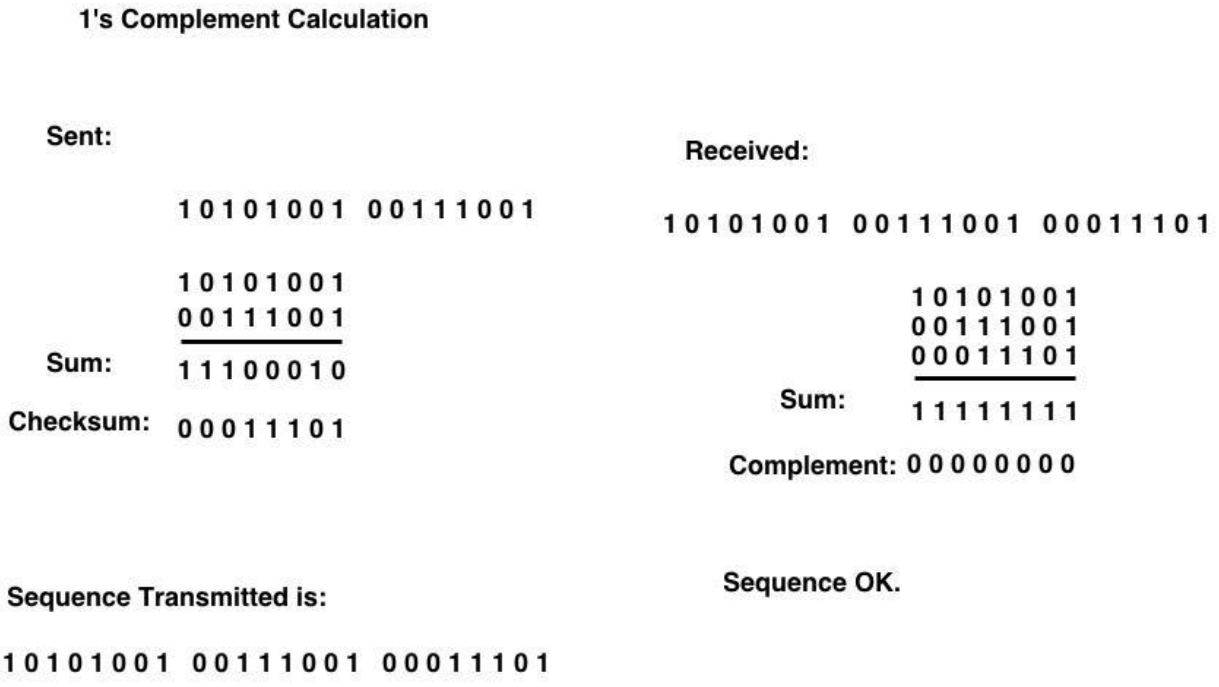


Fig: 2.4.2(a). Illustration of 1's Complement Checksum

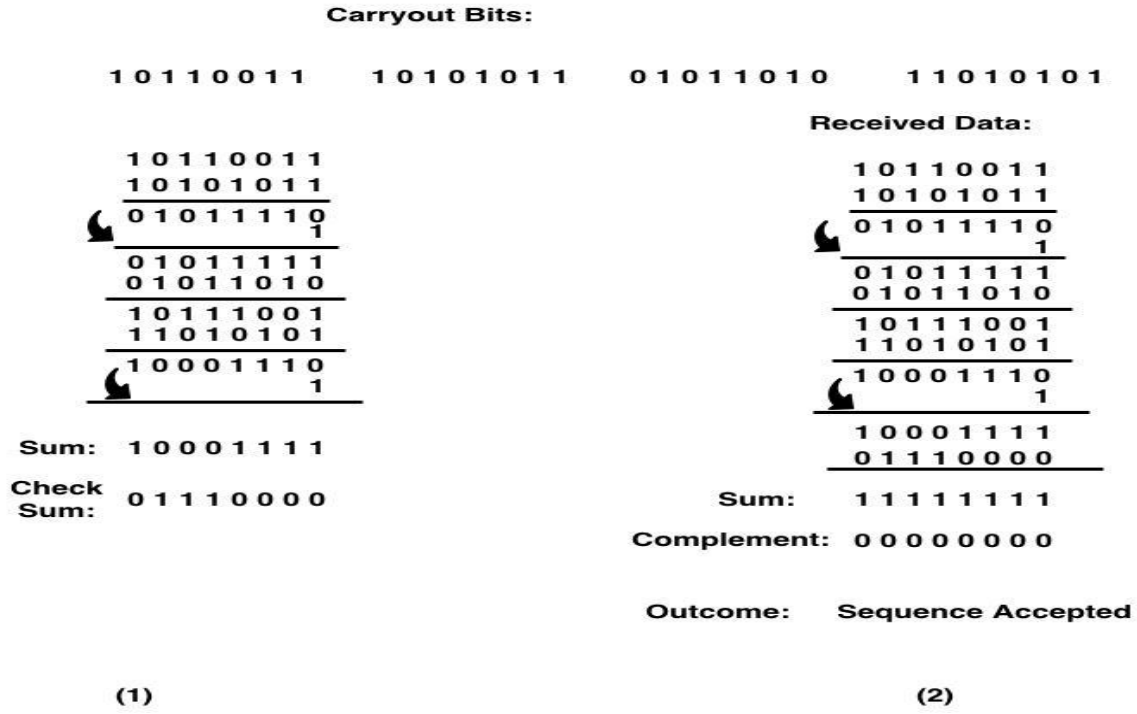


Fig: 2.4.2(b). Illustration of 1's Complement Checksum (Carry Bits)

2.4.3 Twos' Complement Checksum

In order to acquire two's complement addition checksum an integer two's complement addition of all the chunks in the data word is performed. Carry out's of the acquired summation is thrown away similar to the usual single accuracy integer addition. The value of the checksum is not influenced by the sequence in which chunks are handled. The possibilities of errors that are undetected fluctuate with the value of data word, because addition checksum is dependent on the data. For entire lengths of code word addition checksum has a Hamming distance of 2. This implies that it can detect all one bit errors in the code word (Koopman and Maxino, 2009).

Addition checksum can be considered an enhancement of XOR checksums in the sense that bit "blending" between bit locations of data chunks is achieved through bit by bit carries of binary addition. The usefulness of the "blending" relies on the data being added, which establishes the structure of carry bits through different bit locations.

An important reason for unidentified errors is when a set of bit errors in various data chunks align at the same bit location in the chunks and the data in those bit locations comprises

of 1 in one chunk and a 0 in other chunk. The outcome is that the inaccurate data chunks have a zero in the first chunk and a 1 in the other chunk, concluding in the same sum.

A second important reason of unidentified errors is when irrespective of value, the most significant bit (MSB) locations of any two data chunks are reversed. The reason that this kind of error remains undetected is that the sum stays the same and the information regarding carry out is neglected in calculation. This makes it very difficult to identify a set of one's altered to zero's or a set of zero's altered to one's in the most significant bit.

A third important reason of unidentified errors is when a bit that doesn't generate a carry is reversed in the data word and the bit in the following bit location in the checksum is also reversed.

The performance of addition checksum is unsatisfactory when every bit location comprises of equivalent number of zero's and one's. The rationale behind this is that error detection susceptibilities that are dependent on data entail a simultaneous inversion of one and zero bits in the same bit location. On this basis unsystematic data provides almost a worst scenario for unidentified errors, since in every bit location it is inclined to have the same number of zero's and one's. Considering that unsystematic data is frequently utilized for the purpose of analyzing checksums, however actual data transmitted in network messages frequently have a firm prejudice for zero's owing to unutilized data fields, for various scenarios unsystematic data analysis of addition checksums can be deemed as doubtful. The performance of addition checksum is most useful when data is all one's or all zero's, the reason being that reversing a set of matching bits, results in a carry bit impact which is detected easily (Maxino, 2006).

Even for worst situation data, addition checksum is nearly twice as useful as the XOR checksum for large data words. This is due to the fact that the fundamental reason of unidentified errors is reversed bits that are both in conflict and their bit location is similar. On the other hand unidentified errors in XOR also transpire for bit values that are not essentially in conflict.

In worst situation data, ratio of unidentified two bit error of addition checksum is nearly equivalent to $\frac{(k+1)}{2k^2}$ (*k* checksum size). This calculation can be achieved by mutually summing the ratios of undetected errors for every bit location. Undetected error ratio of most significant bit is

equivalent to that of XOR, *1 / k.* Undetected error ratio for all alternate bits is 1 / 2 of XOR (*1 / 2k*) for the reason that only 1-0 and 0-1 error compositions will be unidentified. Multiplication of both percentages by number of bits in every chunk and then summing them jointly results in:

$$\frac{1}{k}\left(\frac{1}{k}\right)+\frac{1}{2k}\left(\frac{k-1}{k}\right)=\frac{1}{k^2}+\frac{k-1}{2k^2}=\frac{k+1}{2k^2} \quad \textit{(Equation 2.4.3)}$$

But, this calculation is merely an estimate since it doesn't consider the third reason of unidentified errors indicated earlier. It also doesn't acknowledge the fact that some of the error compositions comprising of 0-1 and 1-0 will be detectable because of generating carry bit. Nevertheless it is a beneficial estimate and can be viewed as a step forward.

Addition checksum is unsuccessful in detecting nearly $\frac{1}{k^2}$ of two bit errors for large data words with all zero's or all one's. (*k* checksum size in bits)

All burst errors with *k* bits length (*k* checksum size) are detected by addition checksum. Whether burst errors larger than *k* bits can or cannot be identified relies on the number of reversed bits and their location. The same rationale for unidentified bit errors is relevant for burst errors. Therefore, if a burst error larger than *k* bits transpires however the reversed bits do not have the same bit locations and do not belong to any of the three groups of unidentified errors discussed previously than it is improbable that the burst error will be undetected (Koopman and Maxino, 2009).

2.4.4 Fletcher Checksum

Fletcher checksum is an algorithm utilized for the purpose of calculating a checksum that is location dependent. It was conceived by John Fletcher in the later part of 1970's. The basic aim of Fletcher checksum was to offer error detection properties closer to cyclic redundancy codes (CRC) with relatively less calculation strains related to summation approaches (Fletcher, 1982).

Though Fletcher checksum is solely described for 16-bit and 32-bit checksums (Zweig and Partridge, 1990), however fundamentally calculations can be done for any chunk size with an even number of bits. One's complement addition model of Fletcher checksum offers superior error detection as opposed to two's complement model (Nakassis, 1988).

In order to calculate Fletcher checksum there is a chunk size *j* which is 1 / 2 the size of checksum *k* (for instance, a 16-bit Fletcher checksum is calculated with a chunk size of 8-bits through the data word, generating a checksum value of 16-bit). For calculating a checksum repeating through a pair of chunks from D_0 to D_n the algorithm utilized is:

$$\text{Primary Values: } \text{Sum A} = \text{Sum B} = 0$$
$$\text{For incrementing i: [Sum A} = \text{Sum A} + D_{i;}$$
$$\text{Sum B} = \text{SumB} + \text{Sum A;}]$$

(Equation 2.4.4)

Utilizing the same chunk size *j*, both Sum A and Sum B are calculated. The concluding checksum is twice the size of the chunk by concatenated Sum B with Sum A. The accumulation of Sum B turns the checksum susceptible to the sequence in which chunks are processed.

Error detection properties of Fletcher checksum are dependent on the data. The elevated possibility of unidentified errors materializes when the data in every bit location of the chunks is evenly distributed between zero's and one's. This is similar to addition checksums. Nearly worst scenario error detection performance is provided by unsystematic values of data word because of a comparatively even division of zero's and one's in every bit location. In cases where the data is all zero, the sole error that is unidentified is the one in which all bits in a single chunk are transformed from zero's to one's.

Burst error with a length less than $j\left(\frac{k}{2}\right)$ can be detected by Fletcher checksum. *(j* is chunk size, ½ of checksum size *k*). Fletcher checksum is susceptible to burst errors that reverse bits in a chunk from all zero's to all ones or one's to all zero's. If this particular kind of burst error is removed from consideration, then burst errors less than *k* bits *(k*, checksum size*)* can be identified by Fletcher checksum (Fletcher, 1982).

Up to a specific code word length which is modulo-dependent, Fletcher checksum has a hamming distance of 3. Whereas for other remaining code word lengths the hamming distance is 2. Tests conducted by Maxino (2006) validates that two bit errors are identified for lengths of

data word less than $(2^{k/2} - 1) * (k/2)$ bits (*k* checksum size) and $(2^{k/2} - 1)$ equivalent to modulo of Fletcher checksum. Fletcher (1982) argues that if the segregation is less than$(2^{k/2} - 1) * (k/2)$ bits, all two bits errors are detected (*k*, checksum size).

Experiments conducted by Koopman and Maxino (2009) further validates that for data word lengths of 68 bits and beyond, Fletcher checksum has hamming distance of 2, and for code word lengths less than 68 bits a hamming distance of 3. A 16-bit Fletcher checksum beginning at 2056 bit code word length has a hamming distance of 2. As per the calculation, Fletcher checksum is anticipated to have hamming distance of 2 beginning at a code word length of 1,048,592 bits.

Commonly, Fletcher checksums are notably inferior to CRC, even in cases where both attain the same hamming distance. Especially, long length Fletcher checksum has notably high possibility of unidentified errors than $1/2^k$, on the other hand CRC normally has a vague low possibility of unidentified errors than $1/2^k$ (*k* checksum size). The importance of $\frac{1}{2^k}$ is that it is a frequently believed hypothesis that all checksums have similar error detection usefulness equivalent to $1/2^k$.

2.4.5 Adler Checksum

Developed by Mark Adler in 1995, Adler-32 is a checksum algorithm. It is a refinement of Fletcher checksum. When analyzed in parallel with CRC of a similar length, it swaps dependability for speed. Dependability of Adler-32 is more than Fletcher-16, and vaguely less than Fletcher-32 (Deutsch and Gailly, 1996).

Adler checksum (Deutsch and Gailly, 1996) is solely described for 32 bit checksums but theoretically can be calculated for any chunk size comprising of an even number of bits. Adler checksum is almost identical to Fletcher checksum. Fletcher checksum utilizes one's complement addition; integer addition modulo 255 is applied for 8 bit chunks and modulo 655,535 for 16 bit chunks. In a bid to achieve superior blending of checksum bits Adler checksum utilizes prime modulo. Apart from setting the primary value of Sum A to 1 and accomplishing every addition modulo 65,521 (for 32-bit Adler checksum) in place of 65,535,

Adler checksum is similar to Fletcher checksum. Same as Fletcher checksum, the outcome is vulnerable to the sequence in which chunks are processed.

While Adler checksum is not formally described for alternate data word lengths, in order to execute 8 and 16 bit Adler checksums for comparative reasons largest prime integers less than $2^4 = 16$ and $2^8 = 256$ can be utilized. Since Adler algorithm is almost identical to Fletcher algorithm, their performance properties are more or less identical. Experiments conducted by Koopman and Maxino (2009) verify that for data word length less than M * (k / 2) bits (*k*, checksum size and M equivalent to Adler checksum modulus) two bit errors are detected. Experiments further explains that Adler 8 under 60 bits has hamming distance of 3 (utilizing modulo 13 sums) and Adler 16 under 2,024 bits has hamming distance of 3 (utilizing modulo 251 sums). In light of the calculation less than 1,048,368 bits Adler-32 is anticipated to have hamming distance of 3. Adler checksum has hamming distance of 2 for code word lengths larger than those discussed earlier.

All burst errors less than j (k / 2) bits *(j*, chunk size 1 / 2 the checksum size *k*) can be detected by Adler 8 and Adler 16. Burst errors up to 7 bits in length are detected by Adler 32. Deutsch *et al*., (1996) describes Adler 32 chunks as 1 byte or 8 bits in width with running sums of 16 bits. Hence, for Adler 32 $j = 8$. Excluding burst errors that alter data in data chunks from all zero's to all one's and all one's to all zero's, all burst errors less than $k - 1$ are identified. In comparison to Fletcher checksum this is 1 bit less, which was somewhat unforeseen considering that both Adler and Fletcher checksums utilize a nearly similar arithmetical premise. Sheinwald, Satran, Thaler, and Cavanna (2002) asserts that compared to Fletcher checksum possibility of unidentified errors in Adler checksum is higher but doesn't explain precisely that burst error detection is 1 bit smaller in length. The rationale behind this is that the prime modulo utilized by Adler checksums is less than $2^k - 1$. On the other hand modulo utilized by Fletcher checksums is equivalent to $2^k - 1$ (*k*, checksum size).

Adler Checksum

$$A = 1 + D_1 + D_2 + \ldots\ldots + D_n \pmod{65521}$$

$$B = (1 + D_1) + (1 + D_1 + D_2) \ldots\ldots + (1 + D_1 + D_2 + D_n) \pmod{65521}$$

$$= n * D_1 + (n-1) * D_2 + (n-2) * D_3 + \ldots\ldots\ldots\ldots + D_n + n \pmod{65521}$$

$$\text{Adler } 32(D) = B * 65536 + A$$

D, the string of bytes for which checksum to be computed, n is the length of D.

(Equation 2.4.5)

2.5 Cyclic Redundancy Check (CRC)

Prange (1957) initiated the idea of CRC. In essence, CRC work by polynomial division of a dataword, executed by a 'generator polynomial' over Galois field (Ray and Koopman, 2006). A check pattern is provided by the remainder of that division. In reality, shift, XOR and table lookup methods prevail that allows the effective mapping of polynomial division onto software and hardware of computer. Peterson and Brown (1961) give a detailed early investigation of the issue.

A list of methods is given by Feldmeier (1995) for swift implementation of CRC's and checksums and provides a comparison of calculation swiftness. A dataword of 64 bit length was utilized and swift comparisons were done on a SPARC CPU, which may be more representative of desktop processors than embedded processors.

Ray and Koopman (2006) analyze swift methods for CRC calculations, and provide CRC polynomials selected particularly to assist swift software calculations. Polynomial assessment is provided depending on HD minimum hamming weight at the first non-zero HD (which anticipate Pud for error rates that are low) and polynomial patterns appropriate for elevated speed software application). Braun and Waldvogel (2001) provide a method for doing augmented modifications to CRC's when a small part of message alters.

Wheal, Miller, Stevens, and Lin (1986) examined the utilization of a possibility of unidentified error (Pud) design for assessing CRC's.

Fujiwara, Kasami, Kitai, and Lin (1985) provide a performance evaluation for shortened hamming codes (not CRC's) that set the stage for early research in CRC performance evaluation. They exhibit the possibility of unidentified errors for some codes for distinct sizes of dataword for BER value 0.5 to 1e-7. They then provide a summarized comparison that check for maximum unidentified error rate for a BER up to 0.5, which for an embedded network is exceptionally inflated BER, but substitutes for a coding scheme with mathematical worst scenario.

Fujiwara, Kasami and Lin (1989) establishes the HD values of IEEE 802.3-2000 standard polynomial for codeword lengths up to the top Ethernet transmission unit length of 12144 bits. The assessment was applied for fluctuating BER values for both HD and Pud.

Daniel, Costello, Hagenauer, and Wicker (1998) gives a general view of error control codes and how CRC's and checksums correlate to the entire sphere. A small portion of CRC recommends selecting polynomials that are primitive multiplied by a factor of (x+1) in order to make sure all weight error structures are identifiable. It is noteworthy that both CRC and checksum capture $\left(1-2^{k}\right)$ errors for a k bit check pattern.

Sheinwald *et al.,* (2002) contemplates the utilization of CRC from Castagnoli, Brauer, and Hermann (1993) compared to Fletcher and Adler checksums. They deduce that for a specific set of presumptions a CRC has 12,000 times enhanced error identification ability than a Fletcher checksum and 22,000 times enhanced error identification ability than Adler checksum. They also relate checksums and CRC's in accordance with the specification metrics per dataword byte to calculate, susceptibility to data values (checksum Pud is susceptible to data values, CRC Pud is not) and size of lookup table. Comparisons were done for both a low noise medium and for burst errors.

Another research was publicized by Baicheva, Dodunekov, and Kazakov (2000) recognizing better 16 bit CRC's in accordance with the same assessment standards as Baicheva *et al.,* (1998). 16 bit CRC polynomials were recognized that were ideal for every dataword size. Only some polynomials with specific factorizations were examined.

A research of CRC polynomials utilizing minimum HWs as the assessment standard was also publicized by Kazakov (2001). The outcome comprises a polynomial that is optimal for each dataword length from 255 bits to 32768 bits.

Koopman (2002) publicized an extensive research of all probable 32 bit CRC values to discover the polynomials that give HD=6 at the largest probable value. The polynomial recognized was not in the search space commonly addressed founded on encouraging polynomial factorizations, and provoked additional work with extensive research in subsequent publications. The assessment standard for this research was founded on HD.

Koopman and Chakravarty (2004) subsequently publicized a research that provides optimal CRC polynomials for lengths of 4 through 16 bits. The assessment standard was founded on HD and lowest HW at that HD. An extra standard was to recognize a polynomial that had an even better HD at smaller dataword lengths. This was to enhance error identification effectiveness for CRC's utilized for network messages, which can comprise of numerous small messages (which would then attain high HD) and less long messages (which would get a less, but optimal HD) in embedded systems.

A difficulty with utilizing a checksum for error correction resides in its easiness. In a stream of data if various errors transpire they have a probability of canceling each other out. For instance a one bit error can minus four bits from checksum, and another error adds four. If a checksum is 8 bits wide, then in a stream of data there are $2^8 = 256$ potential checksums. This implies that there is a possibility of 1 in 256 multiple errors that may not be identified. This likelihood can be minimized by widening the checksum size to 16 or 32 in order to elevate the potential number of checksums to $2^{16} = 65{,}536$ or $2^{32} = 4{,}294{,}967$.

What is required is an approach for error detection that produces broad distinct values for small data errors.

A cyclic redundancy check (CRC) utilizes a fundamental binary algorithm where every bit of a data component adjusts the checksum through its complete length irrespective of the number of bits in the checksum. This implies that an error at bit level adjusts the checksum so remarkably that an equivalent and opposing bit alter in other data component is unable to cancel the impact of the first.

Encyclopedia of PC magazine describes cyclic redundancy checking as "an error checking approach utilized for the purpose of assuring the preciseness of transferred digital data. The transferred messages are separated into pre-arranged lengths, which are utilized as dividends and divided by a set divisor. The remainder of the computation is added onto the transferred message. The remainder is recomputed on the receiver's end. An error is detected if the recomputed remainder doesn't match the transferred remainder.

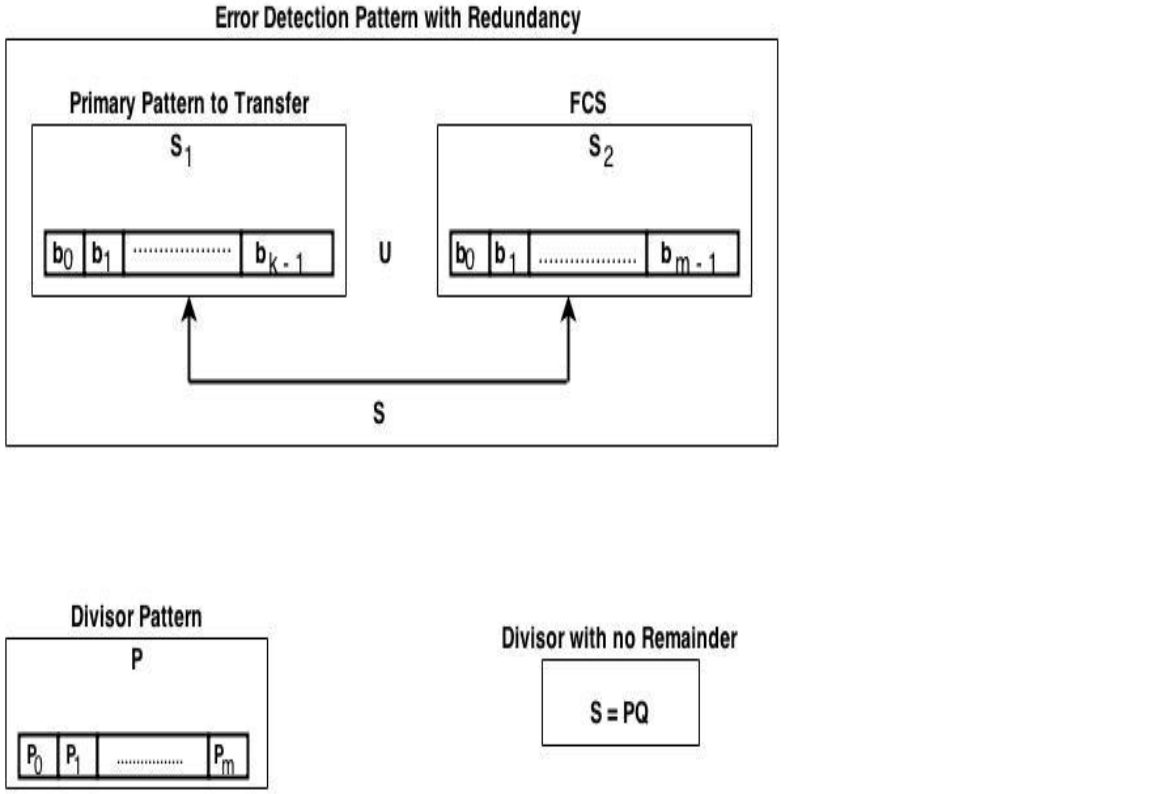
Koopman and Chakravarthy (2004) argues that a CRC can be considered as a non-secure digest function for a data word that can be utilized for the purpose of detecting data alteration. They further state that within the context of mathematics a CRC can be explained as tackling a binary data word as a polynomial over GF(2) (every polynomial coefficient 0 or 1) and applying polynomial division, frequently known as a CRC polynomial. The remainder of the division exercise gives an error detection value that is transmitted as a frame check sequence (FCS) inside a message or maintained or preserved as data integrity check. Whether applied in hardware or software, CRC calculations acquire the shape of a bitwise convolution of data word opposed to binary method of CRC polynomial. Size of the data word is the data shielded by CRC, however doesn't incorporate CRC itself. Application of error detection is done by making comparisons between value of FCS calculated on data with the value of FCS initially calculated and transmitted or preserved with the initial data. An error is acknowledged to have transpired in case the preserved value of FCS and the value of calculated FCS don't match.

Saxena and McCluskey (1990) argue that cyclic redundancy check (CRC) is broadly utilized in data communications and storage devices as a forceful approach for handling data errors. According to Smith (1998) CRC is also utilized for various other spheres, for instance integrated circuit testing and identification of rationale defects.

CRC is considered to be one of the most potent error detecting codes and can be explained as follows:

Assuming that a sender *T* transmits a pattern S_1 of *k* bits $[b_0, b_1, \ldots\ldots, b_{k-1}]$ to a receiver *R*. *T* produces one more pattern S_2 of *m* bits $[b_0, b_1, \ldots\ldots, b_{m-1}]$ in order to help the receiver in identifying probable errors. Pattern S_2 is frequently called Frame check sequence (FCS). It is produced by considering the verity that the entire pattern, $S = S_1 \, U \, S_2$ is derived by the concatenation of S_1 and S_2, and by adhering to a specific arithmetic having the property that is divisible by a pre-arranged pattern *P*, $[P_0, P_1, \ldots\ldots, P_m]$ of *m + 1* bits. Sender transmits pattern *S* to receiver. Division is performed by the receiver on *S* (*k* = message, *m* = FCS) by *P*, utilizing the same arithmetic. In case the remainder is zero, receiver concludes that there is no error. (Campobello, Patane, and Russo, 2003)

In order to digitally accomplish the notion modulo-2 arithmetic is utilized (Stallings, 2000). While addition and subtraction are achieved by bitwise XOR operator, a bitwise AND is utilized to achieve the product operator. In the given scenario a modulo-2 divisor can be achieved as a shift register for both sender and receiver. In sender's scenario, pattern *S1* is the dividend concatenated with a pattern of *m* zeros to the right. *P* is the divisor. Scenario for receiver is easier. Pattern received is the dividend and *P* is the divisor.



S: Pattern for Error Detection, S1: Primary Pattern for k bits to send, S2: Frame Check Sequence, (FCS) for m bits, P: Divisor, Q: Quotient.

Fig: 2.5. Illustration of Cyclic Redundancy Check (CRC).

2.5.1 Modulo 2 Arithmetic

Binary addition with no carries is utilized in modulo-2 arithmetic. This is simply an exclusive XOR function. Binary subtraction is also considered a XOR function (Stallings, 2008).

$$\begin{array}{r} 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1 \\ +\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1 \\ \hline 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0 \end{array} \qquad \begin{array}{r} 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1 \\ -\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0 \\ \hline 0\ 1\ 1\ 1\ 0\ 0\ 0\ 1 \end{array}$$

$$\begin{array}{r} 1\ 1\ 0\ 0\ 1 \\ \times\quad\quad 1\ 1 \\ \hline 1\ 1\ 0\ 0\ 1 \end{array}$$

$$\begin{array}{r} 1\ 1\ 0\ 0\ 1\ \times \\ \hline 1\ 0\ 1\ 0\ 1\ 1 \end{array}$$

(Equation 2.5.1)

In order to illustrate the execution of CRC, let us consider the following steps followed by an example of CRC calculation.

i. A single *P* bit divisor is determined. This *P* bit divisor is utilized by both sender and receiver.

ii. Data being transmitted is appended at the end by (*P* – 1) bits by the sender. XOR division is than applied for the purpose of acquiring the remainder. The quotient is ignored.

iii. Primary data is taken by the sender (minus *P* – 1 zeros) and *P* – 1 remainder added to it.

iv. Data including the added remainder is transmitted to the receiver.

v. On the other end, receiver utilizes the same divisor applies XOR division on the data including (*P* – 1) remainder.

vi. If the outcome of XOR division applied by the receiver is zero the data is deemed as independent of errors. In case the outcome is not zero than the receiver assumes that the data is altered.

Assuming:

Dividend D = 1 0 1 0 0 0 1 1 0 1

Divisor P = 1 1 0 1 0 1

Remainder R = to be computed

n = 15, d = 10, n − d = 5

Data after appended by zeros: 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0

The appended data is divided by P.

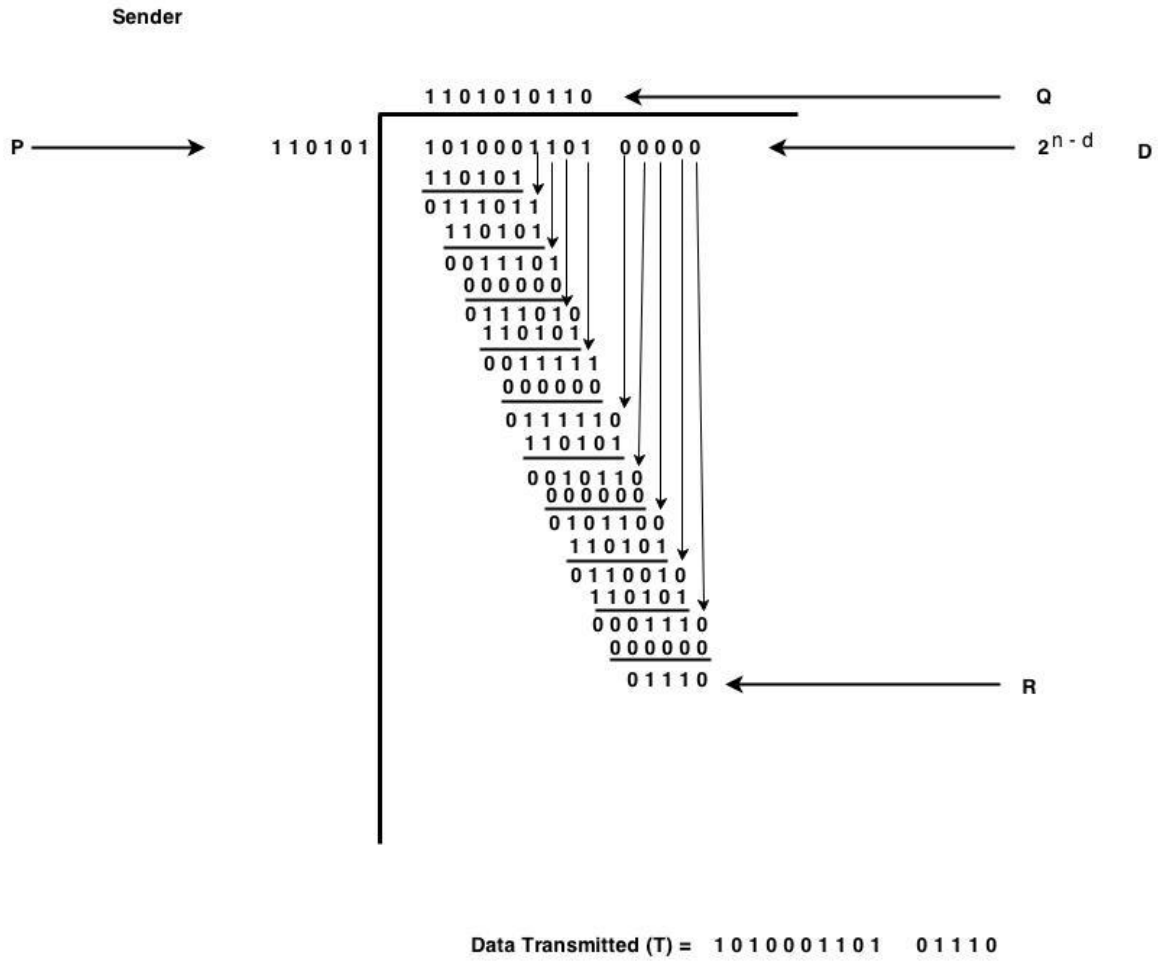


Fig: 2.5.1(a). Cyclic Redundancy Check (CRC) Division performed by the Sender.

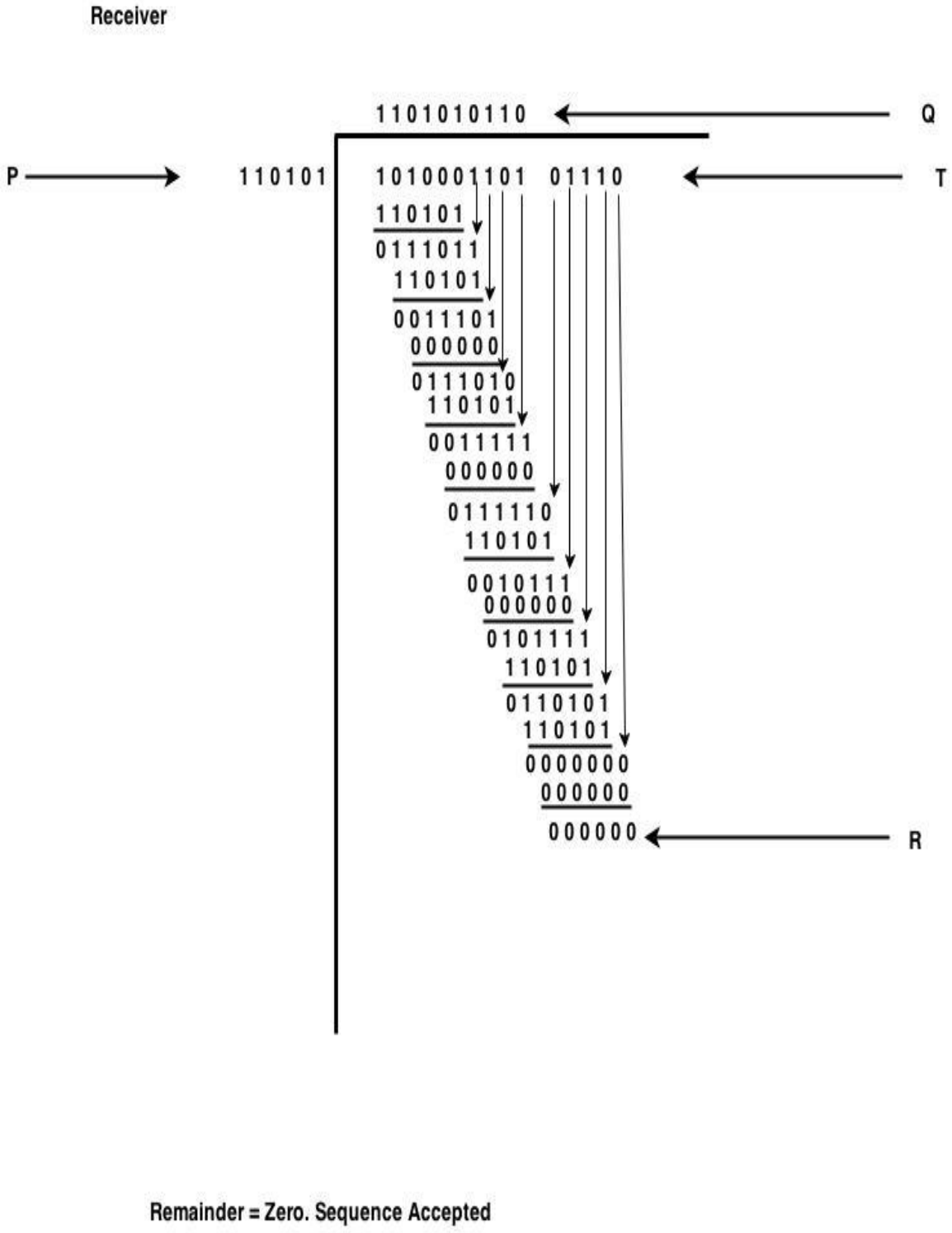


Fig: 2.5.1(b). Cyclic Redundancy Check (CRC) Division performed by the Receiver.

Sequence *P* is one bit longer than the expected frame check sequence. The precise bit sequence selected relies on the kind of anticipated errors. The high and low order bits of *P* should be 1. There is a compact approach in order to identify the existence of one or more errors. An error culminates in the inversion of a bit. This is similar to undertaking the XOR of the bit and 1 (i.e: 0+1=1, 1+1=0). Hence it can be concluded that errors in a block of *n*-bit can be depicted by a field of *n*-bit with 1's in every error location. The concluding block T_r can be defined as:

$$T_r = T + E$$

T = Transferred block.

E = Error sequence with 1's in locations where error transpires.

T_r = Received block.

+ = XOR.

If an error has occurred (i.e: E is not equal to 0), the receiver will be unsuccessful in identifying the errors, only if T_r is divisible by *P*, which is similar to *E* divisible by *P*.

2.5.2 Polynomials

Based on extensive research 16 bit optimal CRC's were publicized by Castagnoli, Ganz, and Grabber (1990). They recognized the significance of utilizing distinct CRC's relying upon the dataword size of concern. Assessment of CRC usefulness was founded on HD and Pud.

Beneficial 24 bit and 32 bit CRC's were publicized by Castagnoli *et al*., (1993) founded on the partly search of probabilities utilizing uniquely aimed search hardware. The search was restricted to fewer than entire probable polynomials because of limited computational strength. Freshly suggested polynomials were compared to prevailing standard polynomials founded upon HD and Pud. Uniquely aimed search hardware was also produced by Chun & Wolf (1994) for analyzing CRC's. The assessment standard was minimum Pud for BER values of roughly 0.01 to 0.5.

Funk (1996) publicized research of the leading CRC's and other truncated sequential codes. His outcome comprised an extensive search of entire codes with a check pattern length of 5 through 16 bits. In some scenarios a k-1 bit CRC and a parity bit, performed moderately better

than a k bit CRC. Perfomance assessment was accomplished founded on HD and Pud for BER values initiating at 10^{-4} and up.

Baicheva *et al.,* (1998) in his publicized paper provides recommendations for better 8 bit CRC's. Only few polynomials with specific factorizations were evaluated. The standard utilized for assessing polynomials were: size of dataword for which the code is 'unchanging', 'better' and 'correct' where these are particular mathematical properties. In addition, hamming distance and covering radius are provided for every code. Lastly number of bits in the polynomial (code weight) is deemed as an element comprising of costs of calculations. Eventually, CRC's were assessed for Pud at a comparatively high BER or 0.01 to 0.5 for an individual dataword size of 40 bits. These assessment standards were mathematically founded, like the description of 'correct' which is founded on how CRC conducts itself as BER moves towards 50% (Cheong, Barnes, & Friedman, 1979).

A different approach for understanding the CRC procedure is to articulate the entire value in a sample variable X, with binary coefficients. The coefficients equates to the bits in binary numbers (Stallings, 2008). Therefore, if $D = 1\ 0\ 0\ 0\ 0\ 1\ 1$ then $D(x) = X^6 + X + 1$, if $P = 1\ 1\ 0\ 0\ 1$ then $P(x) = X^4 + X^3 + 1$.

Forouzan (2007) describes polynomials as "a sequence of zeros and ones that can be illustrated as a polynomial with coefficients of 0 and 1. The power of every expression displays the bit location. The value of the bit is displayed by the coefficient.

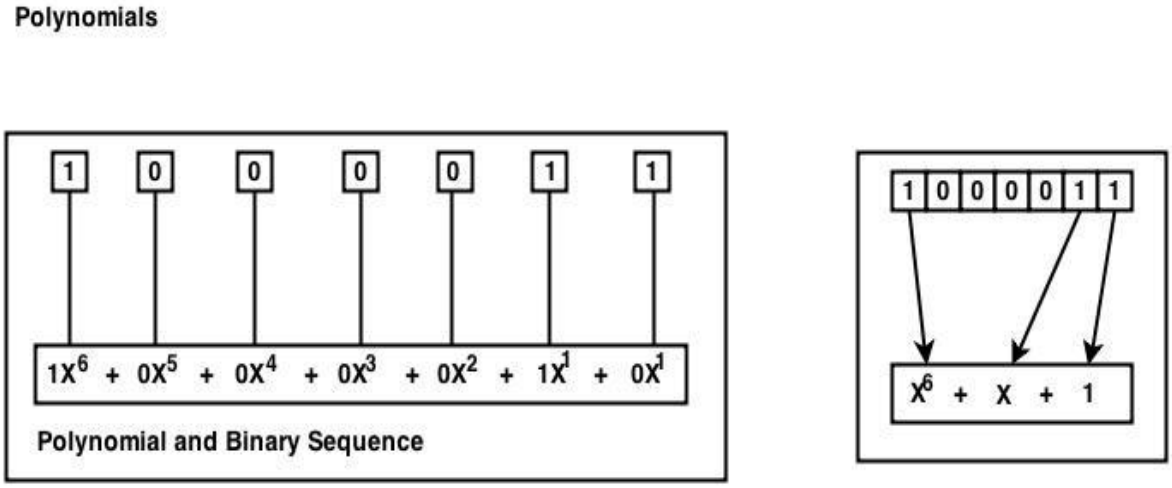


Fig: 2.5.2. Polynomials and Binary Data

Degree of a polynomial is considered to be the highest power in the polynomial. In the above example 6 is the highest power of polynomial. It is worth considering that degree of polynomial is one less than the number of bits in the sequence.

In the context of mathematics polynomial addition and subtraction are accomplished by the addition or subtraction of the coefficients of expressions with the same power. In the example mentioned above modulo-2 is utilized for addition and coefficients are 0 and 1. There are two outcomes; firstly adding and subtracting are identical. Secondly, addition and subtraction are accomplished by incorporating expressions and removing sets of similar expressions. For example the result of adding$(X^7 + X^6 + 1) + (X^6 + X^5) = X^7 + X^5 + 1$. X^6 is removed. In case three polynomials are added and outcome is X^4 three times, a set of X^4 is removed and the third one is retained.

Multiplication of two polynomials is accomplished, expression by expression. Every expression of first polynomial is multiplied by all expressions of second polynomial. The outcome is then clarified by removing sets of similar expressions. For instance the outcome of multiplying $(X + 1)$ x $(X^2 + X + 1) = X^3 + X^2 + X + X^2 + X + 1 = X^3 + 1$.

Polynomials division is fundamentally similar to division in binary. First expression of dividend is divided by first expression of divisor, in order to achieve first expression of quotient. The method is reiterated till the degree of dividend is less than the degree of divisor.
Assuming

$D = 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1, \quad D(X) = X^9 + X^7 + X^3 + X^2 + 1$

$P = 1\ 1\ 0\ 1\ 0\ 1, \quad P(X) = X^5 + X^4 + X^2 + 1$

$R = 0\ 1\ 1\ 1\ 0, \quad R(X) = X^3 + X^2 + X$

D after appending zeros $= 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0$

$D(X) = X^{14} + X^{12} + X^8 + X^7 + X^5$

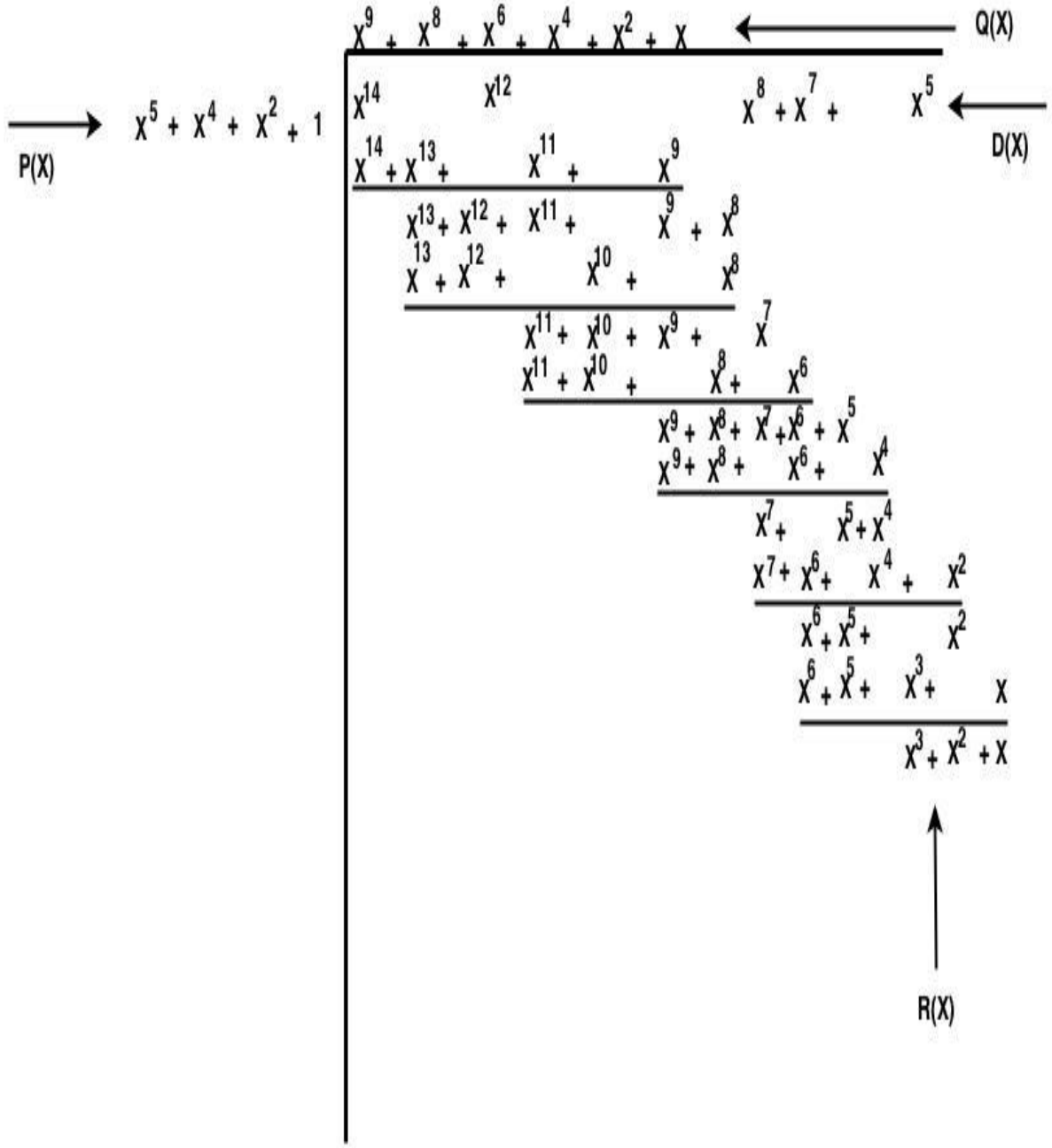


Fig: 2.5.3. Polynomial Division

Ramabadran and Gaitonde (1988) argues that the only way an error E(X) will be unidentified is if it is divisible by the divisor P(X).They demonstrate that all the errors listed below are not divisible by a carefully selected P(X)and therefore, identifiable.

i. If P(X) consists of more than one nonzero expression, all one bit errors are detected.

ii. All double bit errors are detected if P(X) is a polynomial of a particular kind known as primitive polynomial, with highest exponent L, and the length of the frame is less than $2^L - 1$.

iii. P(X) can detect any odd numbers of bit errors, if it comprises a factor (X + 1).

iv. P(X) can detect any burst error, if the length of the burst is lower than or equivalent to (n – k), in other words if it is lower than or equivalent to frame check sequence (FCS).

v. P(X) can detect a fragment of error burst of length n – k +1. Fragment $=1 - 2^{-(n-k-1)}$.

vi. P(X) can detect a fragment of error bursts of length larger than n – k+1. Fragment = $1 - 2^{-(n-k)}$.

According to Stallings (2008) Models of Generator Polynomials / Divisor broadly utilized are:

$$CRC - 12 = X^{12} + X^{11} + X^3 + X^2 + X + 1$$

$$CRC - 16 = X^{16} + X^{15} + X^2 + 1$$

$$CRC - CCITT = X^{16} + X^{12} + X^5 + 1$$

$$CRC - 32 = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

(Equation 2.5.2)

CRC-12 model produces a 12-bit FCS, and it is utilized for the transmission flow of 6-bit characters. CRC-16 and CRC-CCITT generates a 16-bit FCS and both are favored for 8-bit characters in U.S and Europe. Even though CRC-32 is defined as a choice in few point to point synchronous transmission standards and is utilized in (IEEE 802 LAN standards), it is deemed acceptable in majority of applications.

Chapter 3: Design and Analysis

3.1 Introduction

This chapter presents design parameters of the utilized fault model and then presents an analysis of the proposed parameters.

3.2 Design Parameters

i. Size of Frame Check Sequence is (8, 16, 24, 32 bits) or more.

ii. Maximum length of dataword to be safeguarded and dataword size probability dispensation.

iii. Type of Errors: (random data, significant zero's, bit adding/removing, bit lapse, bi-symmetrical burst, erasure burst, bi-symmetric BER, BER random erasure).

iv. Type of Checksums: (XOR, 1's complement, 2's complement, Adler, Fletcher, CRC).

v. Communicating design boundaries: (bit lapse, stuff bits, safeguarding length field, value disparity).

vi. Other: (to evade all zero codewords, to reduce disguised attacks).

3.3 Prospective Analysis Considerations

i. Hamming distance (HD)
ii. Hamming Weights (HW)
iii. P_{ud} for stipulated BER and length of dataword.
iv. Burst error identification (longest burst for which entire errors are detected).
v. Burst error performance (HD for specified burst length).
vi. Smallest dataword length for unidentified k- bit error (k=2, k=5).
vii. Performance of multi HD (Big HD at small length and small HD at big length).
viii. Impact of data values on checksum usefulness.
ix. Cost of calculations (speed, memory, recalculating after alteration in small dataword).
x. Divisibility of polynomial by (x+1).

xi. Utilization of standard CRC.

xii. Utilization of a polynomial with a particular divisibility.

xiii. Evaluation of error detection with entirely random alteration.

xiv. Identification of arrangements in data sequence as means of alteration.

3.4 Analysis

One way to analyze error detection performance is to contemplate random data alterations. But for this fault design, the option of error detection code is mainly insignificant. If crucially random data is being examined by an error detection code, then an unintentional match can be anticipated between dataword and FCS with probability $1/(2^k)$ irrespective of error coding method. As the flaws in the codeword become small, there are very prominent distinctions between coding methods. If more bits are altered the utilized error coding method becomes less significant. But if there is a scenario where only a small number of bits are erroneous then the kind of error code used is significant. Hence the selection should concentrate on performance for only a few bits.

For a small number of bit errors, the hamming distance of a checksum becomes significant. Each checksum has particular sequences of comparatively small number of bit errors that it is unable to identify. Hence if the fault design is a particular sequence of bit errors, performance of checksum relies on that particular sequence which cannot be generalized. But in many scenarios, bit errors are not sequenced, instead they are produced by noise or random alterations, or other methods that can for realistic reasons be thought of as random independent occurrences. This random nondependent occurrence is the fault design adopted in this research. If a specific system has sequences of bit errors, for instance because of a hardware flaw in a transitional phase in a network, then the established outcomes in this research may not be exerted.

For random independent bit errors the question that needs to be asked is how frequently they happen. This research focuses on error detection and not error correction. Therefore, it should be presumed that any data with errors must be dropped and focus will be on enhancing the probability that errors are identified. But for any such system to function, the rate of error must be comparatively low, or virtually all data will be dropped and the system will not serve

any purpose. Hence it is acceptable to consider that the expected entry of bit errors is generally less, than once per codeword (i.e. most code words are without errors). This idea can be expressed in a typical way by assuming bit errors enter through exponential inter arrival time with a mean of BER value that is notably less than the inverse of dataword length. For instance if length of a codeword is 1000 bits, BER should be notably less than 1/1000 for assuring that most codewords are independent of errors. With this method most code words are independent of errors, some carry one bit error, a smaller number carry two bit errors, and an even smaller number carry three bit errors and so on. This can be considered a 'BER fault design'.

As an estimation, the possibility of (r+1) bit errors transpiring in a codeword is BER times the possibility of r bit errors transpiring. Hence each bit of HD increment in an error detection scheme culminates in roughly a factor of 1/BER decrease in the possibility of unidentified error.

Given a BER fault design the possibility of an unidentified error relies on the value of BER, codeword length, the HD of error code utilized, the possibility of unidentified error at the HD, and in uncommon scenarios the possibility of unidentified errors at HD+1. The possibility of unidentified errors at HD merely means the possibility of unidentified errors provided that HD bit errors have transpired.

This can be explained (for a codeword with c bits) as:

$$\mathrm{Pud} = \mathrm{Pud}_{(\mathrm{HD})} + \mathrm{Pud}_{(\mathrm{HD}+1)}$$

$$\mathrm{Pud} = \mathrm{combin}(\mathrm{c}, \mathrm{HD}) * \mathrm{Unidentified\ Fragment}_{\mathrm{HD}} * \mathrm{BER}^{\mathrm{HD}} * (1 - \mathrm{BER})^{(\mathrm{c}-\mathrm{HD})} + \mathrm{combin}(\mathrm{c}, \mathrm{HD}+1) * \mathrm{Unidentified\ Fragment}_{\mathrm{HD}+1} * \mathrm{BER}^{(\mathrm{HD}+1)} * (1 - \mathrm{BER})^{(\mathrm{c}-\mathrm{HD}-1)}$$

(*Equation 3.4*)

P*ud* means "possibility of unidentified error" provided a specific codeword length and BER value. In understanding the first part of the given (*equation 4.4*) "combin ()" is the combination of c bits of the codeword length taken HD at a time, which is the total number of probable errors that can transpire including precisely HD error bits. The unidentified fragment is the number of unidentified errors out of all probable errors that include HD bits, which differs relying on the

error identification code and how many error bits have been injected. For a CRC this is hamming weight divided by total number of probable errors. The remaining names for HD include the possibility that precisely HD bits are in error and precisely (c-HD) bits are not in error. The second part of equation reruns in an identical manner for HD+1 error bits.

P*ud* outcome is not influenced by less than HD errors, since 100% of such errors are identified based on the description of HD. Expressions including HD+2 can be ignored because of the presumption that BER is small in comparison to codeword length. HD+1 involves itself only when a very small fragment of bit errors including HD bits are unidentified and the Unidentified Fragment $_{(HD+1)}$ is on the sequence of 1/BER or larger. For many scenarios HD+1 is not important. The scenarios that are important incline to be only at codeword size close to the change of attained HD, when there are only a few unidentified errors comprising the up-to-date, less, HD number of bits in error. Founded on this methodology of P*ud*, the important elements in the effectiveness of error identification are the unidentified error fragment and the HD, with every bit of HD providing an enhancement in error identification usefulness of an added element of roughly BER. For instance, with an unidentified error fragment of 1.5e-6, a BER of 1e-8 and HD=4, every bit of HD reckons for nearly two orders of enormity more error identification usefulness than the unidentified error fragment. Hence, while it is significant to evaluate unidentified error fragment, for HD higher than 2 (relying on the merits of bit blending in error identification code), HD is frequently the important element in attained error identification usefulness.

Computational performance is another typical election yardstick, with checksums generally assumed to be much faster to compute than CRC's. Nevertheless, research into performance of CRC's has not only created swift CRC computation methods, but has also optimized CRC polynomials for calculation swiftness with very little cutback in error identification usefulness. Further explanation is given by Ray and Koopman (2006). This research focuses only on error detection effectiveness. If a checksum presents adequate error detection efficiency, than that may assist computation swiftness. However, computation swiftness should not be a justification for adopting insufficient error identification competence.

3.5 Additional Design Concerns

3.5.1 Impact of Primary Implant Values

Checksums and CRC's include a recurring calculation over the dataword length that acquires an outcome to design an FCS value. The acquired value must be initialized to some familiar value for this procedure to generate a favorable outcome for any provided codeword. When the primary value is not declared it is generally considered to be zero.

In some scenarios a non-zero value is utilized such as 0xFFFF for a CRC or a 16 bit checksum. This has the benefit of making an all zero codeword improbable. For Fletcher checksum and CRC this has the extra advantage of provoking main zeros of the message to impact the value of FCS. One way of comprehending this is to think of a CRC initialized with an all zero implant value. Any number of all zero dataword blocks can be induced into CRC and it will remain at a zero value, disposing of information regarding how many zero blocks have been operated on. But if the CRC calculation is initialized to any non-zero value, operating blocks with a zero value will cycle sequences through CRC, tracing the zero valued block numbers that have been operated as a kind of token. This can assist with scenarios in which extra significant zero bytes have been attached to a dataword erroneously.

Typically, any non-zero implant value can be utilized and the value chosen doesn't impact error identification properties besides those explained. In specific, CRC error identification in the context of BER fault design is free of implant value utilized.

CRC checksum values can be additionally or alternately adjusted to produce an FCS value. A frequent adjustment is to XOR the outcome of a checksum calculation with a value like 0xFFFF, which reverses bits but still retains the information components of the checksum calculation. In some scenarios, CRC outcomes must be bit inverted to safeguard burst error properties relying upon the calculation approach and bit sequencing of message in the dataword.

3.5.2 Identification of Burst Errors

Burst errors up to the size of data block are identified by all checksums and CRC's (for Fletcher checksum data block size is less than the size of FCS). Nevertheless this property relies on the proper bit sequencing of calculations in two ways. Firstly it is considered that bit

sequencing in the calculation is similar to the bit sequencing of transmission in a network or physical bit sequencing in memory. Secondly, it is considered that bit sequencing of the FCS value is such that the safeguarded dataword's last bit mainly influences the last bit of the FCS. In simpler terms, the last dataword bit and the lowest sequence bit of FCS are separated with a large distance as possible.

Within bytes, an inaccuracy in bit sequencing, data blocks, or FCS can decrease burst error coverage. Ray and Koopman (2006) indicate that IEEE 1934 firewire has 25 bit burst error ability rather than the assumed 32 bit error identification ability anticipated from execution of its (CRC 32) bit error identification code. This is because of the draw back with bit sequencing of transferred bytes being the inverse of bit sequencing within bytes utilized for CRC calculations.

3.5.3 Detection of Modifications in Data Sequence

It may be worthwhile in some applications to detect modifications in the sequence of data. An example perhaps would be when multiple network packets are rebuilt into a long dataword. Additional checksums and longitudinal redundancy check (LRC) are inconsiderate to data sequence owing to the fact that they merely add data blocks. Hence they will never identify an error that is entirely in the shape of mis-sequenced data blocks.

The Fletcher checksum will identify most data sequencing errors provided that the out of sequence data blocks are late enough to be within the range of HD=3 dataword lengths. The reason behind this can be comprehended by considering that SUM B expression of Fletcher checksum in essence multiplies every data block by a steady integer specifying how distant that data block is from the FCS. For a 16 bit Fletcher checksum modulo 256 is utilized by this multiplication, hence prohibiting identification of all re-sequencing challenges that are multiples of 256 data blocks apart.

Typically all data re-sequencing errors are detected by CRC. Any susceptibilities to data sequencing errors would be fixed to the specific data being transferred and the specific polynomial being utilized and anticipated to be artificially-random in character.

In the above analysis, there is no surety that all probable data sequencing errors will be identified. Instead, the deduction to be extracted is that regular average checksums present no

ability, Fletcher checksums give some ability, and CRC's plausibly presents ideal ability to identify data sequencing errors. Enumerating this preservation further is outside the sphere of this research.

Chapter 4: Research Methodology

4.1 Introduction

The fundamental aim of this research is to comprehend how frequently undetected errors can happen and how to relate that information to probity levels.

An unidentified error is thought to have transpired, when an altered codeword is authentic. A codeword is the sequence of a specific dataword and the matching computational outcome of the error coding method, which is frame check sequence (FCS) or frame check. By reversing one or more bits a codeword can be altered. These bit alterations can influence dataword, FCS or both. The altered codeword can then be examined to find out if it is an authentic codeword. In simpler terms, after alteration, there is an error check to find out if the error detection calculation, when applied on the altered dataword, produces a value that is similar to the altered FCS. If the altered FCS value from the codeword accurately emulates the error coding computations applied on the altered dataword, the error is unidentifiable utilizing that error coding scheme. In this research, for an error to be undetectable in codes, the dataword must be altered; however the FCS may or may not be altered in order to create a codeword with an unidentifiable error.

4.2 Research Design

The kind of research method to be utilized relies greatly on specific research objectives and has to correlate with research questions. The selection of a research design is largely compelled by the questions that the research is attempting to answer (Kothari, 2004).

Applicable design for establishing error detection abilities of a specific coding scheme comprises of dataword size, FCS size, number of error bits and on occasions, value of dataword prior to alterations.

The most suitable research design in the context of this dissertation is fault design with manufactured error simulations. A fault design is a binary symmetric medium with bit inversion faults (can be called as bit flip fault design). In other terms, a data value is depicted in bits and every bit error culminates in a single bit value being reversed from 0 to 1 and 1 to 0. The fault

design utilized for this research is independent random bit errors with a number of other considerations.

A data word was created for every test comprising a particular data value and a checksum was created for that data word. The concluding code word (data word + check bits) was exposed to a particular number of inverted bit flaws, giving the appearance of inverted bit errors in the course of transfer. The checksum of the flawed data word was then calculated and comparisons were made with possibly flawed value of frame check sequence of flawed code word. In case the frame check sequence value of the flawed code word was equal to the frame check sequence value calculated for the flawed data word than it was concluded that the utilized check sum was unable to identify a specific set of inverted bits. Excluding CRC, which is recognized as data independent (performance of error detection is not influenced by the values of data word), similar values for data words were utilized for all checksums.

The mathematical technique used for the proposed fault design is Monte Carlo simulation utilizing Mersenne Twister random number generator as the source of mock random numbers (Matsumoto & Nishimura, 1998).

4.3 Simulation Environment

The simulation environment used in this research is the computer simulation tool Matlab version 2016a. Matlab is a highly effective technical computing language and interactive environment for developing algorithms, visualization of data, analysis of data, and numeric calculations. By utilizing Matlab many technical computing complications can be resolved swiftly compared to conventional programming languages like C, C++, and Fortran. Matlab can be utilized in a broad sphere of applications incorporating signal and image processing, financial structuring and analysis, test and measurement and control design. Add-on tool boxes (a selection of specific purpose matlab functions accessible independently) stretches the Matlab environment to resolve specific classes of problems in these application spheres. Matlab offers a range of features for the purpose of documentation and distribution of work. Matlab code can also be incorporated with other applications and languages and share Matlab algorithms and applications. The following steps were taken in Matlab for error detection simulations:

i. Induce a random dataword value with precisely a pre-established number of randomly located bits set to 1 and rest set to 0.

ii. Calculate the checksum of this randomly created dataword, attach that FCS value to create a codeword.

iii. Induce a random error vector with precisely a pre-established number of randomly positioned 1 error bits through the whole codeword, with 0 bits everywhere else in the error vector.

iv. Calculate the altered codeword by Xoring error vector into the codeword. The outcome is outturned bits where error vector has 1 bits.

v. Establish if the culminated altered codeword is authentic by calculating the checksum of the altered dataword and comparing that outcome to find if it is equal to the altered FCS from the codeword.

vi. Document an unidentified error if the altered codeword is authentic, implying that altered dataword relates to the altered FCS. The result is an authentic altered codeword and hence an unidentified error.

4.4 Hardware/Software Environment for Error Detection Simulations

A computer utilized for simulations must consist of following peripherals.

i. **Processor, RAM, and Motherboard:** to accomplish any real time simulations.
ii. **Ethernet Board:** supported by Matlab to download the compiled code and acquire saved data.
iii. **VGA Card:** for an illustrated evaluation regarding real time measurement.

Matlab 2016a 64-bit from mathwork was utilized as the primary simulation platform for error detection. To run simulations and to make a comparison of simulated outcomes the Simulink package was installed along with Matlab. Matlab necessary toolboxes are only supported by windows operating system. The simulations were conducted on Dell Inspiron 3521

Laptop with 64-bit windows 8 operating system, installed memory(RAM) of 4.00 GB and 1.90 GHz intel core processor.

4.5 Random Data Sampling

Random data sampling method utilized in this research was Bernoulli distribution utilizing Bernoulli generator block from Matlab/Simulink. The nature of binomial distribution makes it a sampling distribution. In simpler terms the value of binomial random variable is adopted from a sample of size *n* from a population. Population can be understood as a set comprising of zero's and one's. The sample is acquired as a pattern of *n* experiments with every experiment being a draw from the population of zero's and one's (such experiments are known as Bernoulli trials).

Bernoulli binary generator utilizes Bernoulli distribution for the purpose of generating random binary numbers. The distribution has a parameter p (probability) which gives zero and one when the probability is (1-p). For this kind of distribution the mean value is presented as (1-p) and the variance is shown as p (1-p). When p is indicated by a possibility of zero parameter, any real number between zero and one can be produced. The data rate is set in this block in the sampling time parameter presented as: Sampling time=R_b (data rate)
It encodes the binary input signal and its output is the rational distinction between the current input and the preceding output. The initial state in this block is set to zero.

4.6 Data Testing Methods

Tests for analyzing error detection performance of checksums were accomplished through Monte Carlo simulation in Matlab 2016a. A minimum of 10 tests were conducted for each kind of test and the average of all the tests were attained. For instance for a (1024 + 8) bit code word with random data and 8 bit checksum size 10 tests were executed in order to insert probable 2 bit errors.

The tests for random non-dependent bit errors were carried out by choosing in advance a fix number of bit errors in order to initiate and insert into the code word. For every data word value investigated, the inserted flaws were 1, 2, and 3 bit errors in the code word. For every specific test the entire number of unidentified errors was recorded.

Burst errors are considered to be random bit errors that are limited to a length of *d* bits for a burst error of *d* bit inside a specific message (Koopman and Maxino, 2009). Inside a burst error any number of bits can possibly be altered. Tests for burst errors were carried out in the same way as tests for bit errors. Only exception is that rather than putting through the concluded code word to bit inverts, code word was exposed to particular lengths of burst error with entire probable burst error sequences inside the burst spheres and entire probable locations of burst position.

The smallest number of bit errors for which there is a minimum of one unidentified occurrence is the hamming distance of a checksum. For instance a cyclic redundancy check with a hamming distance of 4 will be able to identify 1, 2, 3 bit errors however, out of all possible 4 bit errors it will be unsuccessful in identifying a minimum of one 4 bit error. In a binary consistent medium presumably non dependent random errors, the primary helping element to usefulness of checksum for various embedded applications is the proportion of unidentified errors at hamming distance, the reason being that the possibility of added errors transpiring is considerably less anticipated (presuming bit error 10^{-6} BER, the frequency of 3 bit errors transpiring is a million times more than 4 bit errors). Hence the analysis of unidentified errors at the HD was accomplished to provide understanding of test outcomes.

In burst error tests, for the inspection of every data value, infectious burst errors beginning from 2 bits up to a minimum (k + 1) bits were inserted. For a specific burst error degree all practical infectious values of burst errors were inserted. For instance in a code word with random data and checksum size of 8 (1024 + 8), all practical burst errors of 9 bits were inserted. Tests were halted the moment there was a single burst error unidentified for that particular burst error degree. The reason for this is that the general standard of intrigue for burst error is the peak length at which burst errors are assured to be identified. The deviation to this was in the tests performed for Adler and Fletcher checksums. For the purpose of uniformity the tests conducted were up to (k + 1) infectious bursts, though it was anticipated that there will be unidentified burst errors at k/2 infectious bits (*k* checksum size).

For the purpose of making comparability easier, this research utilized data word lengths that are multiples of checksum size; this is also typical of real life actual networks. For instance for a checksum of 2's complement addition, a data word of 24-bit utilized with a checksum size

8-bit concluded in a calculation that separates the data word into 3 chunks of 8 bit each. Addition was performed on these 3 chunks in order to calculate the 8 bit checksum, producing a code word (24 + 8) bit.

4.7 Data Analysis

Analyzing the usefulness of checksums normally needs an amalgamation of methods that are experimental and analytic in nature. XOR which is a limited field function can generally be assessed analytically. But analytical method to comprehend error identification becomes very complicated in integer addition functions because of the ingrained inter bit carries. The approach followed in this research is utilizing analysis in order to estimate the outcomes to a degree that is realistic, with imitations utilized to verify the analytic outcomes.

Findings of the research will be presented in terms of graphs. Authenticity of the checksum calculations were scrutinized by comparing it to an earlier research (Koopman and Maxino, 2009), which in turn was scrutinized via comparison to acknowledged publicized outcomes.

It should be observed that the test process was different from merely producing unrestricted random dataword values in the sense that precisely the number of desired bits were set in the dataword for each test (precisely 50% 1 bits), rather than that number of bit being set on average with some dispersal of real number of one bits. So for a 128 bit dataword, if 50% of bits were assumed to be 1 bits than precisely 64 randomly placed bits were ‘1’ for every test. Furthermore it should be observed that error bits influence the whole codeword. This includes dataword, FCS bits or both relying on the error bit location in a specific test.

A number of tests were executed for every data point, where a data point being a specific number of error bits and a specific dataword length of concern. Every data point had adequate test runs to encounter a minimum of 10 unidentified errors for each data point that was utilized for graph outcomes. After ascertaining that a 50% blend of one’s and zero’s resulted in worst case unidentified errors for checksums as described by (Koopman and Maxino, 2009) datawords with 50% randomly placed one bits were utilized for all tests.

CRC error detection performance was tested by utilizing logically precise analysis of all probable k bit codeword error patterns. Hamming weights of CRC doesn't depend on the values of dataword, implying that values of dataword were not significant in this analysis. Methods explained by Koopman (2002) and Chakravarty (2004) were utilized for this analysis. Authenticity of checksum calculations were analyzed by comparing it to a prior research (Koopman and Maxino, 2009).

There is no logically precise analysis approach currently present. The rationale is that performance of checksums relies not only on error detection calculations and error bits, but also on data values in the communication.

Chapter 5: Test Results

5.1 Introduction

Error detection efficiency of checksums relies upon how effectively the checksum calculation 'blends' the bits of the data to construct an FCS value and if that 'blending' is accomplished in a manner that a specific HD is assured for every probable error sequence beneath that HD value. This chapter presents test results of error detection performance for a number of checksums and CRC. The main aim of this phase is to test the error detection effectiveness of checksums like XOR, one's complement, two's complement, Fletcher, Adler and CRC.

5.2 Error Identification Random HD=1

Brief:

Random "Blending" function HD=1

Hamming Distance: HD=1

Unidentified Error Fragment at HD: for k-bit FCS $1/2^k$

Coverage for Burst Error: None

Data Dependency: Hypothetically, but quality of blending is not dependent on data.

Identify Modification in Data Sequence: Yes, depending on HD and unidentified error fraction.

This is a conceptual method to explain error identification effectiveness of checksums and CRC. The presumption is that some checksum like function has been described which gives exceptional blending of bits in calculating a checksum, but fails in identifying some 1 bit errors. In this scenario, HD=1, however because of presumption of 'ideal random' blending, such that a modification in any one bit of the dataword modifies every bit of the checksum with an equivalent and entirely random possibility. In a system like this it is probable that a one bit error in dataword will have no impact on the value of FCS (if HD higher than 1 was assured, that property will construct a foreseeable sequence in the output of the blending function moving hypothetically towards a security weakness). The unidentified fragment of 1 bit errors for this function is $1/2^k$ for a k bit FCS. In common there would also be no assurance of burst error identification because a 1 bit error in an improper scenario may be unidentified. It must be

stressed that this is a hypothetical restriction rather than a discovery that pertains to any specific hash function. Table 5.1 describes the hypothetical effectiveness of a random hash function.

Table 5.1: Random Error Identification.

Length of dataword=1024 Size of checksum	**Logical unidentified fragment at HD=1**
8 bit	0.039
16 bit	0.000015 or $1.51x10^{-5}$
32 bit	$2.3 \text{ X } 10^{-10}$

In actuality, cryptographically dependable hash functions are likely to give this kind of error identification. They were not devised to give a specific HD but rather give cryptographically dependent blending of bit values.

5.3 XOR Checksum

Brief:

Block wise XOR of all data chunks.

Hamming Distance: HD=2

Unidentified Error Fragment at HD: 3.125% for 32 bit block size for all dataword lengths.

Coverage for Burst Error: Block size.

Data Dependency: No

Identify Modification in Data Sequence: No

A Longitudinal redundancy check (LRC) is often known as 'XOR checksum'. It includes Xoring all the blocks of a dataword with each other to produce a check pattern. It is similar to calculating the parity of every bit location independently within a block (check pattern location *i* is the parity of bit location *i* throughout all blocks in the dataword).

XOR checksum error code provide HD=2, with identification of burst error up to block length. Any two bit errors in the identical location of a chunk (also including FCS) are unidentified and performance of error identification is inconsiderate to data values. If there is an odd number of bit errors in any bit location, XOR will identify that there is an error, enhancing

average scenario performance in comparison to single parity bit. XOR can be considered as calculating an isolated parity bit for every bit location of the chunk, and preserving those parity bits as FCS. An error is identified unless each one of those parity bits is unsuccessful in identifying all errors in that bit location.

The possibility of an unidentified error relies upon the block size of calculation. Theoretically it is the possibility that 2 bit errors result in lining up in the identical bit location of the block (to be unidentified an 8 bit block with one error in bit 3 of a block, the second error has to be in bit 3 of a different block or FCS).

According to Koopman and Maxino (2009) the possibility of an unidentified 2 bit error is $n-k\ /k(n-1)$. k is the number of FCS bits and n is the number of codeword bits. Table 5.2 provides unidentified error possibilities from Monte Carlo simulation having a dataword of 1024 bits with logical values from Koopman and Maxino (2009).

Table 5.2: XOR Error Detection.

Length of dataword=1024 Size of checksum	**Simulated unidentified fragment at HD=2**	**Logical unidentified fragment at HD=2**	**Deviation from logical fragment**
8 bit	0.12418	0.12415	0.02% worse
16 bit	0.06160	0.06159	0.01% worse
32 bit	0.03036	0.03031	0.05%

Figure 5.1 illustrates the outcome of simulation for a 32 bit XOR checksum and random HD=1. Graphs for 8 and 16 bit checksums are identical. Though the unidentified error fragment is steady, the possibility of multi bit error increases with increased length of dataword.

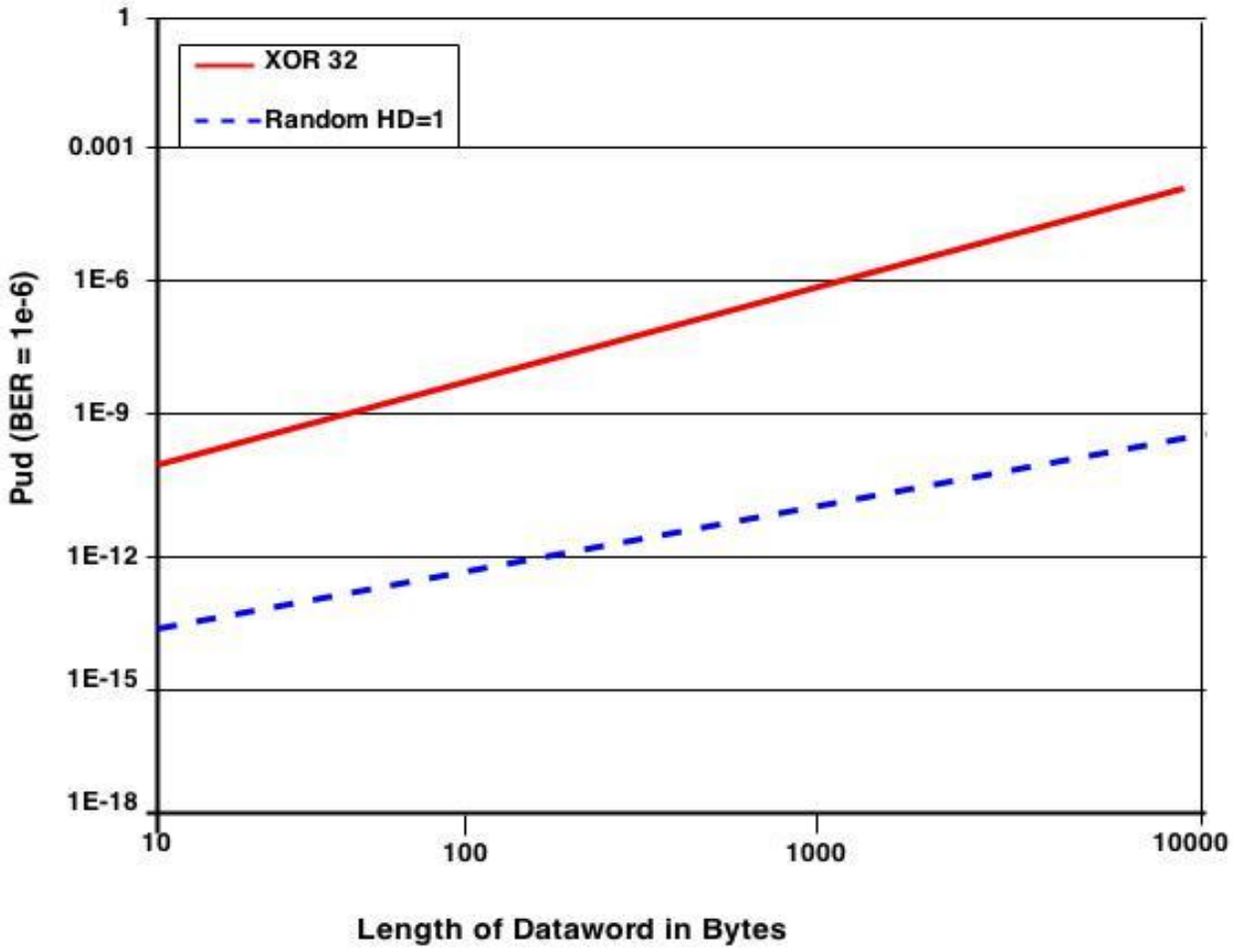


Fig 5.1: Evaluation of 32 bit XOR checksum.

An intriguing feature of figure 5.3 is that an HD=1 random hash value shows more promise than checksum of HD=2. The reason is that unidentified error fraction of the displayed checksum is fairly inflated because of comparatively deficient blending of the bit by the checksum function. In simpler terms, a good hash function with HD=1 can be anticipated to give better performance than HD=2 checksums. Nevertheless, far better performance is given by other checksums and CRC's than checksums exhibited in above figure.

When planning error identification evaluation, it is significant to be conscious of presumed BER. The planned values will rely to a great degree on that value. Further, it is pointless in a number of scenarios to expand the graph to a stage where each message has a likelihood of having an error and the system utilizing error identification codes will plainly discard practically all its data as faulty.

5.4 2's Complement Checksum

Brief:

Common integer addition of entire data chunks.

Hamming Distance: HD=2.

Unidentified error Fragment at HD: For 32 bit FCS = 1.562%

Coverage for Burst Error: Block size.

Data Dependency: Yes

Identify Modification in Data Sequence: No

By utilizing integer addition, checksums of entire blocks of dataword are calculated. The resulting sum creates the check pattern.

A two's complement checksum provides HD=2, with identification of burst error up to the block size. It functions by utilizing 'common' two's complement addition specifications. The number of 2 bit errors is minimized in comparison to XOR checksum. The reason is that a set of reversed bits in the identical bit location can often be identified by the certainty that the carry produced by adding those bits is distinct. But, dependence on carry function to enhance efficiency of checksums implies that the performance is dependent on data with high performance on data values with either all zero or all one and low performance for data values with a random 50% blend of one and zero.

As is common for two's complement arithmetic, carry out from top bit of the addition are disregarded, resulting in the highest bit of every chunk being more susceptible to unidentified errors (entire bits in the chunk are safeguarded by the carries from an addition function, however the highest bit in the chunk is merely secured by an XOR function because of discarding the carry out from addition of highest bit).

The logical unidentified error fragment is complicated, however an easy to understand estimate that is nearly precise at long lengths of data and suspicious at small length of data is provided by $k + 1/2k^2$ (Koopman and Maxino, 2009).

Table 5.3: 2's Complement Error Detection.

Length of dataword=1024 Checksum size	**Simulated unidentified fragment at HD=2**	**Publicized unidentified fragment estimation at HD=2 (Koopman et al., 2009)**	**Variation**
8 bit	0.06943	0.07031	0.86%
16 bit	0.03264	0.03320	1.62%
32 bit	0.01523	0.01611	3.13%

Table 5.3 exhibits the performance of 2's complement error detection. Figure 5.2 in the following section will illustrate the performance of 2's and 1's complement checksum. 2's complement checksum is notably superior than XOR checksum because of the inside carry bits minimizing the possibility of unidentified 2 bit errors. However, on a log-log graph this 'notable' superiority (roughly a factor of 2) is comparatively smaller.

5.5 1's Complement Checksum

Brief:

1's complement integer addition of entire data chunks.

Hamming Distance: HD=2

Unidentified Error Fragment at HD: For 32 bit FCS 1.512%

Coverage for Burst Error: Block size

Data Dependency: Yes

Identify Modification in Data Sequence: No

When it comes to technicalities of calculations, a 1's complement addition is similar to 2's complement with the exception that the carry out of every addition is incorporated back into the lowest bit location of that addition outcome. In simpler terms if a 2's complement addition culminates in a carry out of zero, than that is utilized as the 1's complement addition outcome. But, if a 2's complement addition culminates in a carry out of 1 than the outcome of addition is augmented by 1 to provide a 1's complement addition outcome. This is an execution approach which tackles the reality that an all zero sequence and an all one sequence both depict number zero in 1's complement arithmetic.

From the perspective of error detection, the appeal to this method of calculation is that carry out bits are not discarded in the process of calculations, hence moderately enhancing blending of bits and error identification efficiency. Assessment signifying the dominance of 1's complement addition to 2's complement addition is presented by Usas (1978).

The logical unidentified error fragment is complicated, however simplified estimation for large data sizes is provided by Koopman and Maxino (2009) as 1/2*k*.

Table 5.4: 1's Complement Error Detection.

Length of dataword=1024 Checksum size	**Simulated unidentified fragment at HD=2**	**Publicized unidentified error estimation at HD=2 (Koopman and Maxino, 2009)**	**Variation**
8 bit	0.06211	0.06250	0.62%
16 bit	0.03064	0.03125	1.99%
32 bit	0.01512	0.01563	3.37%

Table 5.4 depicts the performance of 1's complement checksum for error identification. Figure 5.2 illustrates that though 1's complement checksum gives moderately superior error identification than 2's complement, the distinction is hard to see on a log-log graph. If viable, a 1's complement checksum should be utilized instead of a 2's complement checksum. Nevertheless, a more sophisticated checksum like Fletcher checksum is a superior option.

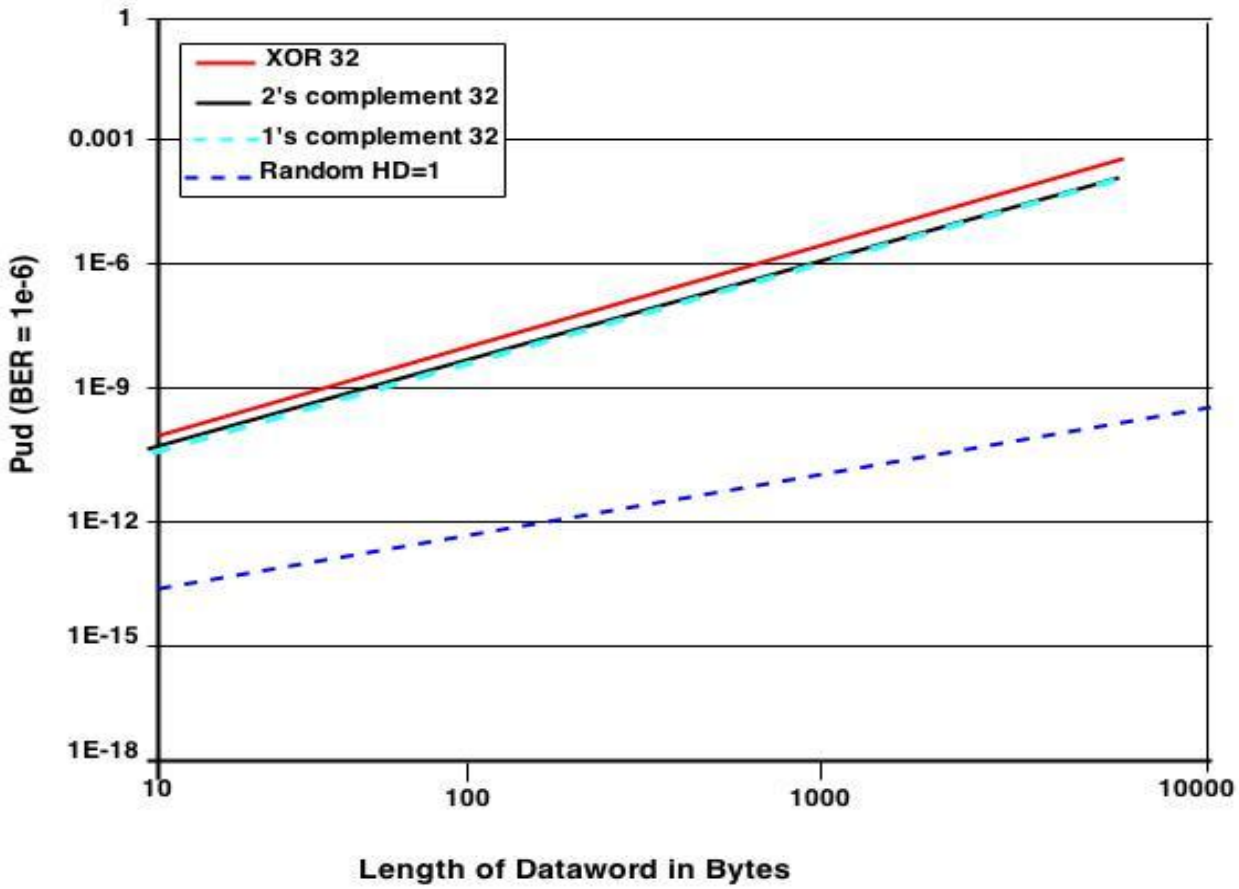


Fig 5.2: Evaluation of 32 bit XOR, 1's Complement and 2's Complement Checksum.

5.6 Fletcher Checksum

Brief:

1's complement addition of two consecutive sums.

Hamming Distance: HD=2 for long lengths, HD=3 for small lengths.

Coverage of Burst Error: At minimum 50% of FCS length (i.e. length of one consecutive sum)

Data Dependency: Yes

Identify Modification in Data Pattern: Yes, depending on HD and unidentified error fragment.

Fletcher checksum (Fletcher, 1982) is calculated with a set of consecutive arithmetic sums. The first consecutive sum is merely the 1's complement sum of all blocks in the dataword. The first consecutive sum is added into second consecutive sum after each data block is operated on.

$$Sum\ A = Sum\ A + next\ data\ block$$

$$Sum\ B = Sum\ B + Sum\ A$$

Sum A and Sum B are set to some steady value. Additions are 1's complement addition and the procedure is repeated over data blocks for the complete dataword, with the dataword length being half of the FCS length. At the conclusion of checksum calculation, FCS is the sequence of Sum A and Sum B. Hence a Fletcher checksum of 32 bits is the sequence of 16 bit Sum A value and 16 bit Sum B value. This repeated method produces the final second checksum, with 1 times the data block, 2 times the next to final data block, 3 times the next most current data block and so on. This makes it feasible to identify modifications in data patterns for HD=3 error identification lengths. For a 1's complement checksum, Fletcher checksum utilizes 1's complement addition as explained earlier.

Nakassis (1988) gives execution suggestions for Fletcher checksum and gives standards for error identification founded on possibility of unidentified random alteration, burst errors of 16 bit, single bit errors and minimum distance between unidentified double bit errors. The improper utilization of 2's complement addition in place of 1's complement addition for calculating a Fletcher checksum has appeared to decrease the minimum distance between unidentified double bit errors from 2040 bits distance to only 16 bits distance for a 16 bit Fletcher checksum. Nakassis also indicates a standard of simplicity for recalculating a checksum after a modest modification to the dataword and suggests that Fletcher checksum is superior in accordance to that standard.

Koopman and Maxino (2009) asserted that an 8-bit Fletcher checksum has HD=3 up to 60 dataword bits and HD=2 for more than 60 bits. This is established by this research. Koopman and Maxino (2009) also confirm HD=3 up to 2039 bits for 16 bit Fletcher checksum. This is also established by this research.

A different way to understand the performance structure of Fletcher checksum is to view the consecutive sums as half the size of FCS. In this way, Fletcher checksum of 8 bits has HD=2 performance beginning at slightly above 15 dataword chunks of 4 bits. 16 bit Fletcher checksum

has performance of HD=2 beginning at 255 dataword chunks of 8 bits and 32 bit checksum has HD=2 beginning at 65535 dataword chunks of 16 bits.

Table 5.5: Fletcher Checksum Error Detection.

Size of Checksum	**Simulated unidentified fragment of 2 bit errors**	**Simulated unidentified fragment of 3 bit errors**
8 bit Fletcher: (56 bit)	0	0.00385
(64 bit)	0.00151	0.00383
(1024 bit)	0.00783	0.00319
16 bit Fletcher: (1024 bit)	0	5.26e-5
(2032 bit)	0	4.71e-5
(2048 bit)	3.72e-6	4.87e-5
32 bit Fletcher: (1024 bit)	0	2.79e-5
(65536 bit)	0	3.31e-7

Table 5.5 illustrates that Fletcher checksums attain a superior HD compared to addition checksums up to a specific length established by checksum size.

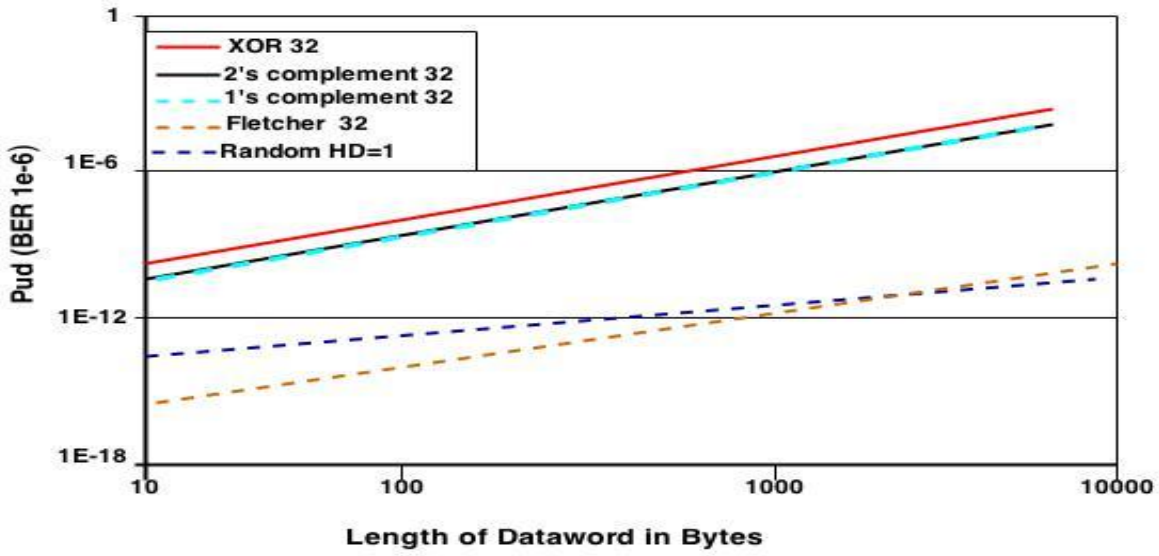


Fig 5.3: Evaluation of XOR, 1's complement, 2's complement, and Fletcher Checksum.

Figure 5.3 exhibits that performance of Fletcher checksum is notably superior to other checksums analyzed and has blending properties that are moderately better than a random hash for all small and middle sized datawords, but inadequate for long dataword sizes. The reason is that while addition function provides Fletcher checksum only mediocre mixing. It has HD=3 rather than HD=2 performance at those lengths and the HD additional bit covers up for the inadequate blending of this BER value. It should be observed that in this research figures reflect dataword lengths in bytes, whereas tables reflect dataword lengths in bits.

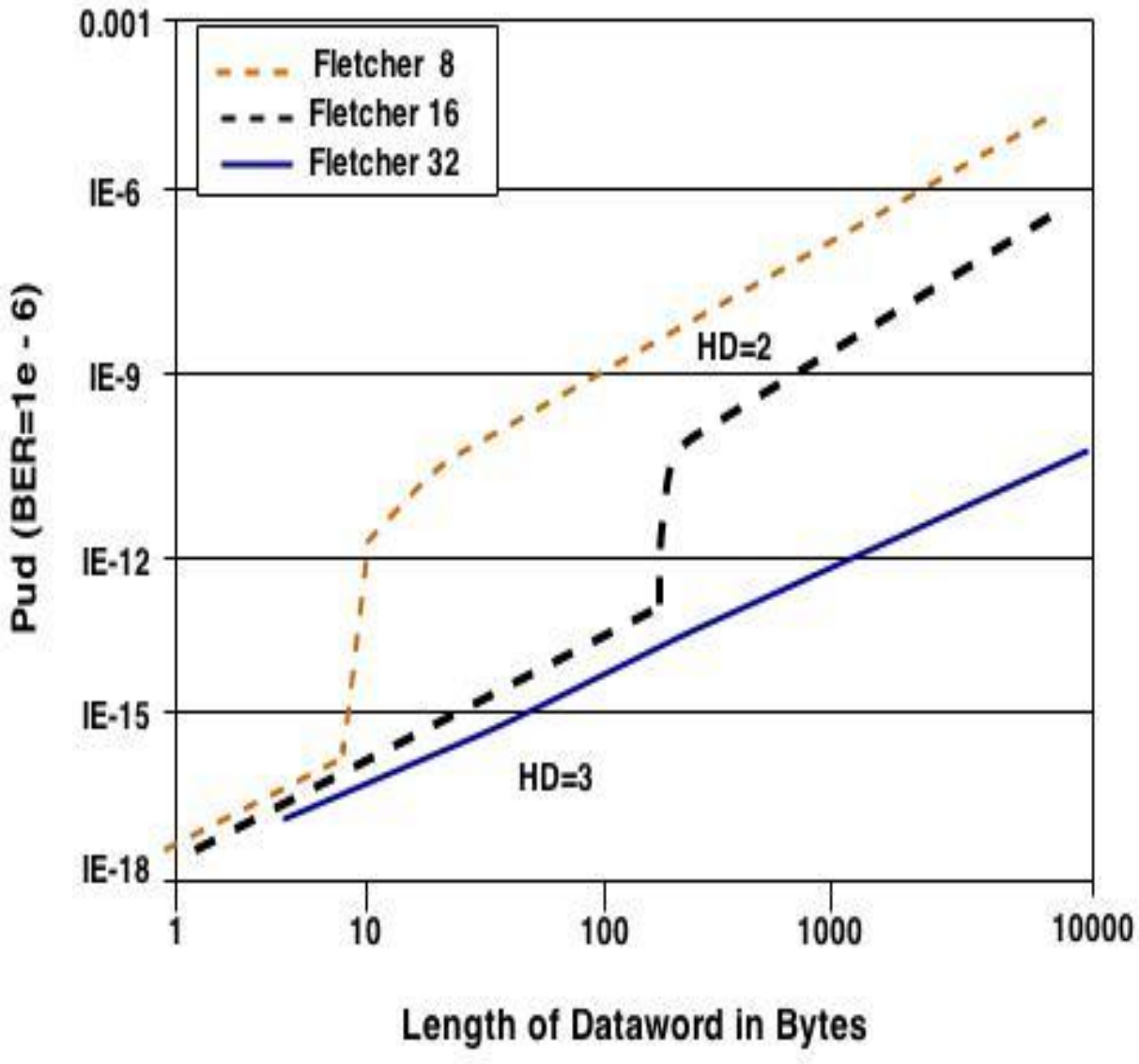


Fig 5.4: Performance of Fletcher Checksum.

Figure 5.4 exhibits an analogy of 8 bit, 16 bit, and 32 bit Fletcher checksum performance. The considerable reduction in Pud for the 8 bit and 16 bit bends relates to the positions where the performance falls from HD=3 to HD=2. This alteration is considerably larger than the distance between bends where they have similar HD, stressing the significance of HD provided a BER fault design.

5.7 Adler Checksum

Adler checksum is similar to Fletcher checksum, with the exception that addition modulo utilized by Adler checksums are less than the appropriate power of 2 for the block size being added (Deutsch and Gailly, 1996). Practically this implies that an 8 bit block size is added modulo 251 and a 16 bit block size is added modulo 65521.

The utilization of a prime modulus gives superior blending of bits, but happens at the expense of having lesser authentic check pattern values. The reason being that some FCS values are never produced because of modular division. For instance, for an 8 bit block size, the values of 251, 252, 253, 254, and 255 are unauthentic under modulus 251 and hence cannot materialize in any byte of the 16 bit check pattern, providing few probable check pattern values and moderately enhancing the possibility of a multi bit error by accident generating an authentic codeword. Tradeoff of superior blending with a prime modulus vs. lesser probable check pattern values is analyzed by Koopman and Maxino (2009). Typically, Adler checksum is inferior to Fletcher checksum and only enhances error identification moderately in areas where it is superior. Because of costly calculations needed, this checksum was not analyzed beyond this point. Generally, Adler checksum is not advocated as a beneficial tradeoff. Because of the complicated calculations needed, a CRC is recommended to be utilized rather than an Adler checksum.

5.8 Cyclic Redundancy Code

Brief:
Cyclic Redundancy Code (CRC) Calculation.

Hamming Distance: Fluctuates depending on response polynomial.
For long lengths HD=2.
For shorter lengths HD relies on clearly defined polynomial and length of

dataword.

Coverage of Burst Error: Block size

Data Dependency: No

Identify Modification in Data Pattern: Yes, depending on HD and unidentified error fragment.

CRC-s are mathematically founded on polynomial division over Galois field GF (2). This implies that dataword is believed to be a polynomial in which every bit has a climbing power of variable 'x' with a '1' or '0' coefficient relying on data word's binary value at that bit location. A 'CRC polynomial' is utilized to divide the dataword and the remainder is utilized as the value of FCS. This procedure of division can be accomplished through a shift and XOR process for every dataword bit, however a lot more effective execution approaches are utilized (Ray *et al.*, 2006). While a CRC calculation can be more time consuming than the calculation of a checksum, it can generate significantly superior error identification outcomes. The properties of error identification are mainly established by the specific CRC polynomial utilized.

Some error identification properties of CRC's deserve mentioning. Error identification for the BER fault design is entirely independent of dataword values. The error identification ability for the bit inverse of a CRC polynomial is similar to the primary CRC polynomial, even though the real value calculated is distinct.

Figure 5.5 illustrates IEEE 802.3 performance of CRC-32 error detection in comparison to checksums. It attains superior HD than all kind of checksums illustrated. Performance of other CRC' may be considerably better (attaining up to HD=6 for datawords as long as maximum length of Ethernet message and even superior HD attainable for smaller datawords).

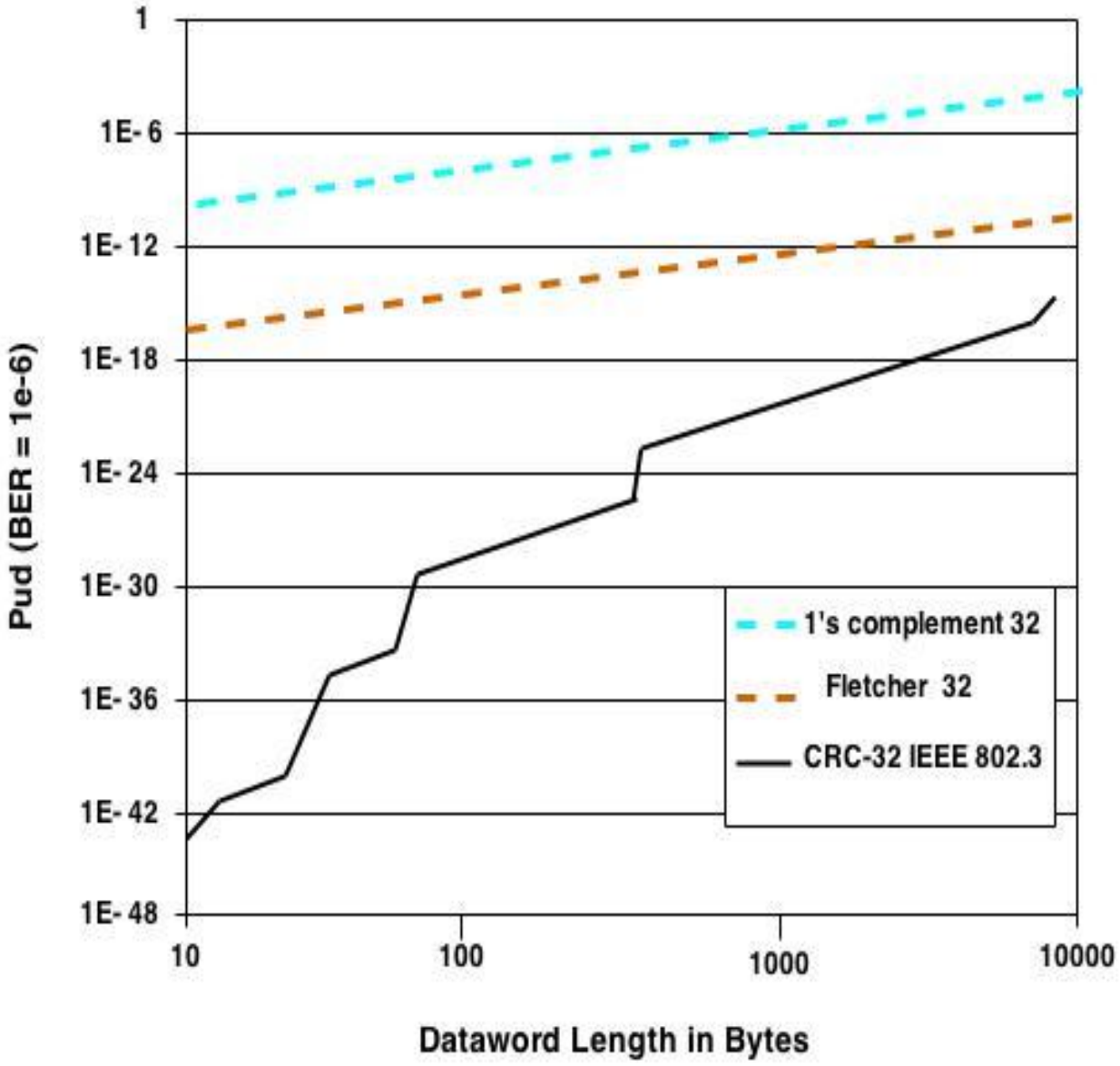


Fig 5.5: Performance of CRC-32.

Table 5.6 exhibits commonly ‘better’ CRC polynomials for diverse HD values. ‘Better’ implies that they have a specific HD out to the longest length of dataword of any polynomial of that size, and at smaller length of dataword even high HD. The lengths displayed are the maximum length of dataword for which that polynomial attains the expressed HD. The number displayed in the column is the highest number of data bits for that HD. The table exhibits some beneficial polynomials that are in +1 notation. Hence, for instance 0xA6 has HD=4 up to 15 bits, HD=3 up to 247 bits, and HD=2 at entire lengths and above 248 bits.

Table 5.6: CRC Polynomials Error Detection (Length in bits).

Polynomial	**HD=2**	**HD=3**	**HD=4**	**HD=5**	**HD=6**
0xA6	Good	247	15		
0x97	Long	---	---	9	
0x9C	Long	---	119		
0x8D95	Good	65519	1149	62	19
0xC86C	Long	---	---	---	135
0xAC9A	Long	---	---	241	35
0xD175	Long	---	32,751	---	54
0xBAAD	Long	---	7985	108	20
0x80000D	Good	16,777,212	5815	309	
0xBD80DE	Long	---	4074	---	2026
0x9945B1	Long	---	8,388,604	---	822
0x98FF8C	Long	---	---	4073	228
0x80000057 (Maxino ,2006)	Good	4,294,967,292	---	2770	325
0x82608EDB (CRC-32)	Long	4,294,967,292	91607	2974	268
0x90022004 (Koopman, 2002)	Long	---	65506	---	32738
0x8F6E37A0 (Koopman, 2002)	Long	---	2,147,483,646	---	5243
0xD419CC15	Long	---	---	65505	1060
0x80002B8D (Maxino, 2006)	Long	---	2,147,483,646	---	3526
0xBA0DC66B (Koopman, 2002)	Long	---	114,663	---	16360

Dataword lengths smaller than 8 bits are excluded from the table, as are the existence of any HD>6 lengths. The expression '---' implies that there are no dataword lengths for which that HD is given. The expression 'good' implies that for long dataword lengths it is a "good" polynomial to utilize. The expression "long" implies that HD=2 performance at dataword lengths longer than the highest number in other columns for that polynomial, and that the performance is not as robust as the "good" polynomial for long dataword lengths at that size. All CRC performance numbers are logically precise in the table and the evaluation, with the exception where utilization of estimation in the evaluation is expressed clearly.

For scenarios in which many altered bits are anticipated, the estimate unidentified error fragment for large numbers of bit errors is pertinent. For large number of bit errors CRC's give quality performance to all kinds of checksums. The reason is that they give quality bit blending, along with good HD. This attribute was not precisely quantified through all CRC's because of computational unsuitability for many error codes analyzed, but founded on prior research evaluating various distinct CRC'. For large number of bit errors, typically the unidentified error fragment for CRC' approaches $1/((2^k) - 1)$. "-1" expression is due to the reason that any nonzero value cannot generate an FCS of zero, decreasing the number of probable FCS values by 1 for each value of dataword except all zeros. This is nearly as robust as a good hash function, except with the benefit of HD=2 at entire dataword lengths. The anomaly is that for CRC polynomials divisible by (x+1), all odd number of bits errors are identified- however with the expense of an equivalent duplication of the unidentified error fragment for even number of bit errors to roughly $1/(2^{(k-1)})$.

5.9 Discussion

This section provides a brief summary of the outcomes of testing phase.

Effectiveness of error detection relies on various elements involving the data size to be safe guarded, kinds of errors anticipated, the smallest number of bit errors that are unidentifiable by hamming distance, and the unidentifiable error fragment of errors as a function of number of bit errors existent in a specific chunk of data. Fletcher checksums give considerably superior error identification than other types of checksums. In various scenarios CRC's provide considerably superior performance than checksums at short to medium length of dataword

because of higher HD, and provide moderately superior performance even for extremely long dataword lengths. In many scenarios a random independent bit error fault design (bit-inverse) is the ideal method to analyze the performance of CRC and checksum. Primary implant values, though possibly useful for identifying some flaws like additional leading zeros, do not influence the performance of CRC error identification. Performance of checksums relies on data values, with a data blend of 50% 1's and 50% zero's bits within the data normally culminates in inferior performance of error identification. Error detection performance of CRC is not dependent on data values being operated by the CRC.

Chapter 6: Error Identification Performance and Calculation Expense Tradeoffs

6.1 Error Identification Performance

In datalink layer there are distinct approaches for identifying errors. However, not all approaches of error identification can identify errors precisely and adequately. Besides single, double, and burst errors, kinds of errors anticipated could be short term errors or permanent errors. Short term errors include unanticipated alteration to data rather than damages to physical channel. These errors can be originated by supply voltage changes, generated voltage malfunction, etc. Short term errors are normally handled by error detection with retransmission. Permanent errors are normally an outcome of manufacturing defects and can only be handled by replacing unusual components with spare devices, or detouring around the broken region (Yu, 2011).

XOR checksum approach is uncomplicated and straight forward and identifies all single bit errors. CRC has superior performance in identifying single bit errors, double bit errors, odd number of errors, and burst errors. 1's and 2's complement checksum are not as effective as CRC in error identification. Fletcher checksum was created to present error identification performance close to CRC, with less calculation costs.

6.1.1 Performance of XOR

XOR checksum identified all 1 bit errors. Burst errors were also identified by XOR only if the total number of errors in every data unit was odd/even. It was unable to identify errors where the total number of bits reversed was even. If any 2 bits were reversed in transmission, the reversed bits canceled each other out and the data unit passed XOR check, even though it was flawed.

6.1.2 Performance of 1's and 2's Complement Checksums

The Conventional 1's and 2's complement sums utilizes 16 bits to identify errors in a message of any size. It can identify all errors including an odd number of bits as well as most errors including an even number of bits. But they were found to be less robust than CRC in identifying errors. For instance, when the value of one word was increased and the value of other

was decreased by the same amount, 2 errors were not identified because the sum and checksum remained similar. Also when the values of various words were increased and the total change was a multiple of 65535, the sum and checksum didn't change which means that errors were not identified.

6.1.3 Performance of Fletcher Checksum

Some weighted checksums are suggested by Fletcher (1982), in which each word is multiplied by its weight that is related to its location in the text. The multiplication eliminated the first problem mentioned in the previous section.

6.1.4 Performance of CRC

CRC is a powerful approach used for identifying errors. Unlike XOR CRC is founded on binary division. Instead of addition of bits a pattern of redundant bits known as CRC (remainder) was attached to the end of data unit so that it is precisely divisible by a generator/divisor. CRC gave a robust performance in identifying signal, double, odd and burst errors. It can be executed both in hardware and software. It is more robust when implemented in hardware.

Table 6.1 and 6.2 presents error detection rates of error codes and CRC analyzed. Table 6.3 provides strengths and weaknesses of error codes.

Table 6.1: Error detection capability of 32- bit checksums

Error detection code	**Error Detection Coverage**			
	HD	Single bit errors	Double bit errors	Burst error
XOR-32	2	100%	96.8%	94.1%
1's Complement-32	2	100%	98.4%	96.7%
2's Complement-32	2	100%	98.3%	96.6%
Fletcher-32	3	100%	99.9%	99.8%

Table 6.2: Error detection capability of CRC-32

Type of Error	Error Identification Performance
Single bit errors	100%
Double bit errors	100% as long as the generator polynomial has at minimum three 1's (it does).
Odd number of bits in error	100% as long as generator polynomial has a factor $x+1$ (it does).
Error burst of length less than $k+1$	100%
Error burst of length equal to $k+1$	99.5% (almost 100%)
Error burst of length greater than $k+1$	99.7% (almost 100%)

Table 6.3: Strengths and weaknesses of error detection approaches

Approach	Strengths	Weaknesses
XOR	Uncomplicated and easy to utilize. Only needs one additional bit per byte.	At times two errors in one byte may cancel each other out and cannot be identified.
16 bit-Checksums	Assure that full complement of bits is received.	Needs additional bits to be transmitted. Does not identify all errors in the same number of one's and zero's.
CRC	Identifies large number of errors and can be used for a big packet of data.	Needs an additional 16 or 32 bits be transmitted with every packet of data, so can be remotely slower than other approaches.

6.2 Calculation Expense Tradeoffs

Choosing an optimal checksum for an application is generally not founded on error identification properties solely. Other factors like calculation expenses repeatedly play an important part in selecting checksums.

Feldmeier (1995) research on swift software executions for checksum algorithms illustrates that 1's complement addition is roughly twice as fast as Fletcher checksum and Fletcher checksum is at minimum twice as fast as CRC.

Compared to all checksum algorithms XOR checksum has the smallest calculation cost. But, it also has inferior error identification properties compared to all checksums. The error identification properties of XOR don't notably adjust with length of codeword.

If implemented in hardware 2's complement addition checksum (also called add checksum) has a moderately superior calculation cost than XOR because of transmission delay of carry bits. If implemented in software 2's complement checksum has identical calculation cost as XOR checksum. In error identification properties a 2's complement checksum is almost twice as good as XOR checksum at minimal or has no additional expense. This is due to the fact that bit "blending" between bit locations of data chunks is achieved through bit by bit carries of binary addition. The usefulness of the "blending" relies on the data being added, which establishes the structure of carry bits through different bit locations. Error identification capability is not notably influenced by the length of codeword.

For some hardware executions, 1's complement checksum has no extra cost over 2's complement checksum. In software executions, it has moderately superior calculation expense than 2's complement because of the integration of MSB carry bit. It catches up for this moderately superior cost by being moderately superior at error identification than 2's complement checksum. Codeword length does not influence the properties of error identification. 1's complement checksum should generally be utilized rather than XOR and 2's complement checksum, except if there is extremely convincing justification for doing otherwise.

Fletcher checksum has twice the calculation expense of 1's complement checksum because of it comprising of two consecutive sums rather than one. However, at long lengths of codeword it is at minimum a sequence of magnitude superior. For small lengths of codeword, it is a number of sequences of magnitude superior than 1's complement because of HD=3 error identification.

Adler checksum, because of its utilization of prime modulo has moderately superior calculation cost than Fletcher checksum. It has no more than equivalent error identification

property to Fletcher checksum. Similar to Fletcher checksum its error identification capability also goes down after a specific length of codeword. For small codeword lengths, whenever provided with an option Fletcher checksum should invariably be utilized. It has a low cost of calculation and also it provides comprehensive error identification properties.

For all checksum algorithms CRC has by far the top most calculation expense. It is usually twice the calculation cost of Fletcher checksum. However, it has the most superior properties for error identification for all checksums. For the same checksum size, best CRC polynomial is sequence of magnitude superior to Fletcher checksum for lengths of codeword less than 2^k (k checksum size). For lengths of codeword longer than this, best CRC polynomial is roughly a sequence of magnitude superior than Fletcher checksum. All the checksums that were analyzed, CRC has the most significant variation in error identification capability regarding length of codeword.

To present as the confirmation of Koopman and Maxino (2009) research, the calculation cost of the authentic code utilized to attain the data for this research was established. As anticipated XOR and add checksums have precisely the identical cost when executed in software. 1's complement checksum had a calculation expense of 1.8 times that of 2's complement checksum. Fletcher checksum had a calculation expense of 4 times that of 2's complement checksum. The rationale is that the non-optimized version of Fletcher checksum was utilized. According to Koopman and Maxino (2009) Fletcher checksum can be optimized to twice the calculation expense of add checksum.

Figure 6.1 illustrates that the smaller length of codeword the more advantage of utilizing a CRC in comparison to other checksum algorithms. For lengths of codeword more than 2^k (k checksum size), the advantage of utilizing a CRC goes down abruptly due to it giving HD=2 error identification performance. Hence, though it may be hard to explain enhanced calculation expense of utilizing a CRC for large datawords in usual organization and desktop surroundings, situation is relatively distinct for short datawords (less than 100 bits) normally seen in embedded networks. For network applications that are embedded, utilizing a CRC can give superior error identification performance. Maxino (2006) provides CRC polynomials that have superior error identification performance at nearly similar expense as typical CRC polynomials. Though,

Fletcher checksum can give HD=3 at small lengths of dataword, it is surpassed by a CRC at entire data lengths by a minimum of one bit of HD.

Obviously, these performance evaluations are merely estimation ratios and numerous elements establish the optimal option for a specific application. But, the most common belief in broad communication that Fletcher checksum is nearly as optimal as a CRC at drastically less calculation expense are not actually correct for applications that are embedded. Checksums other than CRC surrender sequences of magnitude in error identification efficiency in exchange of swiftness and speed. Additionally, applications that utilize a XOR checksum could have considerably superior error identification for basically same calculation expense merely by utilizing a 2's complement checksum or possibly a 1's complement checksum.

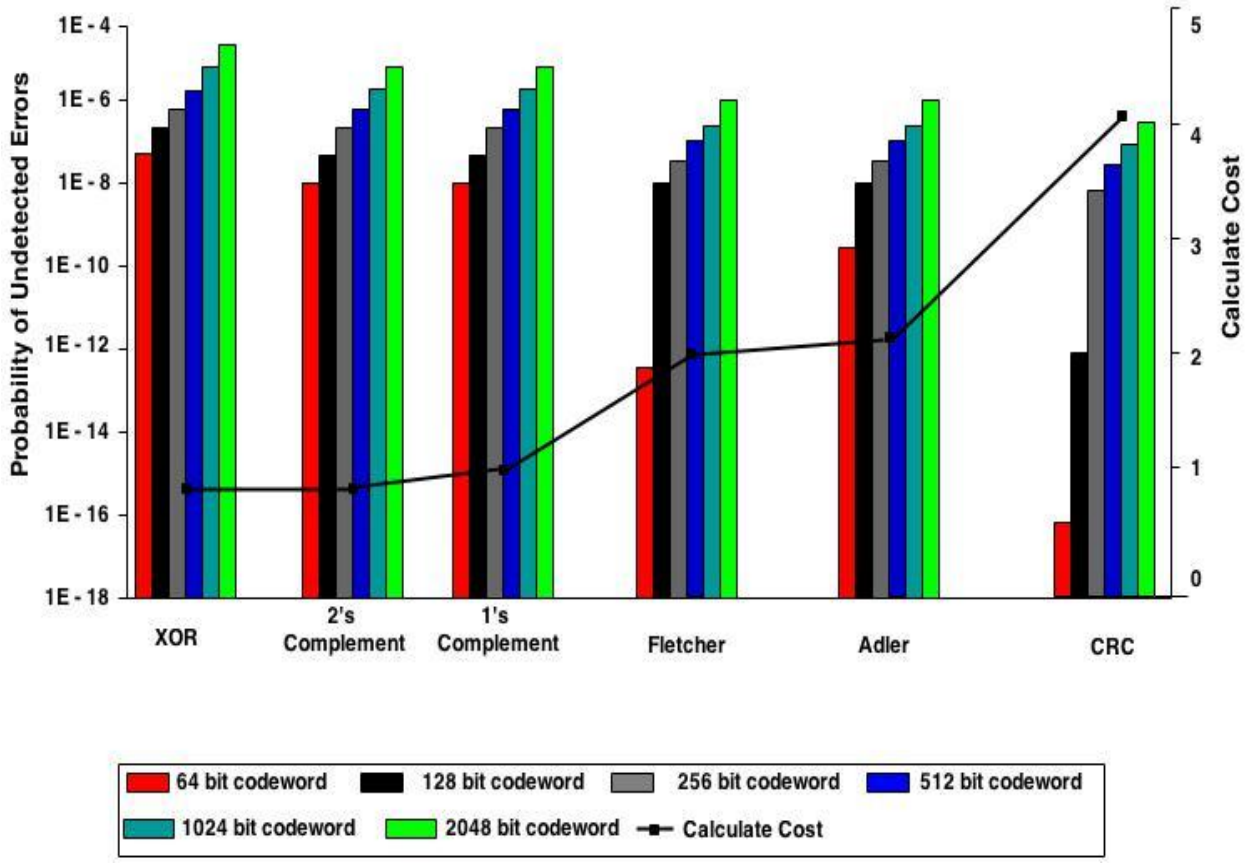


Fig 6.1: Calculation Expense Tradeoffs for 8 bit checksums (BER=10^{-5}).

Chapter 7: Conclusion, Recommendations and Future Research

7.1 Conclusion

Error identification properties of checksums differ immensely. The possibility of unidentified errors for a k bit checksum is not always $1/2^k$ in practical applications as is often considered. Instead, it depends on elements like the kind of algorithm utilized, codeword length, and the kind of data inside the message. The common establishing element of error identification performance is the algorithm utilized with clear distinctions obvious for small messages that are common of networks that are embedded.

Even for messages that are slightly long, error identification performance for random independent bit errors on random data should be regarded as possibly superior design than $1/2^k$ (k checksum size). Conduct on small number of bit errors appears to be a restricting element of complete error identification performance.

Founded on our research of unidentified error possibilities, for applications where it is notable that burst errors are the influential source of errors, XOR, 2's complement and CRC checksums give superior error identification performance than 1's complement and Fletcher checksums.

For other applications, a CRC polynomial, whenever practical must be utilized for error identification reasons. It gives a minimum of one extra bit of error identification ability in comparison to other checksums, and accomplish it at merely a factor of two to four times higher expense of calculation. In applications where cost of calculation is a serious limitation, Fletcher checksum is normally a better option. Fletcher checksum has lesser calculation cost and opposite to wide spread impression, is also more efficient in many scenarios. For applications that are seriously restricted, 1's complement checksum should be utilized, but if practical, with 2's complement is a less efficient substitute. There is typically no rationale to persist with the general exercise of utilizing an XOR checksum in contemporary designs. The reason being that it has identical software calculation expense as an addition checksum however is merely half as efficient at identifying errors.

7.2 Recommendations

Utilizing CRC's and checksums in applications needs considering various elements. These elements involve alteration of field length weakening HD, physical layer encoding error identification code interactions, encryption and error identification code interactions, and mutually related data errors created by data layout structures. Error identification methods and the data that they safeguard should be cleansed at regular intervals in order to prevent a buildup of undiscovered flaws. In addition, complicated circuits may initiate mutually related bit errors or alternatively weaken the efficiency of error identification. Each of these elements has the likelihood to unauthenticate any calculations utilized to establish the efficiency of checksums and CRC's.

Utilizing various error identification codes in synchrony can assist in identifying errors, but not at all times to the levels anticipated. Commonly, implementing various error identification codes to the same dataword will possibly not enhance HD. But, nesting error identification codes (various small datawords each safeguarded separately with an all embracing error identification code) can be beneficial and may possibly culminate in enhanced HD for the combined method.

As to which error identification code to utilize, there is no one size fits all solution. A recommended method to establish if a specific error code fulfills integrity needs has subsequent levels:

i. Establish which functions should be contemplated.
ii. Establish the maximum tolerable failure possibility for every function.
iii. Establish vulnerability to failure.
iv. Calculate maximum permitted failure possibility.
v. Establish error design.
vi. Establish which errors can be threatening.
vii. Calculate threatening error possibilities.
viii. Calculate needed error coding coverage and if that coverage is achieved by suggested error code.

When establishing whether unidentified error possibilities are sufficiently low, it can be adequate to select an error code with an adequately large HD, which assures that all errors that are anticipated to transpire within the functional life of a system are ensured to be identified (provided that a random independent fault design precisely shows the errors that will be detected in operation).

When implementing an error identification code, it is significant to assure that presumptions (BER are guaranteed) to be accurate not only at the time of design but also throughout operations. Determining precise error design is crucial in establishing the possibility of error identification "escapes".

The following are the suggestions that are not recommended for CRC and checksum utilization:

i. Selecting a CRC depending upon commerciality instead of analysis.
ii. Thinking that a good checksum is just as adequate as an inadequately selected CRC.
iii. Analyzing randomly altered data identification ability when a BER design is more suitable.
iv. Utilizing factorization or other elements without reasoning, to select a CRC rather than logically detailed evaluation.
v. Unsuccessful in safeguarding the length field of the message.
vi. Unsuccessful in selecting a precise fault design and implementing it.
vii. Disregarding error codes and symbol encoding interactions (involves stuffing and scrambling).

7.3 Future Research

There is scope for future related research. In a number of systems it is significant to evaluate other fault designs. For instance, bit slip in communication networks, in which a bit is practically duplicated or erased from the middle of a bit stream because of timing and synchronization flaws.

In order to handle short term and manufacturing errors, extensive research into a packet reorganization algorithm combined with shortening error control coding approach is needed.

Physical layer interactions and error coding is definitely a problem that can undermine error identification efficiency. Additional research is required to develop upon the research of Bulpin, Glick, Moore, and James (2006) and more commonly assure that protocol framing, bit encoding, scrambling and other methods do not undermine error identification efficiency. There are some physical layer characteristics that will enhance performance in some systems, like utilization of Manchester encoding (RZ) bits where the system declines incorrectly structured bit sequences (Koopman & Chakravarty, 2001).

Because of the impact of physical layer interactions, it may be that long CRC's are wished for in order to alleviate issues. Research on CRC and checksum efficiency at 64-bit is limited and should be analyzed as sizes of payload, sizes of memory and rate of data enhances.

It appears realistic that cryptographically protected hash functions may be appropriate for some applications where HD=1 is tolerable. But additional research should be accomplished to make sure that there are no delicate spots for specific hash functions, with regards to giving a crucially 'Ideal' blending function for error identification reasons.

The utilization of numerous layers of error identification codes is not fully comprehended. For instance, many layers of CRCs, fluctuating CRC's, or amalgamation of checksums and CRC's is suggested repeatedly by developers but is not fully acknowledged. While there may be an approach of assuring that such layers enhances attained HD, till now no such approach has resulted in success.

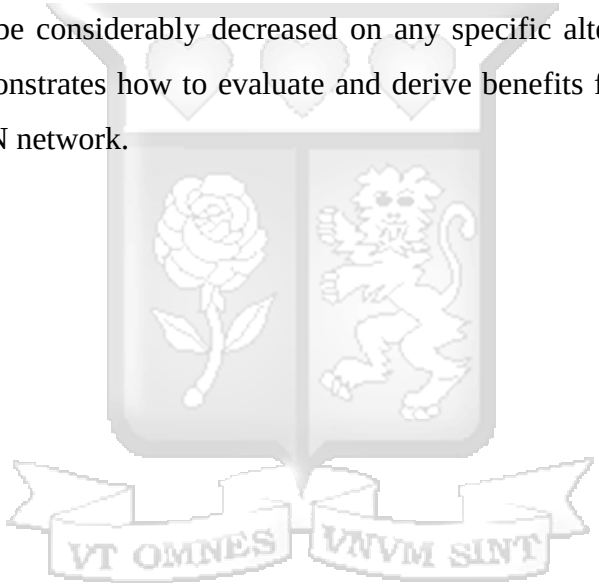
Interactions between encryption, compression, and error identification are not properly appreciated, particularly when the error identification code is vulnerable to compression, encryption, or other alterations. In addition, while it is typical to claim that encryption gives data integrity protection, which is only accurate if there is an approach to authenticate the decrypted data in a dependable way. For now HD=1 for authenticating decrypted data should be presumed, unless substantiated otherwise.

The present Monte Carlo simulations give rationally current outcomes and the appearance of the curves indicates that the established restrictions are near to real restrictions.

Specifically, simulation outcomes exhibit where the lower HD is present. However, there is a probability that a solitary unidentified error exists that went unnoticed by the simulation.

Measuring the efficiency of Fletcher checksum and CRC's for identifying data sequencing errors is not properly grasped. Practically, it may be suitable to have a superior standard error identification code for every piece of a fragmented data set that secures a sequence number.

Lastly, an important area for future research could be to investigate system level impact on error identification efficiency. For instance, if a control system can be built particularly to outlast a finite number of randomly altered messages within a restricted time period, error identification needs can be considerably decreased on any specific altered message. Koopman and Szilagyi (2009) demonstrates how to evaluate and derive benefits from this kind of system capability utilizing a CAN network.



References

Baicheva, T., Dodunekov, S., & Kazakov, P. (1998). On the cyclic redundancy-checkcodes with 8-bit redunduancy. *Computer Communications, 21(11).*1030-1033. http://dx.doi.org/10.1016/s0140-3664(98)00165-0

Baicheva, T., Dodunekov, S., & Kazakov, P. (2000). Undetected error probability performance of cyclic redundancy-check codes of 16 bit redundancy. *IEEE Proc., Commun., 147(5),* 253-256. http://dx.doi.org/10.1049/ip-com:20000649

Bluetooth specification. (2000). Retrieved 8 March, 2015 from http://www.bluetooth.com/developer/specification/specification.asp

Bosch, R. (1991). *Controller Area Network (CAN) Specification, Version 2.0.* http://www.can.bosch.com

Braden, R., Borman, D., & Partridge, R. (1988). *Computing the Internet Checksum*. IETF-RFC 1071.

Brown, F., & Waldvogel, M. (2001). Fast incremental CRC updates for IP over ATM networks. In *Workshop on High Performance Switching and Routing* (pp. 48-52). IEEE.

Bulpin, J., Glick, M., Moore, A., & James, L. (2006). Physical Layer Impact upon Packet Errors. In *Passive and Active Measurement Workshop*. PAM 2006.

Campobello, G., Patane, G., & Russo, M. (2003). Parallel crc realization. *IEEE Transactions on Computers, 52(10),* 1312-1319. http://dx.doi.org/10.1109/tc.2003.1234528

Castagnoli, G., Ganz, J., & Graber, P. (1990). Optimum cyclic redundancy-check codes with 16-bit redundancy. *IEEE Transactions on Communications, 38(1),* 111-114. http://dx.doi.org/10.1109/26.46536

Castagnoli, G., Brauer, S., & Hermann, M. (1993). Optimization of cyclic redundancy-check codes with 24 and 32 parity bits. *IEEE Transactions on Communications, 41(6),* 883-892. http://dx.doi.org/10.1109/26.231911

Carvey, P. (1996). Designing for wireless LAN communications. *IEEE Circuits and Devices Magazine, 12(4),* 29-33. http://dx.doi.org/10.1109/101.526877

checksum - definition of checksum in English from the Oxford dictionary. Retrieved 13 March http://www.oxforddictionaries.com/definition/english/checksum

Checksums. Retrieved 13 March 2015 from http://www.technopedia.com/definition/english/checksum

Christensen, W. (1982). *Modem Protocol Documentation.* Retrieved from http://www.textfiles.com/apple/xmodem

Chun, D., & Wolf, J. (1994). Special hardware for computing the probability of undetected error for certain binary CRC codes and test results. *IEEE Transactions on Communications, 42(10),* 2769-2772. http://dx.doi.org/10.1109/26.328943

Cheong, S., Barnes, E., & Friedman, D. (1979). "On some properties of the undetected error probability of linear codes," *IEEE Trans. Inform, Theory,* vol. IT-25, 110-112.

Daniel, J., Costello, J., Hagenauer, J., & Wicker, J. (1998). Application of error control coding. *IEEE Transaction on Information Theory, 38(1),* 2531-2560.

Deutsch, P., & Gailly, J. (1996). Zlib Compressed Data Format Specification Version 3.3. IETF RFC 1950.

Definition of CHECKSUM. Retrieved 13 March 2015 from http://www.merriam-webster.com/dictionary/checksum

Dunbar, R. (2001). *Embedded Networks, Pervasive Low Power, Wireless Connectivity*. (Published Doctoral Dissertation). Massachusetts Institute of Technology, Cambridge, USA.

Emware. (2000). Retrieved 5 March , 2015 from http://www.emware.com/solutions.emit.

exclusive OR - definition of exclusive OR in english from the oxford dictionary. Retrieved from http://www.oxforddictionaries.com/definition/english/exclusive-OR

Feldmeier, D. (1995). Fast software implementation of error detection codes. *IEEE/ACM Transactions on Networking, 3(6),* 640-651. http://dx.doi.org/10.1109/90.477710

Fletcher, J. (1982). An Arithmetic Checksum for Serial Transmissions. *IEEE Transactions on Communications, 30(1),* 247-252. http://dx.doi.org/10.1109/tcom.1982.1095369

FlexRay Consortium (2005). FlexRay Communication Systems Protocol Specification, Version 2.1 Revision A, Stuttgart: FlexRay Consortium GbR.

Forouzan, B. (2007). *Data Communications and Networking* (4th ed.). McGraw Hill.

Fujiwara, T., Kasami, T., Kitai, A., & Lin, S. (1985). On the Undetected Error Probability for Shortened Hamming Codes. *IEEE Transactions on Communications, 33(6),* 570-574. http://dx.doi.org/10.1109/tcom.1985.1096340

Fujiwara, T., Kasami, T., & Lin, S. (1989). Error detecting capabilities of the shortened Hamming codes adopted for error detection in IEEE Standard 802.3, *IEEE Transactions on Communications, 37(9),* 986-989. http://dx.doi.org/10.1109/26.35380

Funk, G. (1996). Determination of best shortened linear codes. *IEEE Transactions on Communications, 44(1),* 1-6. http://dx.doi.org/10.1109/26.476086

Gershenfeld, N. (1999). When Things start to Think. *Foreign Affairs, 78(6),* 148. http://dx.doi.org/10.2307/20049565

IEEE. (1999). IEEE Std 802.11, 1999 Edition. "Information Technology-Telecommunications and information exchange between systems". IEEE Standards Association. ISBN 0-7381-1809-5.

IrDA. (1998). *Infrared data association, IrDA specification*. Retrieved 8 March, 2015 from http://www.IrDA.org

Kazakov, P. (2001). Fast calculation of the number of minimum-weight words of CRC codes. *IEEE Trans. Inform. Theory, 47(3),* 1190-1195. http://dx.doi.org/10.1109/18.915680

Kaisi, T., & Kitakami, M. (2002). Single bit error correcting and burst error locating codes, in Information theory. *In The Information Theory*, IEEE.

Kelly, K. (1997). New rules for the new economy. Twelve dependable principles for thriving in the turbulent world, *Wired Magazine,* 140-197. Retrieved from http://www.wired.com/archive/5.09/newrules.html

Kothari, C. R. (2004). *Research methodology: methods and techniques*. New Age International.

Koopman, P. (2002). 32-bit Cyclic Redundancy Codes for Internet Applications. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks.*(pp. 459-472). Washington, D.C: IEEE Computer Society.

Koopman, P., & Chakravarthy, T. (2004). Cyclic Redundancy Code (CRC) Polynomial selection for Embedded Networks. In *Proceedings of the International Conference on Dependable Systems and Networks* (pp. 145-154). IEEE.

Koopman, P., & Chakravarthy, T. (2001). *Analysis of the train communications network protocol error detection capabilities.* Carnegie Mellon University technical report.

Koopman, P., & Szilagyi, C. (2009). A flexible approach to embedded network authentication. In *Proceedings of the International Conference on Dependable Systems and Networks.* (pp. 165-274). IEEE.

Koopman, P., & Maxino, T. (2009). The Effectiveness of Checksums for Embedded Control Networks. *IEEE Transactions on Dependable and Secure Computing, 6(1),* 59-72. http://dx.doi.org/10.1109/tdsc.2007.70216

Lian, Q., Zhang, Z., Wu, S., & Zhao, Y. (2005). Z-Ring: Fast prefix routing via a low maintenance membership protocol. In *International Conference on Network Protocols.* (pp. 132-146). IEEE.

Maxino, T. (2006). *The Effectiveness of Checksums in Embedded Networks.* (Master's thesis, Carnegie Mellon University, Pennsylvania).

Matsumoto, M., & Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul., 8(1),* 30. http://dx.doi.org/10.1145/272991.272995

McAuley, A. (1994). Weighted sum codes for error detection and their comparison with existing codes. *IEEE/ACM Transactions on Networking, 2(1),* 16-22. http://dx.doi.org/10.1109/90.282604

McShane, I. *Communications Protocol – McShane, Inc. McShaneinc.com.* Retrieved from http://www.mcshaneinc.com/html/Library_CommProtocol.html

Minar, M., Grey, M., Roup, O., Krokorian, R., & Maes, P. (1999). *Hive: Distributed Agents for Networking Things.* Retrieved 7 March, 2015, from http://xenia.media.mit.edu/~nelson/research/hive-asama99/asama-html/paper.html

Moravec, H. (1999). Robot: Mere Machine to Transcendent Mind. *Foreign Affairs, 78(3),* 131. http://dx.doi.org/10.2307/20049291

Modicom, Inc. (1996). Modbus Protocol Reference Guide, pI-MBUS-300 REV-J.

Nakassis, A. (1988). Fletcher's error detection algorithm: How to implement it efficiently and how to avoid the most common pitfalls. *ACM SIGCOMM Computer Communication Review, 18(5),* 63-88. http://dx.doi.org/10.1145/53644.53648

National Institute of Standards and Technology., Retrieved 27 March 2015 from http://www.nist.gov/dads/HTML/xor.html

"*Part3: Carrier Sense multiple access with collision detection (CSMA/CD) access method and physical layer specification", i-1515*. IEEE standard 802.3-2000.

Pentland, A. (1998). Smart Rooms, *Sci Am, 274(4),* 68-76. http://dx.doi.org/10.1038/scientificamerican0496-68

Peterson, L., & Davie, L. (2007). *Computer networks, a systems approach* (3rd ed.). Elsevier.

Peterson, W., & Brown, D. (1961). Cyclic codes for error detection. In *Proceedings of the IRE, 49,* 228-234.

Prange, E. (1957). "*Cyclic Error Correcting Codes in Two Symbols*". AFCRC-TN-57-103, ASTIA, Document No. AD 133749, Air Force Cambridge Research Centre.

Plummer, W. (1989). TCP checksum function design. *ACMSIGCOMM Computer Communication review, 19(2),* 95-101. http://dx.doi.org/10.1145/378444.378454

Ray, J., & Koopman, P. (2006). Efficient high Hamming distance CRCs for embedded networks. In *Proceedings of the International Conference on Dependable Systems and Networks* (pp. 3-12).Washington, D.C.: IEEE, Computer Society.

Ramabadran, T., & Gaitonde, S. (1988) A tutorial on CRC computations. *IEEE Micro, 8(4),* 62-75. http://dx.doi.org/10.1109/40.7773

Saxena, N., & McCluskey, E. (1990). Analysis of checksums, extended-precision checksums, and cyclic redundancy checks. *IEEE Transactions on Computers, 39(7),* 969-975. http://dx.doi.org/10.1109/12.55701

Schwiebert, L., & Jiao, C. (2001). Error Masking Probability of 1's Complement Checksums. In *Proceedings of 10th International Conference on Computer Communications and Networks* (pp. 505-510). ICCCN'01.

Sheinwald, D., Satran, J., Thaler, P., & Cavanna, V. (2002). "Internet Protocol Small Computer System Interface (iSCSI) Cyclic Redundancy Check (CRC) Checksum Considerations", IETF RFC 3385.

Simon, H. (1989). *The Science of the Artificial.* Cambridge, Massachusetts: MIT Press.

Sklower, W. (1989). Improving the efficiency of the OSI checksum calculation. *ACMSIGCOMM Computer communication Review, 19(5),* 32-43. http://dx.doi.org/10.1145/74681.74684

Smith, M. (1998). *Application Specified Integrated Circuits* (1st ed.). Addison-Wesley-.. Longman.

Stallings, W. (2000). *Data and Computer Communications*. Upper Saddle River NJ: Prentice
. Hall.

Stallings, W. (2008). Data and Computer Communications (8th ed.). Prentice Hall.

Sun Microsystems (2000). Jini Connection Technology. Retrieved from
http://www.sun.com/jini/

Stone, J., Hughes, J., & Partridge, C. (1995). Performance of checksums and CRCs over real data *ACM SIGCOMM Computer Communication Review*, 25(4), 68-76.

Tennenhouse, D. (2000). Proactive computing. *Communications of the ACM, 43(5),* 43-50.
http://dx.doi.org/10.1145/332833.332837

Technopedia. (n.d.). Error Detection. Retrieved from
http://www.technopedia.com/definition/1796/error-detection

Technology Product Reviews, News, Prices, & Downloads | PCMag.com| PC Magazine.
Retrieved 2 April 2015, from
http://www.pcmagazine.com/encyclopedia/term/40466/crc

Thomsen, D. (1983). XOR Checksum. *Science News.*

Thales Navigation (2002). *Data Transmission Protocol Specification for Magellan Products Version 2.7.*

The Jini Specification. (2000). *Kybernetes, 29(1).*
http://dx.doi.org/10.1108/k.2000.06729aae.003

TTTech Computertechnik AG (2003). *Time Triggered Protocol TTP/C High-Level Specification Document, Protocol Version, 1.1,* specification ed. 1.4.3

The HART Book (2005). *The HART Message Structure. What is HART?* http://www.thehartbook.com/technical.htm

Usas, M. (1978). "Checksum versus Residue Codes for Multiple Error Detection". In *Proceedings of Eighth Annual International Symposium for Fault-Tolerant Computing* (pp. 224). FTCS'78.

Weiser, M. (1991). The Computer for the 21st Century. *SCI Am, 265(3),* 94-104. http://dx.doi.org/10.1038/scientificamerican0991-94

Wheal, M., Miller, M., Stevens, G., & Lin, S. (1986). "The reliability of error detection Schemes in data transmission", *J. of Electrical and Electronics Eng.,* 123-131.

William, R. (1993). "*A Painless Guide to CRC Error Detection Algorithms"* Retrieved March 20, 2015, from http://www.ross.net/crc/crcpaper.html

Wikipedia. Adler-32. Retrieved March 23, 2015 from http://en.wikipedia.org/wiki/Adler-32.

Yu, Q. (2011). *Transient and Permanent Error Management for Networks on Chip*. (Published Doctoral Dissertation). Edmund A. Hajim school of engineering and applied sciences. University of Rochester, Newyork.

Zhao, B., & Olivera, F. (2006). Error reporting in organizations. *Academy of Management Management Review, 31*, 1012-1030.

Zweig, Z., & Partridge, C. (1990). *TCP Alternate Checksum Options.* IETF-RFC 1146.

Appendices

Appendix A: Fletcher Checksum Algorithm Computation

I: Algorithm for 8-bit Fletcher Checksum

Algorithm for 8 bit Fletcher checksum is computed over a pattern of data bytes e.g. C(1) through C(N). This accomplished by preserving 2 unsigned 1's complement 8 bit registers A and B whose elements are zero at the initial stage and executing the subsequent loop where *i* stretches from 1 to N.

$$A = A + C_i$$

$$B = B + A$$

It can be displayed that at the culmination of the loop A will have 8 bit 1's complement summation of entire bytes in the datagram and B will have $(N)C(1) + (N-1)C(2) + \ldots\ldots C(N)$. The bytes shielded by this algorithm should be same as those across which the TCP checksum computation is executed with Pseudo header being C(1) across C(12) and TCP header starting at C(13).

For checksum calculations, the checksum field must be equivalent to zero. At the culmination of the loop A moves into the first byte and B moves into the second byte of TCP checksum. Tcp checksum does not regulate the check bytes so that so that the receiver checksum is zero. This is different from the OSI model of Fletcher checksum.

There are numerous swift algorithms for the purpose of computing the two bytes of the 8 bit Fletcher checksum. Obviously any calculation which calculates the identical, number as computed by the aforementioned loop can be utilized for computing the checksum. One benefit of Fletcher checksum in comparison with the traditional TCP checksum algorithm is the capability to identify the position exchanges of bytes of any length inside a datagram.

II. Algorithm for 16-bit Fletcher Checksum

16 bit Fletcher checksum works exactly in the similar way as 8 bit sum, the only exception is that A, B, and C_i are 16 bit proportions. It is significant (as with typical TCP checksum) to pad a datagram consisting an odd number of bytes with a zero byte. Outcome A

should be put in the TCP header field and outcome B should show up in an TCP substitute checksum data alternate. This alternate should exist in each TCP header. The two bytes secured for B should be adjusted to zero in the course of checksum computation.

TCP header checksum field shall consist of the proportion of A at the culmination of the loop. The TCP substitute checksum data alternate must be available and consist of the proportion of B at loop culmination.

Appendix B: Cyclic Redundancy Check (CRC)

In data communications CRC is a highly effective method of identifying errors. The effectiveness of CRC in identifying errors relies on its length. For any specific set of data an 8 bit CRC can undertake 1 of 256 values of which only one value is the accurate one. If some arbitrary error transpires, the possibility is that only 1 out of 256 that the CRC structure will result in emulating the specific data. Therefore the possibility of its not emulating and identifying the error is the other 255 probable numbers out of 256, which provides:

$$\frac{2^8 - 1}{2^8} = \frac{256 - 1}{256} = \frac{255}{256} = 99.6\%$$

Possibility of identifying errors with a 16 bit CRC is:

$$\frac{2^{16} - 1}{2^{16}} = \frac{65536 - 1}{65536} = \frac{65535}{65536} = 99.99\%$$

Possibility with 32 bit CRC is:

$$\frac{2^{32} - 1}{2^{32}} = \frac{4294967296 - 1}{4294967296} = \frac{4294967295}{4294967296} = 99.9976\%$$

Therefore, a long CRC immensely enhances possibility of identifying an error. If a system produces an arbitrary error every 1/1000 second (a bad rate in contemporary digital systems), 1000 errors per second will be produced. 1 error out of every 256 will be undetected by an 8 bit CRC, hence 4 errors will be undetected in each second. A 16 bit CRC is prone to miss 1 error out of 65536. So the time period between misses would be about 650 seconds or 11 minutes. Finally a 32 bit CRC evades 1 error out of every 4, 294, 967, 296 which means 50 days or 4,294,967,296 seconds between misses.

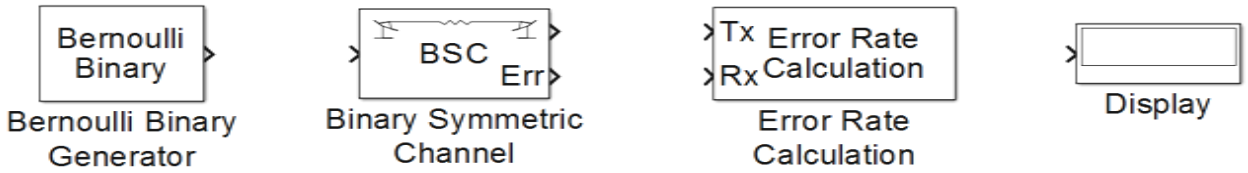
These numbers are rather cautious, for the reason that they presume that all these errors are genuinely arbitrary and serious. In contemporary communication systems far less than 1000 errors are produced and large number of these errors are assured to be identified (like single bit errors or errors including a moderate number of successive bits known as burst error). So even a 16 bit CRC is generally regarded as proficient enough in detecting errors.

Appendix C: Simulink Blocks

Simulink is an addition to Matlab used for the purpose of representation and simulation of systems. In simulink, systems are depicted on screen as block illustrations. Many components of block illustrations are accessible (like summing functions and transfer functions), virtual input devices (like function generators) and output devices (like oscilloscopes). Simulink is assimilated with Matlab and data can be moved between programs with ease.

Unix, Macintosh, and windows environment support simulink and it is incorporated in the student version of Matlab for personal computers and laptops.

I: Simulink Blocks used for building a Noise Channel:

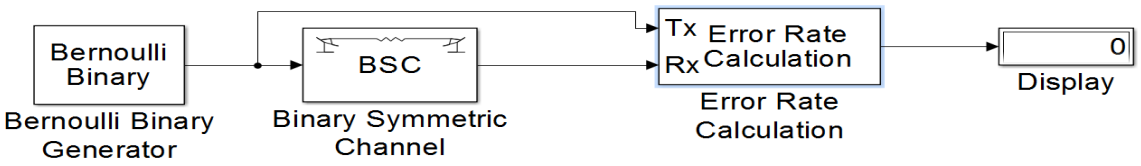


- **Bernoulli Generator:** Bernoulli generator creates an arbitrary binary pattern.

- **A binary symmetric channel:** A binary symmetric channel block simulates a channel with noise. The block initiates arbitrary errors to the signal by altering a 0 to 1 or 1 to 0. With a possibility set out by the error probability parameter of the block.

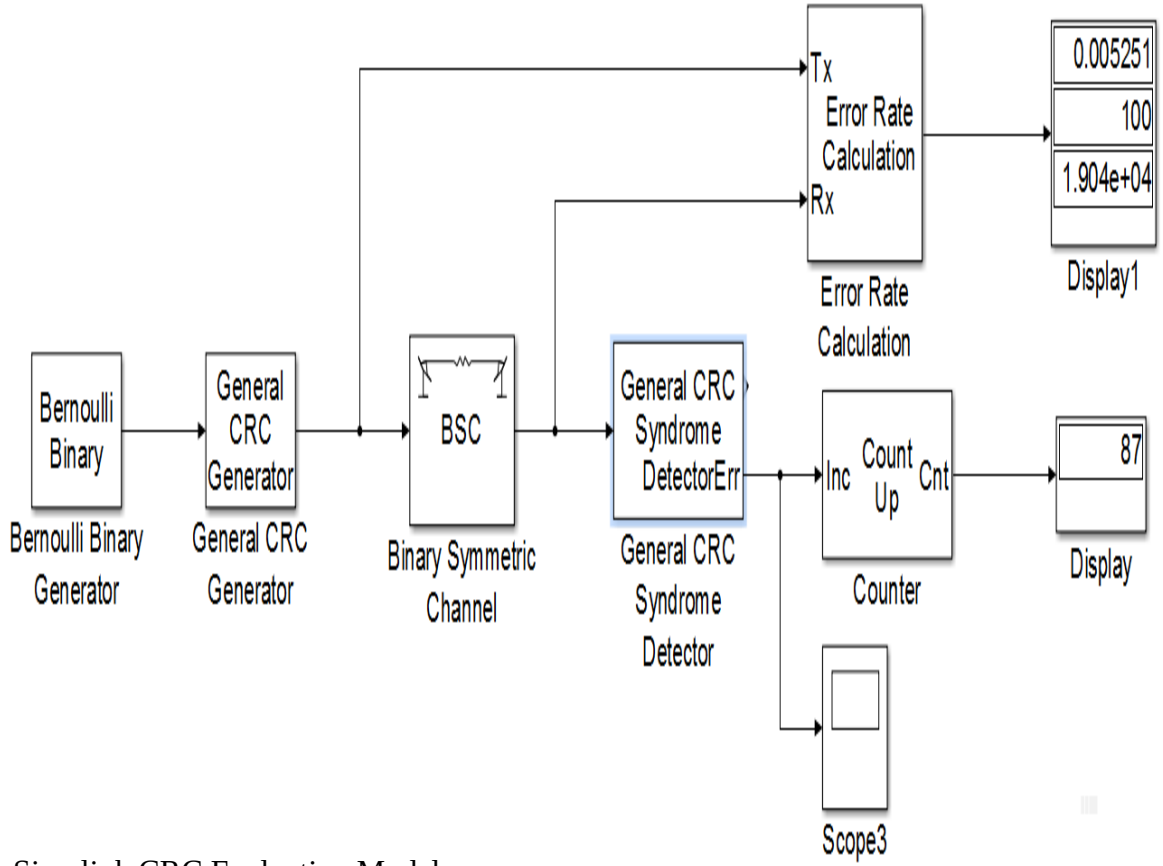
- **Error rate channel block**: The error rate of the channel is calculated by the error rate channel block. There are two input ports of the block. Tx for transferred signal and Rx for received signal.

- **Display block:** Output of the error calculation block is showed by the display block. It displays the bit error rate, number of errors, and total number of bits that are transferred.



Simulink Channel Noise Model.

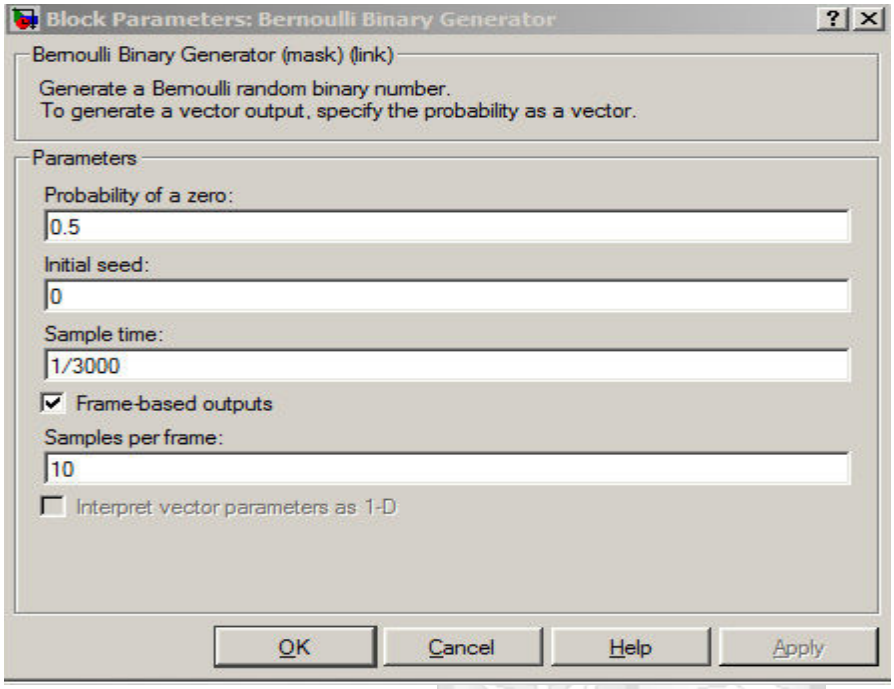
II: CRC Evaluation Model



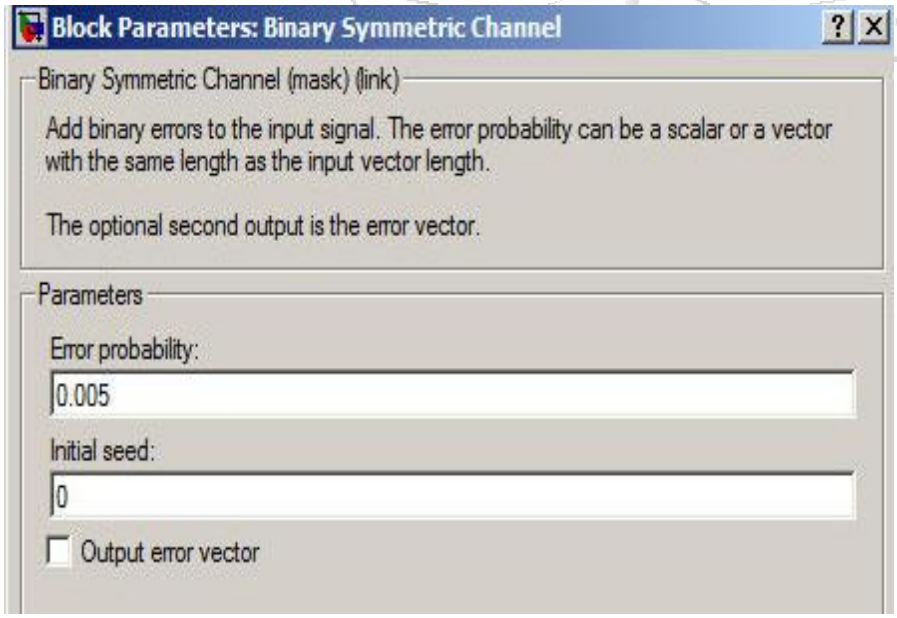
Simulink CRC Evaluation Model.

Appendix D: Configuration Parameters for Simulink Blocks

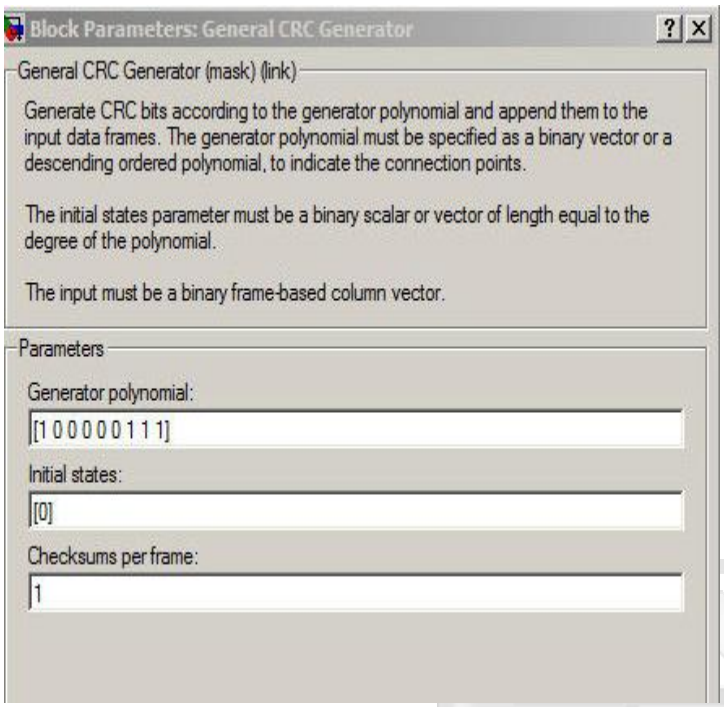
Bernoulli Binary generator Configuration Parameters.



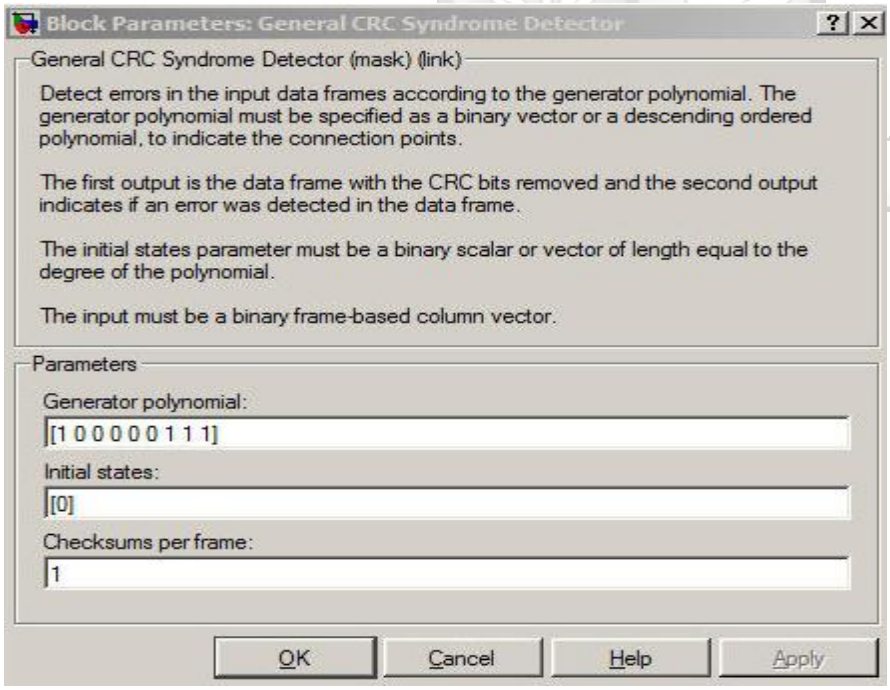
Binary Symmetric Channel Parameters



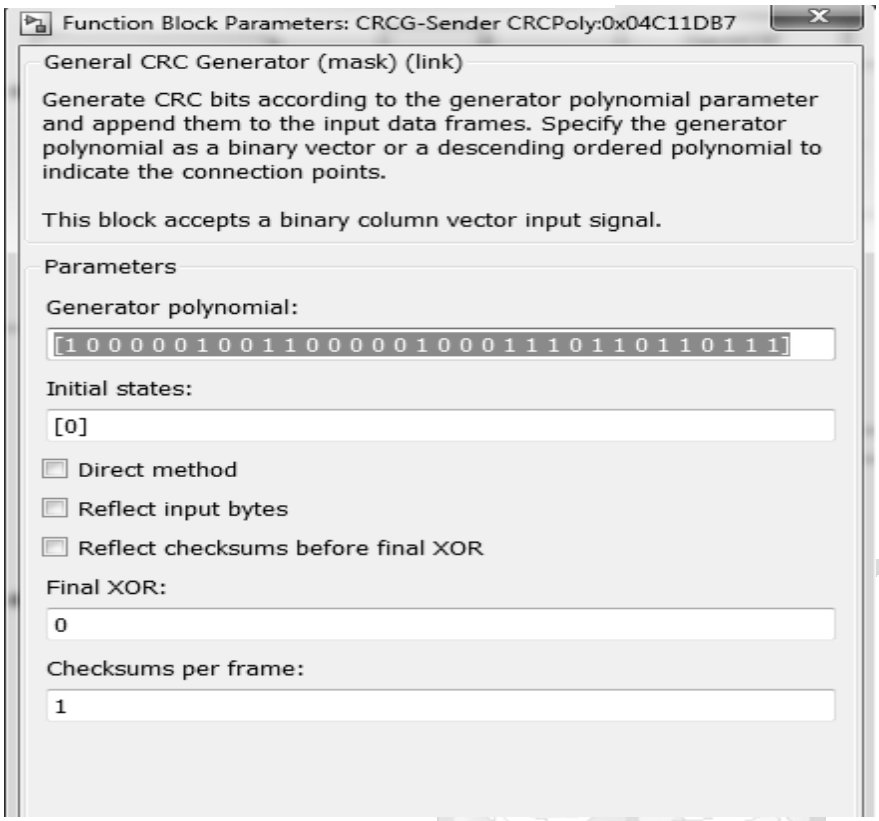
General CRC Generator Parameters



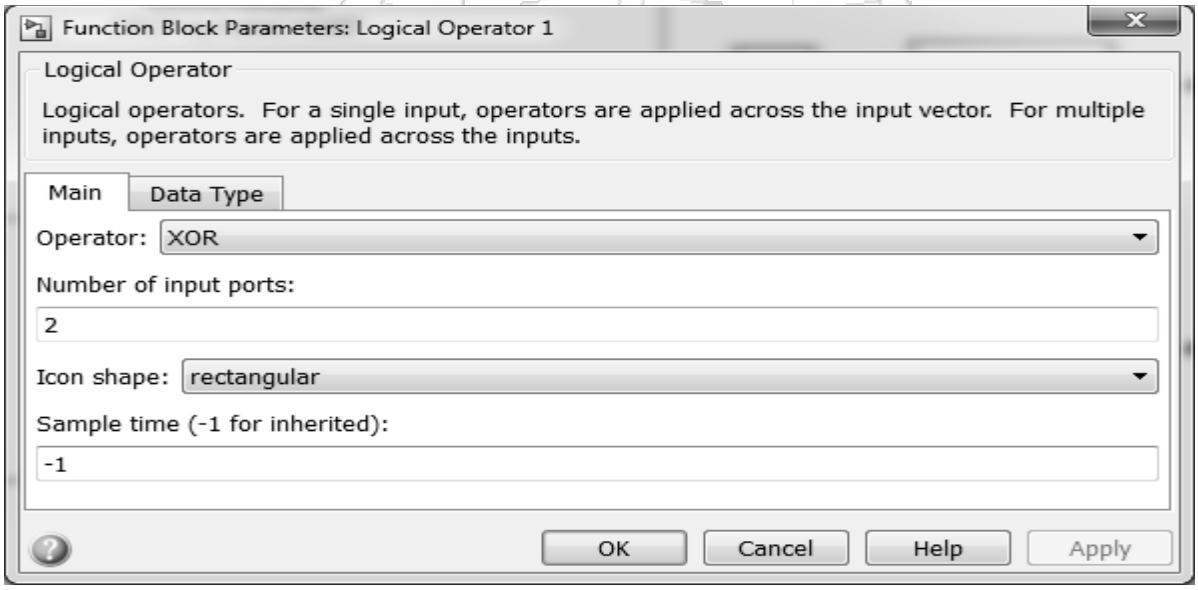
General CRC Syndrome Detector Parameter



CRC Polynomial 0x04C11DB7 Parameters



Logical Operator Parameters



Appendix E: Turnitin Report

Analyzing Error Detection Performance of Checksums in Embedded Networks

ORIGINALITY REPORT

19%	16%	12%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.ece.cmu.edu Internet Source	6%
2	www.docstoc.com Internet Source	1%
3	ip-doc.com Internet Source	1%
4	Theresa Maxino. "The Effectiveness of Checksums for Embedded Control Networks", IEEE Transactions on Dependable and Secure Computing, 2008 Publication	1%
5	Submitted to Strathmore University Student Paper	1%
6	books.huihoo.org Internet Source	1%
7	www.zlib.net Internet Source	<1%

Submitted to University of Hertfordshire

VT OMNES
VNVM SINT

VT OMNES
VNVM SINT

VT OMNES
VNVM SINT