ABSTRACT

Implementation of Lossless Compression Algorithms for the MIL-STD-1553

Bernard O. Lam, M.S.E.C.E

Mentors: Michael W. Thompson, Ph.D.
Russell W. Duren, Ph.D.

This thesis focuses on the bandwidth limitations faced by the legacy MIL-STD-1553 data bus. In order to improve bandwidth performance, lossless implementations of data compression routines have been proposed. Using data bus captures from the F/A-18 C/D simulator it has been possible to determine data characteristics, resulting in statistics showing the inherent redundancies within the data. This thesis proposes three compression algorithms which have been developed for use on the MIL-STD-1553 data bus. The three methods are Common Value Tracking, Modified Run-Length Encoding, and Differential Encoding. It will be shown that in some cases, compression ratios over 10 to 1 are possible, significantly improving the data transfer capabilities of the legacy communication system. The compression algorithms have been designed to provide bounded deterministic operation as required by real-time systems. In this thesis we will explore the strengths and weakness of each of these algorithms and also the decisions and challenges associated with integration with MIL-STD-1553 systems.

Implementation of Lossless Compression Algorithms for the MIL-STD-1553

by

Bernard O. Lam, B.S.E.C.E

A Thesis

Approved by the Department of Electrical and Computer Engineering

_____
Kwang Y. Lee, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree
of
Master of Science in Electrical and Computer Engineering

Approved by the Thesis Committee

_____
Michael W. Thompson, Ph.D., Co-Chairperson

_____
Russell W. Duren, Ph.D., Co-Chairperson

_____
Wickramasinghe Ariyasinghe, Ph.D.

Accepted by the Graduate School
December 2008

_____
J. Larry Lyon, Ph.D., Dean

Page bearing signatures is kept on file in the Graduate School.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First, I would like to begin by thanking Dr. Russell Duren for his guidance and support in finishing this thesis. Additional thanks to Dr. Michael Thompson can also not be overlooked for his invaluable assistance. It has been a privilege to have worked with the two of you these past few years. Thanks to Dr. Ari for his participation on my thesis committee.

I would like to thank my family for the countless years of motivation and support. Thanks to my parent for always placing my education at the forefront, and instilling in me the values of hard work.

Finally, I would like to send out a special thanks to my dear friend April. My experiences at Baylor University have undoubtedly been shaped by knowing you. Thank you for your encouragement over the years.

DEDICATION

To Mom and Dad, “Apwoyo”

CHAPTER ONE

Introduction

The MIL-STD-1553 data bus, with over 30 years in operation, continues to be widely utilized in military avionics systems. The cost trade off of the current standard outweighs any possible inclinations to overhaul the current system to another communication protocol in most applications. The decision to prolong the operation of the 1553 bus requires that reengineering methods be applied in order to extend its functionality. There are numerous options that present themselves when deciding to reengineer a system which include meeting budgetary and functional requirements. In this chapter a brief identification of the challenges associated with legacy systems and several options for reengineering these systems will be addressed.

Legacy Systems

The major dilemma facing the military, in this age of rapidly changing technology, is its aging fleet of aircraft. Astonishingly, the large majority of military aircraft have been in service for 20 or more years. Yet, with limited military budgets it is likely that the current fleet will continue to be utilized in the years to come. The aged electronic components of these aircrafts are sometimes referred to as "legacy systems" [1]. A legacy system is a system composed of obsolete hardware or software components which nevertheless remain in operation, although, they are often at a point at which they should either be redesigned or replaced. The overwhelming problem related to legacy systems stem from five different factors: I/O Compatibility, Processing

speed, addressing limits, memory, and bandwidth limitations. These factors limit the overall computational performance as well as the functional capabilities of the system [2], [3]. In an environment where military technologies experience continual evolutionary modifications spawned by new developments in mission requirements, these factors can lead to several significant problems.

The decision to extend or substitute the operation of a legacy system, such as the 1553 data bus, is contingent on three factors: economics, time, and reliability. The complete replacement of the 1553 data bus, for example, requires a large amount of economic capital. The typical cost to develop and test an avionics system is equivalent to approximately 40 percent of the total cost of the aircraft [3]. Alternatively the typical strategy for maintaining and reengineering a legacy system is to identify the most critical areas which can be cost effectively upgraded. The economic challenges are often the compelling motivation for retaining or replacing of the antiquated systems. If the progressive maintenance costs exceed the acceptable budgetary limitations as compared to the cost of a new system, it is not sensible to continue maintaining the legacy system.

Legacy systems are increasingly more expensive and time consuming to maintain over time. The upgrading process for a legacy system is usually dependent on what mission critical changes must be made first. The system can either be entirely redesigned or incremental changes can be made according to the needs of the designers. The time required to maintain or modify these systems is important because it is time in which they are not in operation. This also suggests that the reliability of the system is an important factor. Conversely, the procurement of a new aircraft can range any where

from 15 to 20 years. Military systems require long development cycles because they are mission critical systems which are complex by nature and are exposed to stringent verification processes. Timing and scheduling is a critical component to understanding what decisions are made in regards to the future operation of a legacy system.

As long as the legacy system meets the basic needs required, there is usually a reluctance to dramatically overhaul the entire system. The decision to retire a legacy system often occurs when the maintenance cost or the inability to accomplish goals outweigh the overall cost to completely replacing the system. Some of the questions which must be addressed when making such considerations are:

- What is the rationale for maintaining the current system?
- Does the current legacy system meet the minimal functional requirements and is it stable?
- Are new technologies and requirements able to be incorporated easily into the system?
- Are the cost and scheduling required to redesign the current system too great?
- What is the cost for maintaining the current system as opposed to redesigning the entire system?

Reengineering Legacy Systems

There are two areas in which a legacy avionics system can potentially be reengineered: hardware and software component improvements [1]. One option is to reengineer and overhaul the hardware of the avionics system. While this option has its advantages, the integration of new components with the existing avionics system can pose numerous problems. Additionally, changes to hardware components often require the additional modification of the Operational Flight Program (OFP). The OFP is

identified as the aircraft program software embedded within the avionics system [3]. Depending on the requirements for the software upgrade, this can add considerable complexity to the reengineering process.

The alternative reengineering option is to modify the software of the system. Legacy hardware systems often impose limits on memory, processing speed and other hardware constraints. Also, the software design practices followed 20 or more years ago are dramatically different from current standards. As a result, the modification of legacy software can present numerous difficulties for programmers. The major drawback to increased modifications to the original OFP is the likelihood that the software might not perform as expected. Continued evolutionary changes to the OFP can weaken the system's operational integrity, requiring that great care be taken to ensure that proper verification processes are met. Despite the negatives, this solution can be advantageous in that the changes to the hardware components can be avoided. Also, the software changes can often be included with ongoing software maintenance activities.

The relative cost of the reengineering can vary upon the combinations of required improvements. This thesis examines the application of the software option for improving the bandwidth capabilities of the MIL-STD-1553 Data Bus. This upgrade strategy is a very cost efficient solution of improving the 1553 data bus. The advantage of this solution is that it requires no changes to the hardware and it does not require that there be dramatic changes to the current OFP. Although data compression is not a comprehensive solution to the challenges facing the 1553 data bus, it offers the potential for significant improvement.

Related Work

This thesis is a continuation of prior research investigating the application of low-complexity data compression schemes to the legacy MIL-STD-1553. This research was undertaken by Bron Weston in [4]. The results of this research yielded three proposed compression algorithms in which one was analyzed for timing feasibility. This thesis improves upon two of the three compression algorithms and compares the timing achievements of the three techniques. Additionally, this continued research explores error handling methods as well as implementation challenges and their possible solutions.

In reference [5], researchers provided a variation to the differential algorithm used in this research, which will be examined in depth in Chapter 5. In this related work, the researchers apply the compression algorithm to executable files for the purpose of updating software. Similar to our research, the compression method is suited for mobile devices with limited computational, storage and bandwidth capabilities. Their approach differs from ours, in that the method they employ the assumption that the compression process occurs offline. This allows for the majority of the encoding complexity to be placed on the compression side. On the other hand our application requires the use of limited computational resources. This limits the complexity of compression algorithms on both the compression and decompression ends.

Thesis Overview

This thesis will discuss the customized data compression algorithms that have been developed for the MIL-STD-1553 data bus. The aim of the data compression schemes is to maximize the effective bandwidth of the 1553 bus by minimizing the time required to transfer data. The thesis shall be constructed as follows. Chapter 1 has been a basic introduction to the problems facing legacy systems. Chapter 2 reviews the design and protocol standard associated with the MIL-STD-1553 data bus. Chapter 3 is an introduction to the related topics associated with data compression. In this chapter, the compression strategy is clearly defined based upon the known system characteristics of the MIL-STD-1553. Chapter 4 presents a statistical analysis of typical 1553 bus traffic. The data for this analysis was obtained from bus traces provided by the F/A-18 Advanced Weapons Lab at China Lake. Chapter 5 discusses the three proposed compression algorithms which have been developed and also addresses the implementation challenges which have been identified. Chapter 6 presents the findings and analysis of the results. Chapter 7 presents the conclusions.

CHAPTER TWO

MIL-STD-1553

The design of the MIL-STD-1553 bus was largely driven by the necessity for reduction of wiring and the associated weight, resistance to EMI (electro magnetic interference), and a standard interface. Developed in late 1960's and early 1970's, for the primary use of the F-16, the MIL-STD-1553 bus standard has been a widely utilized within the military, and numerous other safety and mission critical systems [7], [8], [9]. The standard established electrical and protocol standards for use on a common bus architecture. The 1553 architecture was an improvement upon previous generations of avionic systems which were composed of separate and dedicated digital and analog systems. The preceding standalone system, based on point-to-point wire configurations, was an inadequate method for interconnecting complex avionics systems with an increasing number of subsystems [7], [8]. Another disadvantage of the point to point topology, shown in Figure 1, is the excess weight added to the aircraft.

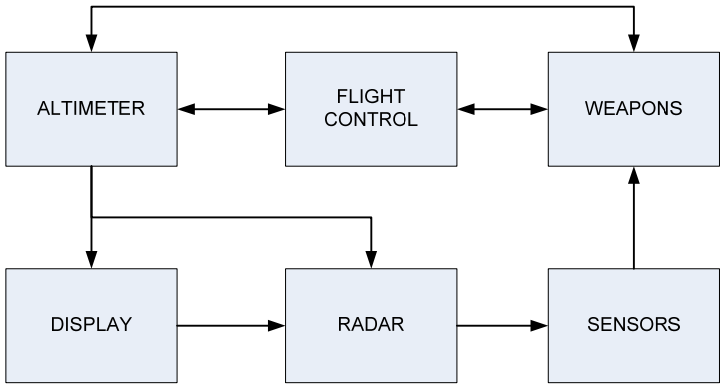

Figure 1: Point to Point System Configuration

During the 1960's and 1970's, with increasing technological advances, there was a progression toward incorporating digital technology into avionics systems [6], [7], [8]. Digital technology and microprocessors promoted the development of smaller subsystems, dramatically reducing the weight load of the system. The digital MIL-STD-1553 topology, illustrated in Figure 2, concentrated upon a structure of various subsystems interconnected by way of a common data bus. The bus architecture was distinguished for its ability to allow communication between unlike aircraft subsystems. Communication between the various subsystem components is achieved by maintaining uniform I/O ports, and bus protocols. Initially designed for military avionics, the communication standard has also been adopted for applications ranging from spacecrafts to use in the commercial sector [6].

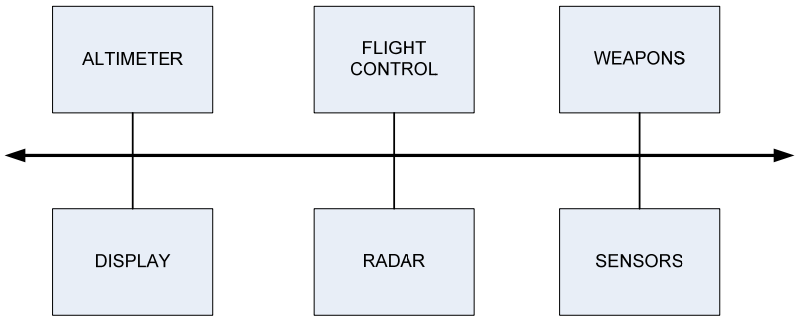

Figure 2: Common/Shared Bus Configuration

In this chapter the discussion will be focused upon introducing the characteristics, and functionality of the MIL-STD-1553 data bus. The configuration and protocol of the 1553 bus is essential to understanding the requirements and constraints of the reengineering design decisions.

1553 Characteristics

The 1553 standard is based upon a serial command/response bus protocol with a 16 bit instruction set. It operates at a data rate of 1 MHz, which is substantially slower in comparison to current standards, and will prove insufficient for future demands. Newer network technologies such as Fibre Channel and Ethernet with gigabit transmission rates are currently in development. The protocol additionally supports an asynchronous time division multiplexed (TDM) arbitration scheme, which allows for multiple simultaneous transactions to occur while using the same communication medium without conflict [7]. This means that each packet of information is allotted a specific time in which it must be completed.

The system architecture of the1553 data bus is divided into the following four elements: Bus Controller (BC), Remote Terminal (RT), Bus Monitor (BM), and Transmission Medium (Data Bus) [6]. Figure 3 shows an illustration of the multiplexed MIL-STD-1553 dual redundant bus architecture. The system is configured with two groups of redundant system components: the bus controllers, and the data buses. This design configuration was developed largely to ensure continued operation despite possible system failures. In the case that one data bus or mission computer is to become inoperable, the 1553 bus is designed to substitute to the undamaged secondary component avoiding a catastrophe [10]. The distinguished reliability record of the 1553 bus has made it attractive for implementation in numerous types of mission critical systems.

The Bus Controller, often referred to as the AYK-14 Tactical Mission Computer (MC), is responsible for regulating any activity on the bus. The BC initiates and

coordinates tasks by issued commands to the remote terminals indicating when data can be either transferred or received. The controller also monitors the current status or failure states of the RTs [6], [7]. Although, two mission computers are present each independently has its own responsibilities. MC1 is responsible for meeting all the navigation related functions for the F-18. MC2 is dedicated to the processing of weapons delivery functions [11]. The two mission computers share information between each other, which is facilitated by a BUS # 3. If one MC were to stop operation, the secondary unit assumes control of the system with limited functionality.

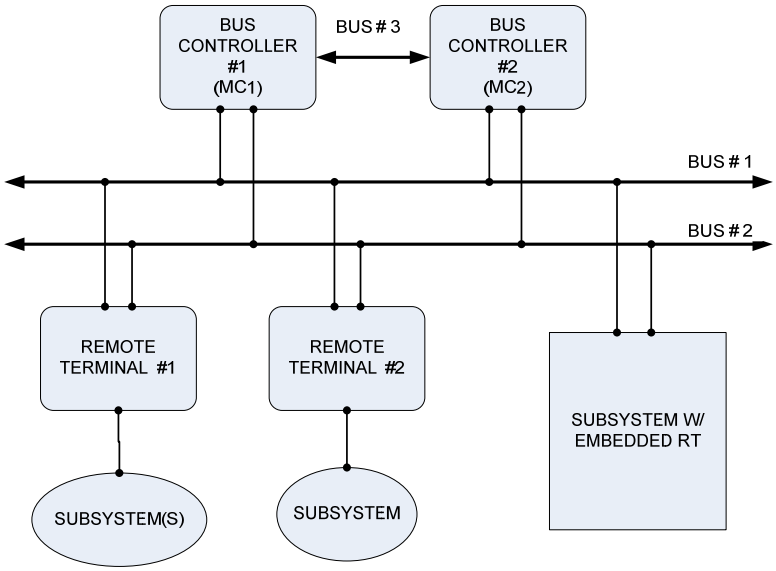

Figure 3 : MIL-STD-1553 Architecture

Each Remote Terminal functions as an intermediate interface between the data bus and the corresponding subsystems [8]. The RT's vary from dedicated navigational units, to radar and weapons units. It is possible for the RT interface to receive and transmit messages from either the BC or other RT's. Whenever a command is issued by the BC to the RT, the RT is required to respond within fixed time constraints in order to

be deemed a valid transmission. Upon the request of the BC, the RT is able to transmit messages to other RT's or to the Bus Controller [4], [7]. The 1553 architecture is restricted to a maximum of 31 total remote terminals. As indicated in the diagram, there are also combined units of subsystems embedded with RT's.

Bus Monitors strictly receive data and do not transmit messages to any other units. Its intended function is to mine for selected bus information which can subsequently be utilized for analysis. The monitor can either be designated as a recorder or a back-up unit [8]. When it is allocated an address location on the bus it is able to function as a Remote Terminal [7].

Protocol

The 1553 protocol transmits messages using a serial Manchester II Bi-phase encoding scheme. Although the encoding scheme consumes double the bandwidth of the original signal, there are numerous advantages including a self-clocking component and reduced susceptibility to transmission errors. All communications and message formats of the MIL-STD-1553 consist of three general word types: Command Words, Data Words, and Status Words. These three word types are shown in Figure 4.

When decomposed all the word formats consist of three general fields: the synchronization field, the information field, and the parity bit. The first three-bit times are associated with the synchronization field. After the synchronization field, the following sixteen bits comprise of the information field. The content contained in the information field differs for each word type and shall be detailed in subsequent paragraphs of this section. The final bit time is reserved for an odd parity bit [7], [8].

The total length of the three fields totals 20 bit times with each transmission bit spanning the duration of one megabit per second.

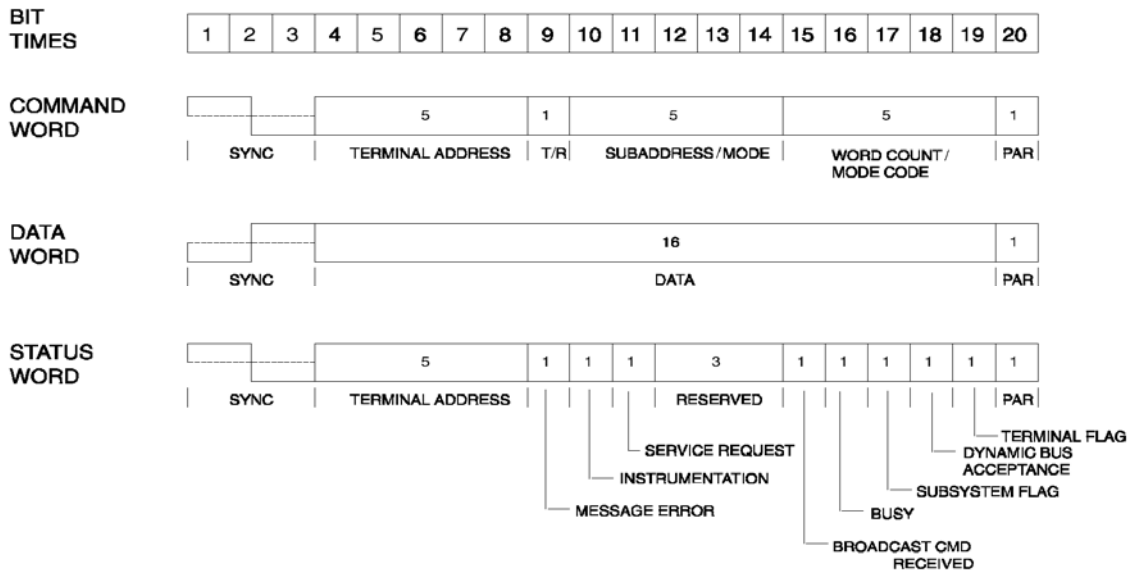

Figure 4 : 1553 Word Format [7]

The SYNC Waveform does not conform to the Manchester waveform encoding standard. Instead the width of the waveform spans three bit times. There are two synchronization patterns associated with the three word types. The status and command word types share the same sync pattern. For these two word types, the first half of the pattern begins with logic high, and then transitions to a logic low. On the other hand, the data word pattern starts at logic low and then transitions to a logic high [8].

The information field of the three word formats varies according to their differing functions. The field of the Command Word contains the parameters required for the remote terminal to execute necessary operations. This word type is exclusively transmitted by the Bus Controller to the Remote Terminals. Included in the information field are the following four sections: Terminal Address, Transmit/Receive Bit,

Subaddress/Mode, and Word Count/Mode Code. The five bit terminal code identifies the remote terminal request by the Bus Controller. Each remote terminal is allocated a unique address. The Transmit/Receive (T/R) Bit functions to designate the direction of the data flow. A logic high sets the T/R bit into the transmit position; while, the logic low sets it to receive mode. The five bit Subaddress/Mode provides the necessary request for the remote terminal subaddress and in addition can be used for the mode control. Finally, the five bit Word Count/Mode Code identifies the size of the data transmission. The field can additionally be used for Mode Code Commands [8], [12]. The Mode Code performs a variety of tasks such as status requests, data management, and numerous other operational controls.

The Data Word contains all the information sent to the receiver for analysis and processing. The entire 16 bit information field of the data word is allocated for data. Data can be transmitted from the Remote Terminal or Bus Controller. The data word utilizes a big endian standard in which the MSB is the first bit transmitted [8], [12]. The 1553 bus allows a maximum transmission of 32 data words per message transfer.

Finally the information field of the Status Word contains status verifications about the state of the RT, and the transmission process. The information field structure consists of three general segments: Terminal Address and the Status Field. The five bit terminal address identifies the transmitting RT, followed by the 11 bit Status Field. Contained in the Status Field are the notifications such as message error bit (bit time 9) and transmission verifications [8], [12]. The Status Word is transmitted by the RT to the BC notifying it of status conditions.

Message Formats

All communication transactions, on the 1553 data bus, are categorized into ten different message types. The composition of each message is derived from the three word types discussed in the previous section. These message types are placed into two groups: Information Transfer Formats, and Broadcast Information Transfer Formats [7]. Information Transfer Formats follow the normal command/response communication protocol, in which transmitted messages are succeeded by a reply. An example of this is a command message sent from the BC and received by the RT, followed by a transmission of a status word from the RT to BC. Broadcast transmissions are commands which are transmitted to several Remote Terminals concurrently. Although the command message is sent at the same time no response is made. Instead, the BC is able to receive the messages by polling each RT individually.

Despite the ten different messages there are in truth only three basic message transmissions. The additional message types are based upon mode command with any data word transfers. The three basic message transfers are: BC to RT, RT to BC, and RT to RT. These three message types are shown Figure 5. Considered the receive command the message transfer from the Bus Controller to the Remote Terminal (BC-RT) is the most commonly exchanged transfer. The message sequence is initiated by a command word from the BC followed by the required number of data words. As previously mentioned, 32 data words is the maximum amount that can be transferred with each transmission. The specified word count is contained in the information field of the Command Word. After all data words have been received, the RT responds by sending a Status Word to the BC [12].

The message transfer from the Remote Terminal to the Bus Controller is referred to a transmit command. In this sequence, a command is sent to the RT requesting a transfer of data. The T/R field, in command word, indicates a request for transfer of data by the RT when it is logic one [7], [12]. The RT then transmits a response status word followed by the requested number of data words. The number of data words is contained within the original command word.

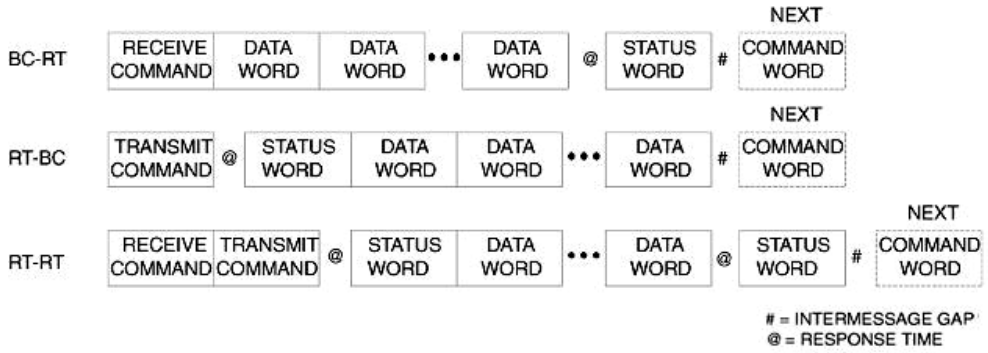

Figure 5 : 1553 Message Format [7]

Remote terminal to remote terminal (RT-RT) transfers allow for messages to be transmitted between RTs without utilizing the BC as a buffer. The exchange sequence is initiated by the BC which sends two consecutive commands. The first is a receive command sent to the receiving TR. Within the command, the T/R bit is set to logic 0 and the number of data words being sent is indicated. The second command word is a receive command which is sent to the data transmitting RT. Following the command sequences, the transmitting RT responds by sending a status word followed by the data words [7], [12]. Once the receiving RT obtains all the transmitted data words, it then transmits its own status word.

Rate Groups

As previously mentioned, the 1553 bus communication is based upon time division multiplexing. The TDM communications are partitioned into four separate periodic rated groups: 20 Hz, 10 Hz, 5 Hz, and 1 Hz. The scheduling of tasks is structured prior to each runtime, and is dependent on what tasks must be completed. Each time frame, known as a minor cycle, is initiated by a 50 millisecond hardware interrupt at which time the specific task must be completed. A rescheduling sequence of tasks occurs after every major cycle, which is equal to 20 frames. It should be noted that no more than two tasks or rate groups are executed concurrently per frame [11].

Error Detection and Handling

Bus errors can stem from various sources such as electromagnetic interference (EMI), hardware failure, and other random noise. Depending upon the system these error incidents can greatly impede performance and even lead to system failures. One of the most important characteristics of the 1553 bus is its ability to detect and mitigate such transmission failures. The failure rate for the MIL-STD-1553 is relatively low, making it suited for mission critical systems [13].

The basic built-in validity checks are essential to the robustness of the system. All data exchanges must maintain continuity checks in order to validate the transmission. Validity checks are comprised of the integrity certification of four different factors: the Sync Waveform, the Manchester Encoding, the Parity, and the total word count [12].

The first two factors, the Sync Waveform and Manchester Encoding, fall under the category of pattern fault detection. Under this classification the errors are

recognized by discerning integrity and pattern flaws in the message transmission. These communication protocols provide the foundation for the error detection within the system.

As indicated in the protocol section, each word type incorporates a parity bit field. Although, parity checks adds a degree of protection against errors it is not perfect. While the 1553 architecture is able to detect single bit error, double bit and other even multi-bit errors are undetected. The 1553 architecture does not possess a cyclic redundancy check (CRC), but it is normal for them to be incorporated within the specific subsystems.

All exception and invalid commands are issued by the remote terminal. Under the condition of a detected error, the handshaking message sequence is not completed. After sending a message to the RT, the BC waits for approximately 14 microseconds to receive a status word reply. If the transfer is not received, the transmission is determined a failure [14].

The objective of the failure detection and tolerance of the MIL-STD-1553 is to intercept failures from propagating through out the system. It is critical that system operation failures are minimized during the encounter of an error. A system failure is an unacceptable outcome when dealing with a mission critical system.

CHAPTER THREE

Data Compression Theory

Data compression is the process of reducing the required storage or transmission time of an information source. The compression results are achieved by taking advantage of the redundancies contained in the data. Therefore, much of the data compression that will be applied to the 1553 bus is data dependent. Furthermore, the efficiency of the data compression technique is dependent on the data being encoded. In the case of the 1553, the compressed data is transmitted across a communication bus and then decompressed. If the compression process is computationally efficient, it is possible to reduce the overall transmission time.

The rationale for utilizing data compression algorithms is due in part for the need to maximize the effective bandwidth of the 1553 data bus. The increase in subcomponents has increased the congestion on the 1553 bus architecture. Although, there are other solutions to addressing this problem, data compression provides an effective method which does not require dramatic changes to the system. This technique will only require minor changes in the software while circumventing any modifications to the hardware. A notable advantage of employing data compression techniques is the fact that the reengineering process in very cost efficient.

In chapter 1, the various options for improving the performance of the 1553 bus were discussed. In this chapter and those that follow, we will direct the attention towards the software side of reengineering the 1553 bus utilizing data compression techniques. To be clear, it should be mentioned that there will be various references

made in regards to data lengths in the upcoming sections. There are two different terms that will be utilized to identify data length. The 1553 bus can complete message transmissions as large as 960 16-bit data words sent during one transaction. When describing a complete data transmission, it shall be denoted as a "set". However, a section of a complete message transmission will be referred to as a "segment". Segments of data are limited to a maximum of 32 16-bit data words, which arises from the transfer limitations for a sub-address at a given remote terminal.

Impact of Data Compression

The implementation of compression algorithms can dramatically reduce bandwidth demands on a given system. Increased compression allows for more efficient use of the communication bus. In essence, it enables an increase in the effective capacity of the transmission medium which is equivalent to an increased data rate. The time conserved, by the compression process can then be allotted to alternative tasks. The usefulness of the compression technique is dependent on the data being encoded. Data sets composed of higher degrees of correlation are compressed more efficiently; while non-repetitive data sets achieve lower degrees of compression.

A possible disadvantage of compression is the added computational complexity to the data. The complexity of the encoded source imposes additional time requirements on the system. There is a direct correlation between supplementary time requirements and the encoding complexity. In particular, the time required to encode and decode information is greatly impacted by the implementation complexity. In the case of the 1553 bus, the proposed compression algorithms are very basic so that the computational requirements for implementing the schemes are low.

Finally, another problem that must be addressed is the possible affect of errors related to the compression techniques. The likelihood of increased error propagation within encoded data is an immense concern. Compressed data is more sensitive to transmission errors as compared to uncompressed. Therefore a solution must be determined to minimize and detect transmission errors. The challenge related to implementing a data compression scheme is taking advantage of the positives while limiting the negatives.

Terminology

Data compression techniques can be divided into two classes: lossy and lossless compression. As suggested by its name, lossless compression maintains the integrity of the source data. During decompression the original data can be reconstructed without any loss of information. Lossless methods are typically utilized for text related applications. Lossless methods have also been applied in applications such as WinZip, and GIFs.

In contrast, lossy compression algorithms are unable to be perfectly reconstructed to their original form. The implementation of this method offers the advantage of achieving more efficient compression ratios. Lossy methods are often implemented in applications such as audio and image compression. The reduction in quality is usually not noticeable to ones visual and audio senses, and therefore the perceived quality remains the same. Lossy methods are not practical in some cases. For the purposes of the MIL-STD-1553 data bus, lossless compression methods are essential for maintaining the integrity of the information on the transmission bus.

Compression Ratio

The metric developed to evaluate the efficiency and performance of compression methods is known as the compression ratio. The compression ratio is a function of the source information to that of the compressed data size. The following equation defines the compression ratio as:

$$C_{ratio} = \frac{x}{\hat{x}} \qquad (1)$$

Where $\hat{x}$ denotes the compressed information, and x represents the original source information. The compression data $\hat{x}$ is derived from the encoding of the source data. For example, imagine that the initial size of the source data is ten words and a compression sequence is applied which reduces the size to four words. The compression ratio is calculated through the division of the length of the source data by that of the encoded data. The compression ratio for this example is equal to 2.5.

Resulting compression ratios greater than 1 indicate that there has been a reduction in the original size of the data. Conversely, a ratio less than 1 denotes that the compression process has not been effective. This means that the compressed data is larger than that of the original source; an indication that there has been an expansion as opposed to decompression. Consequently, it is desirable to attain higher compression ratios which denote improved performance.

Timing Analysis Concept

The objective of employing data compression techniques on the 1553 bus is to reduce the congestion on the transmission line. Through the reduction of the

information size, transmission speeds can increase, maximizing the static bandwidth resources. Under the assumption that the data is compressed efficiently and the complexity required to encode/decode the data is minimal; the expectation is that there will be a saving in overall data transmission time. Figure 6 is an example comparing the likely timing diagrams for uncompressed and compressed data.

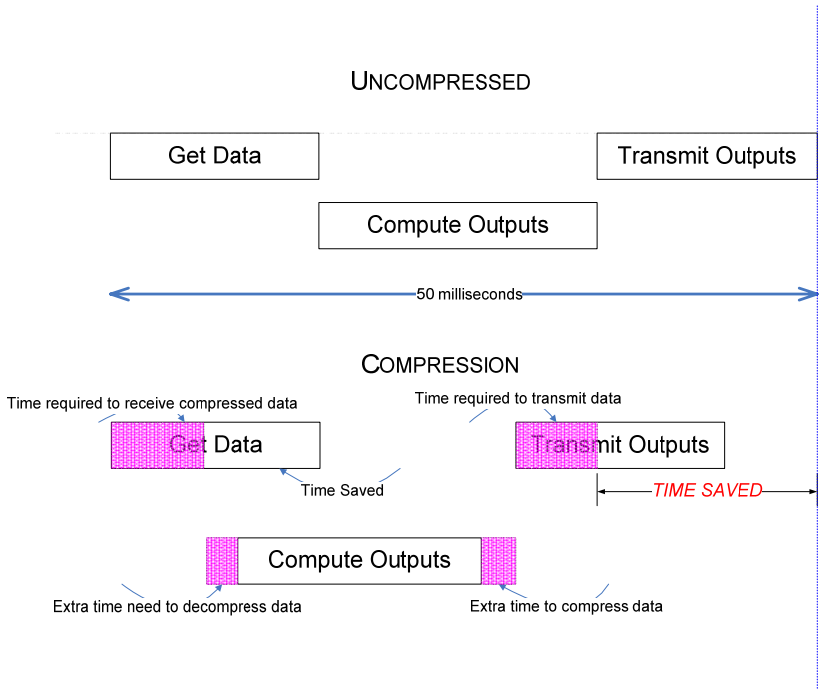

Figure 6 : Successful Timing Analysis (Conceptual)

The uncompressed scenario, in Figure 6, depicts a typical data transmission and computation execution sequence. Each block segment indicates the percentage of time required to execute the specific task. In the case of the uncompressed data, the execution of the three tasks elapses over the complete 50 millisecond time frame. Alternatively, under the compressed scenario an optimum outcome is attained in which transmission time is conserved. A closer look at the compression scenario example

reveals that although there are time savings in the transmission process, added time is required to decompress and compress the data. No conservation of time occurs during the computation process, the time to compute outputs will remain static. As long as the overall decoding/encoding process is minimized as well as there being an efficient use of transmission time, it is possible to achieve a saving in time.

Although, the incorporation of a compression algorithm will require added computational overhead in regards to encoding/decoding, the possible advantage of conserving overall processor execution time makes this valuable. It should be noted (as depicted in Figure 7), under the condition that the extra time required to decompress/compress data is greater than the time savings attained during transmission, it is possible to exceed the timing limitations. Such conditions must be avoided because it can adversely affect the functionality of MIL-STD-1553.

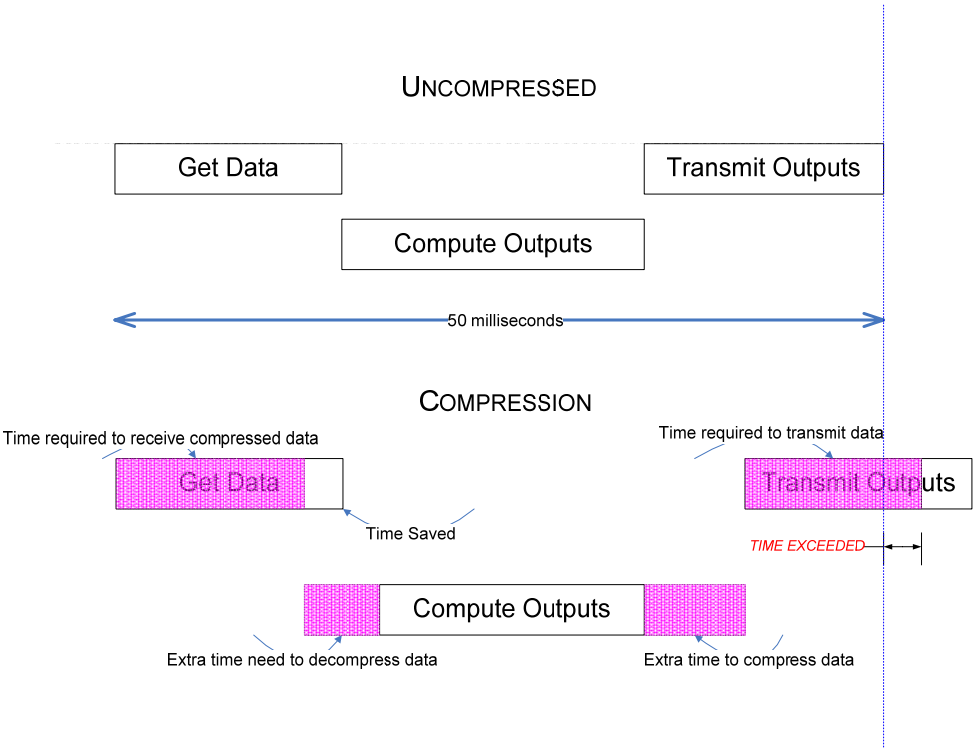

Figure 7 : Unsuccessful Timing Analysis (Conceptual)

Design Goals and Requirements

After an evaluation of the MIL-STD-1553 system characteristics, a compression criterion was developed for the reengineering process. The MIL-STD-1553 bus is a mission critical real-time system, which requires great care be taken when managing its information. Any proposed compression techniques must be well suited for implementation on the 1553 data bus. The five criteria which were developed are:

1. Given the 1553 bus is a mission critical system, it is essential that the performed compression method be lossless. The bus must manage critical flight conditions and controls, therefore the integrity of all source information must be maintained.
2. Limiting the worst case expansion of the algorithm is important to the effectiveness of the scheme.
3. The resources are limited on the legacy system, therefore computational and memory requirement must be utilized shrewdly.
4. The 1553 protocol has strict timing standards; therefore all timing requirements must be preserved. Compression techniques must not extend past timing boundaries.
5. The methods must be compatible with the message format of the MIL-STD-1553 architecture.

CHAPTER FOUR

Bus Transfer Statistical Analysis

Bus Trace

In order to adapt a compression algorithm for the MIL-STD-1553 data bus, it is important to obtain accurate bus statistics of data transfer compositions. Through statistical characterizations of the bus transfers, it is possible to predict and determine optimal compression techniques. Actual and simulated flight bus traces were provided by the F/A-18 Advanced Weapons Lab at China Lake. A Naval F/A-18 Flight Simulator, located at China Lake facilities, was utilized to gather virtual simulations of bus transactions. The actual flight data was gathered by recorded flight logs from actual mission and combat conditions. The information from the actual recorded flight logs is classified, which required that secure methods be taken during the characterization of the data. Therefore all analysis of the actual flight data was performed manually.

A total of twelve simulation bus traces were acquired by Baylor University for research purposes. The simulation bus traces were collected in a text file format. The test files are composed of two basic sections: the configuration section and the data transfer information. The data transfer section contains the monitored simulation data transfers from the F/A-18 bus transactions. Figure 8 shows an example of the data transfer section. Each transmission sequence begins with a series of descriptors such as the mode code, the word count, the remote terminal identification, and the message sequence. The transmission descriptors are followed by the data words, which as noted in earlier sections has a total maximum number 32-16 bit words.

	SEQUENCE NUMBER	RT	MUX CHANNEL	T/R	MSG. #	WORD COUNT/ MODE CODE	TIME TAG	STATUS WORD	DATA
*	0	MC1	AV3 X	T	0	32	10:11:00.20257	A000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

Figure 8: Example of Bus Trace Data

Each bus trace contains approximately 30 seconds of flight data, captured from bus transfers between MC1 and MC2 [15]. The bus traces are reflective of various combat and flight situations, which allows for a representative range of possible conditions encountered by the F/A-18. This allows for testing of both low and high redundancy bus transactions. Although the best performance is achieved during the compression of high redundancy data, a critical point in the analysis is how well the algorithms perform during periods of low redundancy. Hypothetically, the periods of low redundancy transmissions are encountered during combat and high activity conditions.

MATLAB routines were written to parse the raw trace text files, allowing for a convenient method of acquiring and processing the large collections of data. The MATLAB routines are capable of reassembling the data from specific rate groups and message sequences. For example the 20 Hz rate group consists of the consecutive message numbers 6 thru 18. The program searches for all transmissions containing the message number sequences corresponding to the 20 Hz rate group, and extracts the data words.

F/A-18 Bus Statistics

A statistical characterization of the bus data helps to identify the limitations and advantages that can be exploited by the compression routine. Given that the compression performance is subject to the composition of the source data, the characterization of the data leads to a better understanding of the results. The compression algorithm is then adapted from the prior knowledge of the statistics allowing for an optimal algorithm to be developed. In this section, three statistical models are explored relating to the correlation of data within each data set, and also those between successive data sets.

In the first model, an assessment of the statistical results revealed that there was a significant amount of zeros within data messages. Table 1 presents the results of the percent of zeros for each rate group obtained from the simulation bus trace data. The most significant fraction of zeros, on average, was contained in the 10 Hz rate group, at 75.7(MC1) and 84.24(MC2) percent; the 5 Hz rate group then followed with approximately 78(MC1) and 61.50(MC2) percent. The analysis revealed that the lowest percentage of zeros could be found in the 20 Hz rate group. These statistical results indicate that a compression algorithm which reduces the redundancy of zeros should achieve reasonable compression efficiencies. Although, the 20 Hz rate group performance will not match that of the other groups, it should also be reasonably efficient.

Table 1 : Percent of Zeros within Flight Data

Statistical Method	20 Hz		10 Hz		5 Hz	
	MC1	MC2	MC1	MC2	MC1	MC2
Max Zeros (%)	50.99	74.64	78.01	93.14	79.05	88.57
Min Zeros (%)	41.78	59.18	74.08	78.57	74.29	57.14
Avg. Zeros (%)	43.96	65.76	75.70	84.24	78.00	61.50

In previous research, conducted in [4], it was demonstrated that the F/A-18 message transfers contained numerous consecutive repeated data values. A consecutive repetition of a singular value is referred to as a "run". This research certified the large frequency of runs present in the bus trace data. Figures 10 thru 21 illustrates the distribution of runs found within the bus trace data. The graphs below compare the size of a given run with the number of occurrences found within the data. Additional tables corresponding to the graph data can be found in Appendix B.

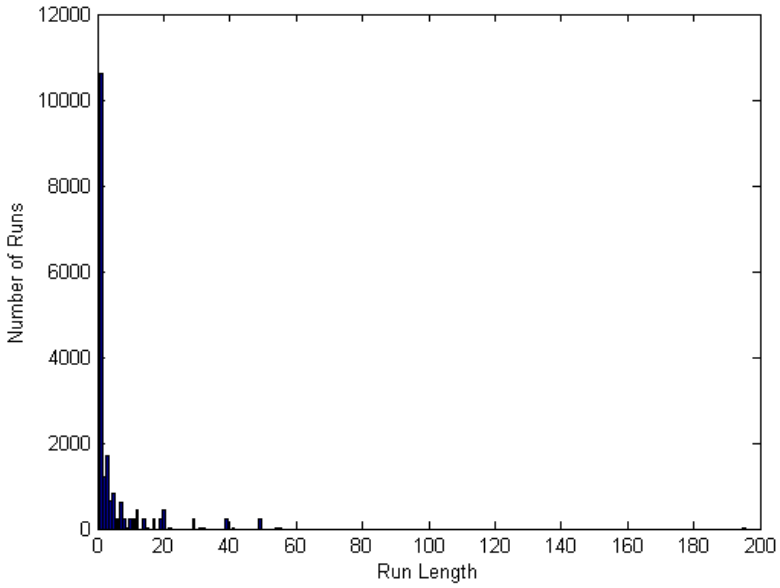

Figure 9: 20 Hz MC1 Dump 1 Run-length Statistics

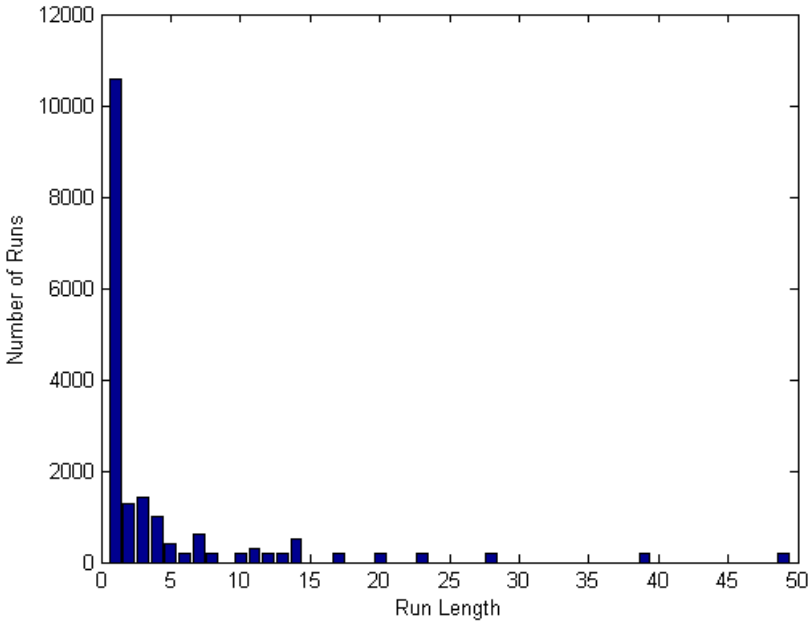

Figure 10: 20 Hz MC1 Dump 2 Run-length Statistics

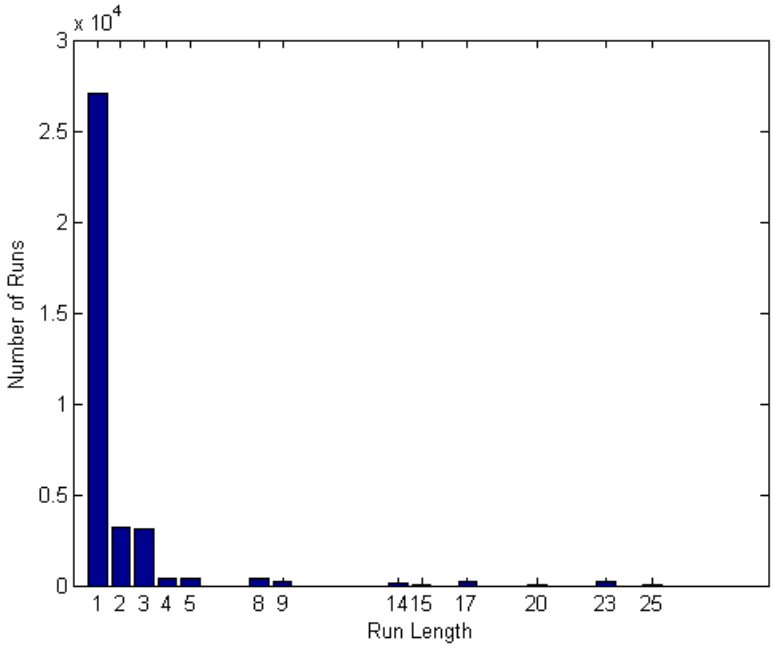

Figure 11: 20 Hz MC2 Dump 1 Run-length Statistics

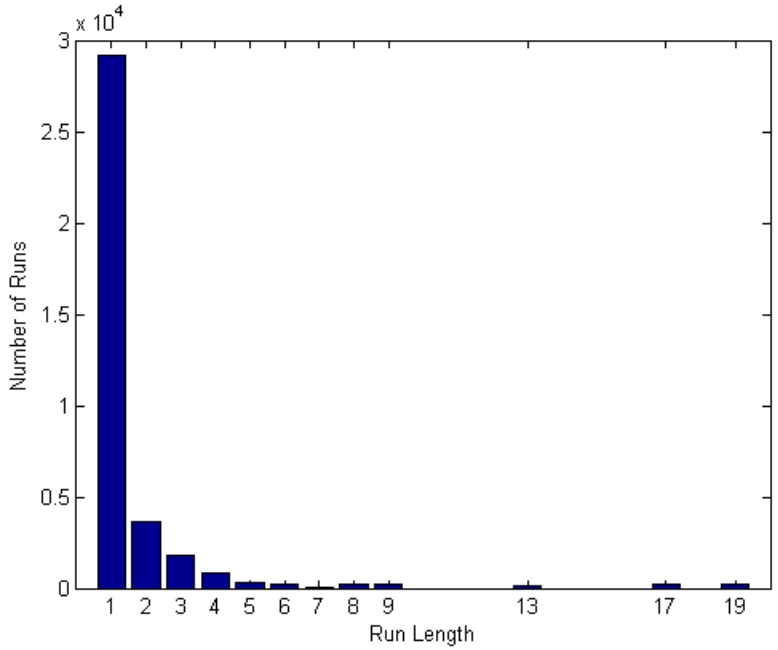

Figure 12: 20 Hz MC2 Dump 2 Run-length Statistics

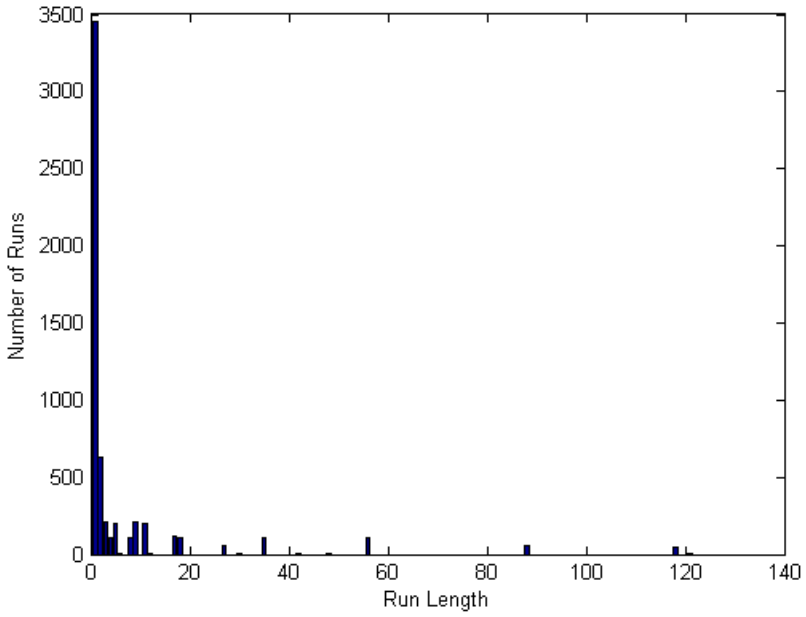

Figure 13: 10 Hz MC1 Dump 1 Run-length Statistics

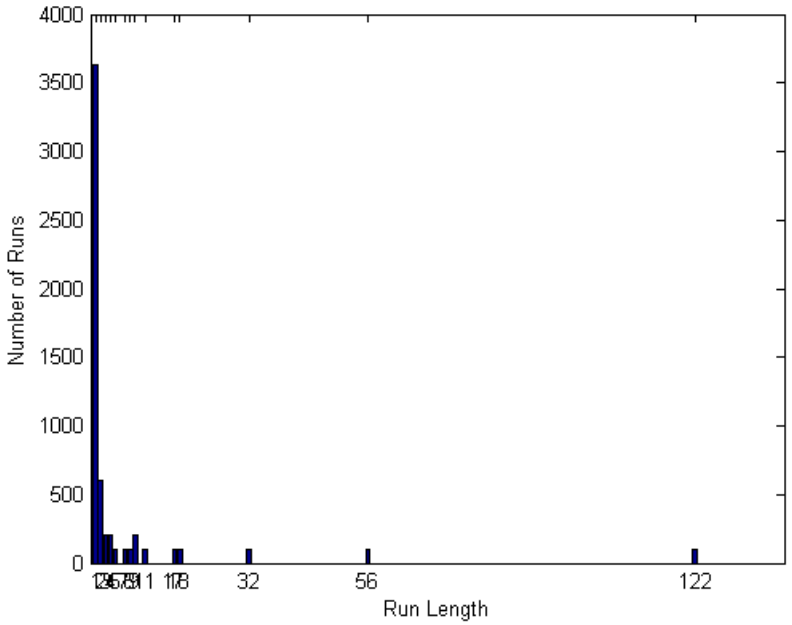

Figure 14: 10 Hz MC1 Dump 2 Run-length Statistics

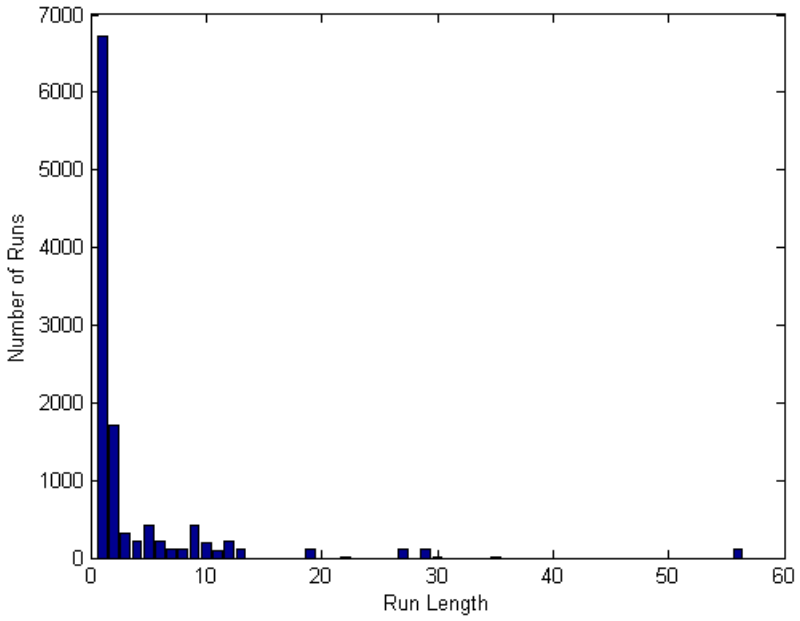

Figure 15: 10 Hz MC2 Dump 1 Run-length Statistics

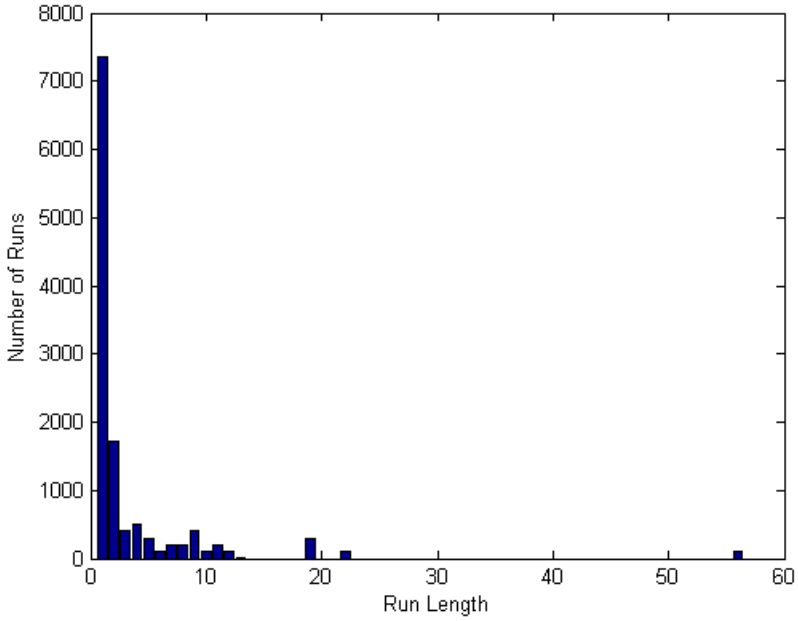

Figure 16: 10 Hz MC2 Dump 2 Run-length Statistics

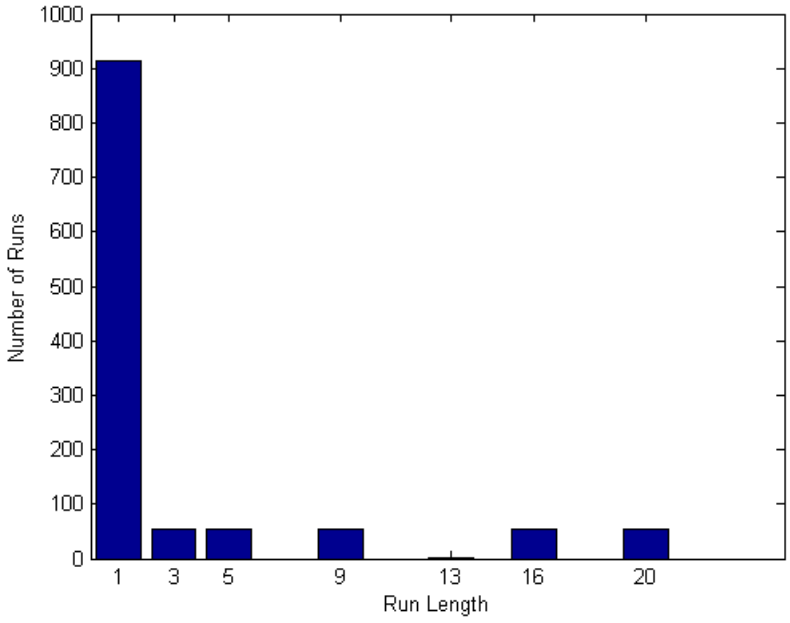

Figure 17: 5 Hz MC1 Dump 1 Run-length Statistics

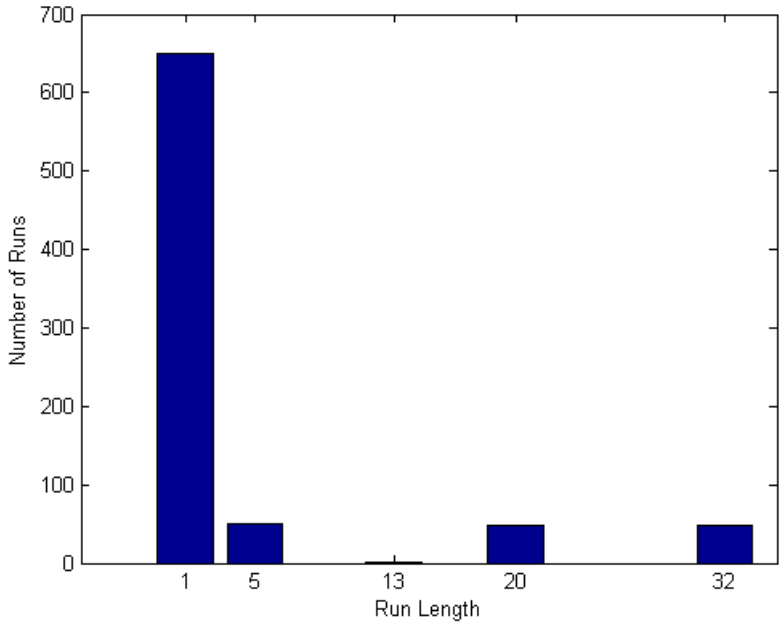

Figure 18: 5 Hz MC1 Dump 2 Run-length Statistics

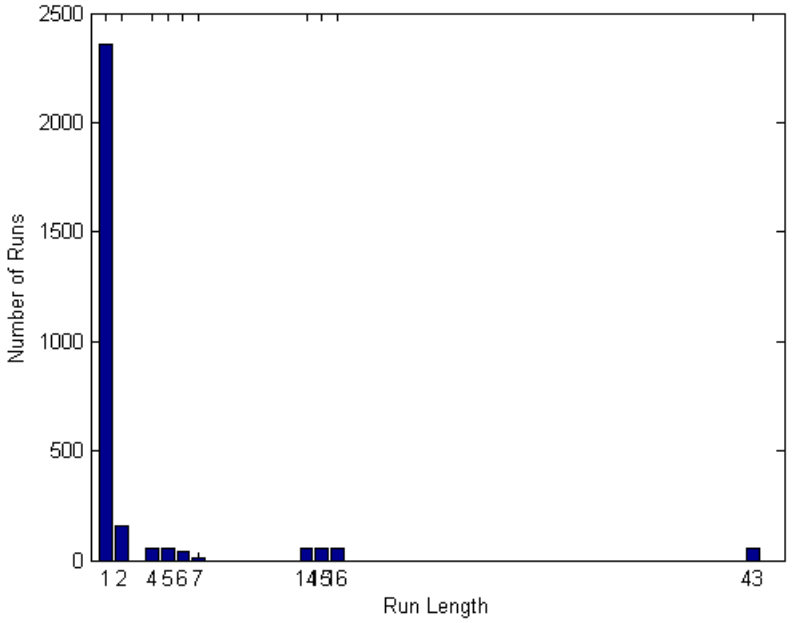

Figure 19: 5 Hz MC1 Dump 2 Run-length Statistics

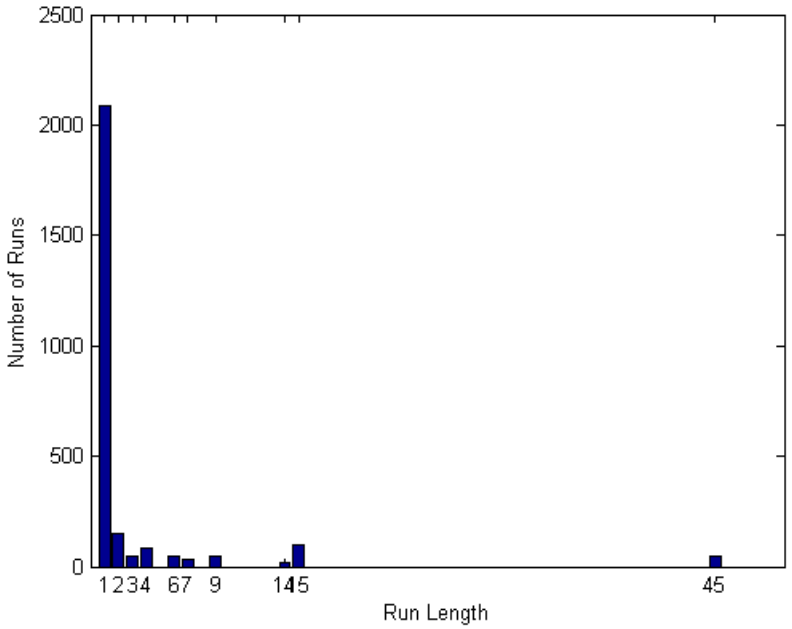

Figure 20: 5 Hz MC1 Dump 2 Run-length Statistics

In the third model, an analysis was performed to confirm the frequency of changes between successive message transmissions. The statistical studies indicate that on average changes for the 5 Hz $_{(MC1)}$, 10 Hz, and 20 Hz rate groups are relatively low as compared to the substantially higher values achieved by 5 Hz $_{(MC2)}$ rate group. The 5 Hz $_{(MC2)}$ rate group achieved an average change of approximately 67.28 percent; whereas, the others maintained a percentages ranging from 0.69 to 5.96 percent. The low percentage of changes for the 5 Hz $_{(MC1)}$, 10 Hz, and 20 Hz rate groups indicate that there is a very high level of redundancies between successive message transmissions. This is encouraging because the repetition of transmitted word locations can be effectively exploited with an appropriate compression algorithm. The results achieved by the 5 Hz $_{(MC2)}$ rate group were less encouraging because the high levels of changes

is not a good indication that an algorithm exploiting such redundancies would be effective for this specific rate group.

Table 2 : Percent of Change within Successive Message Transmissions

	20 Hz		10 Hz		5 Hz	
Statistical Method	MC1	MC2	MC1	MC2	MC1	MC2
Max Changes (%)	57.57	40.34	24.87	21.43	25.71	75.71
Min Changes (%)	0.00	1.45	0.52	0.29	0.00	67.14
Avg. Changes (%)	2.18	4.89	0.85	5.96	0.69	67.28

The overall results from the three models are encouraging, and the potential increases in the effective bandwidth of the1553 bus seem promising. The resulting models show that there are redundancies within the message transmissions which can be utilized to take advantage of and increase the performance capacity of the current system. The development of appropriate compression algorithms can be determined based upon the statistical results from the three models.

CHAPTER FIVE

Proposed Algorithms

The efficient utilization of transmission resources, on the 1553 data bus, requires that some sort compression technique be performed on the data. The transfer of redundant information needlessly congests the transmission bandwidth; reducing time resources which could otherwise be utilized for other operations. Compression is achieved by the exploitation of the inherent redundancies and correlation found within the transmission data. This means that the performance of the developed algorithms will largely be dependant on the source data. The redundancies found from the statistical analysis include:

1. Large number of zeros within data sets
2. Successive repeated data word values
3. Relatively small number of changes between message transmissions

Based upon the statistical models discussed above, three lossless algorithms have been proposed for application to MIL-STD-1553. The three developed methods are: Common Value Tracking (CVT), Modified Run-Length Encoding (MRLE), and Differential Encoding (DE). Each of the algorithms operates on a predefined set of data that is to be transmitted over the bus. Common Value Tracking achieves compression by reducing the excessive amount of zeros within the data composition. Modified Run-Length Encoding takes advantage of the consecutively repeated values, and Differential Encoding exploits the fact that there are only a minor amount of changes occurring between successive message transmissions. The novelty of these methods is that they

require low computational costs, and can be performed with only one pass over the source data. This allows for fast data encoding and decoding processing.

Common Features

The three compression algorithms share a set of common features. Figure 21 below illustrates the structure of the general compressed data frame for the schemes. Note that the top of the compressed data diagram is comprised of a section referred to as the position word. Position words indicate the status of the condition for the encoded data. The position words are followed by the data word section, which contains all the uncompressed data values. As depicted in the diagram, both the position and data word locations have variable sizes. The length of the position word section is a static value which is depends on the original data size. The derivation of the size, in equation 2, will be discussed in the next few paragraphs. The data word section length is based upon the number of uncompressed data values during the encoding process.

The position word is an encoded reference which retains all the information required to reconstruct the encoded data set. Each bit in the collection of position words represents an address from the original block of data. In the first position word, the first bit location is reserved for the compression status; the remaining bits are used to for encoding purposes. The first position word is capable of representing 15 data word locations, while each of the following position words can reference a total of 16 data word locations. For zero tracking, the condition is defined as the presence of a zero in the corresponding address position from the original data. For modified run-length coding, the condition is defined as the presence of a run. The condition for Differential Encoding is defined by the presence of a change between successive data sets.

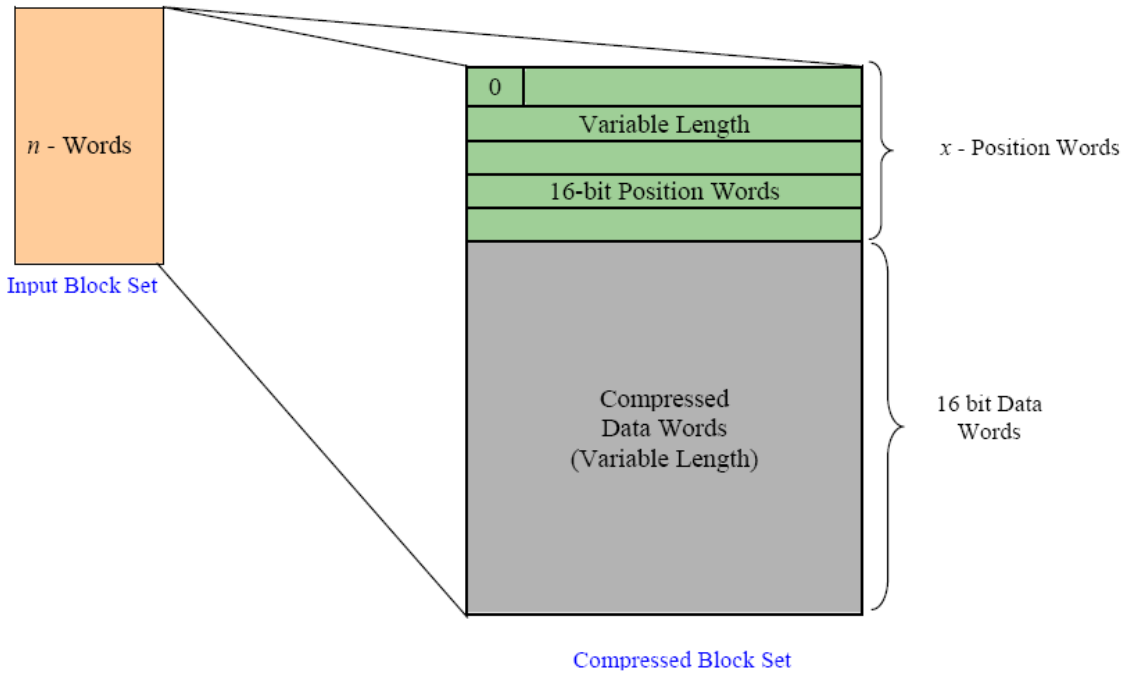

Figure 21 : Variable Length Frame Format – Compressed

Due to the fact that the algorithms operate on a variable length block size, it also requires that the size of the position word also be variable. For example, given a source differential block size denoted by the variable *n*. The calculation of the variable position words can be determined by the division of *n* by the word size of 16 (the number of bits per 1553 data word), yielding a quotient variable *q*. The length size of the position word, P_{Length}, is then given by:

$$P_{Length} = \begin{cases} q & for \quad \mathrm{mod}(n+1/16) = 0 \\ q+1 & for \quad \mathrm{mod}(n+1/16) \neq 0 \end{cases} \qquad (2)$$

When the modulus of *n* divided by 16 is equal to zero, the size of the position word is equivalent to *q*. Alternatively, if the modulus is greater than zero, then the size is equal to *q* plus 1.

An additional common feature among the three algorithms is the implementation of a Compression Status Bit. The compression status bit indicates weather the data transfer contains compressed or uncompressed data. The most significant bit of the first position word is reserved for the status bit. This format was developed in an attempt to limit the data expansion for situations where the data does not compress. As shown in Figure 21, when the status bit is set to binary zero, it indicates that the data within a given block is compressed. This means that the receiving terminal must first decompress the data prior to processing. Figure 22 shows an example of the encoded frame in an uncompressed state.

When the transferred data is uncompressed the compression status bit is set to binary one. Note that during an uncompressed state that the position word only occupies one word location. The additional 15 bits of the position word are unused because no position information is required. For all three techniques, one word location is lost, which leads to an expansion of the source data.

There are cases when compression techniques are not efficient or effective. Under these circumstances it is advantageous to transmit the uncompressed messages rather then the compressed data. This limits expansion by a significant degree. The transfer of an uncompressed block can additionally be beneficial when the excess time required to compress and decompress information exceeds the timing gains attained during transmission. Therefore, allowing for the strict 1553 timing requirements to be maintained.

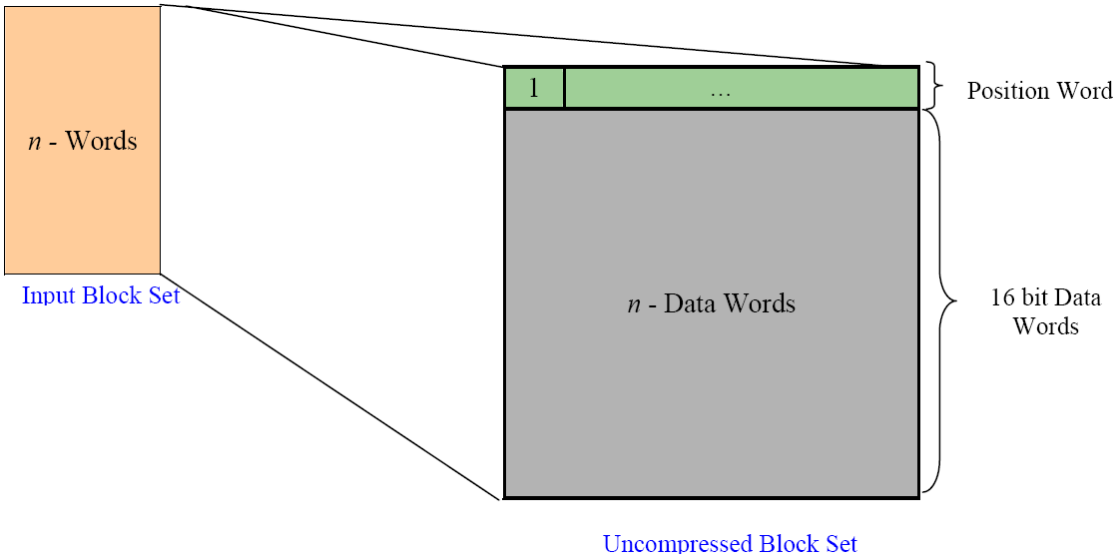

Figure 22 : Variable Length Frame Format – Uncompressed

An alternative method for encoding the compression algorithms is to parse larger source data into 32 word block segments rather than using a variable length frame format. The 32 word frame segments are based upon the natural transmission block size inherited from the protocol of the 1553 data bus. As shown in Figure 23, this means that each transmission of a message set is composed of multiple compressed segments. Since in this case the frame size is known, there are two position words required to address the 31 data words.

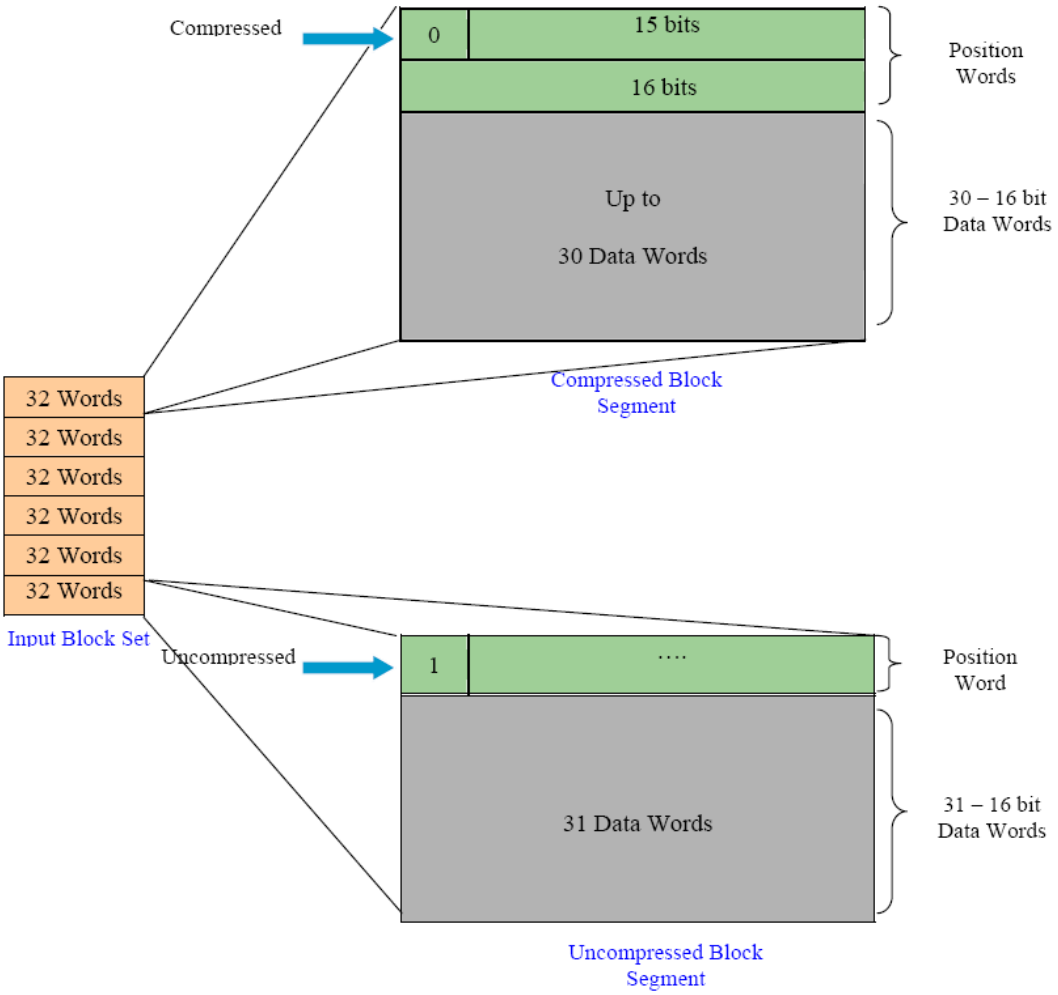

Figure 23 : 32 Word Block Frame Format

Compression Algorithms

Common Value Tracking

Common Value Tracking is a method which compresses data by eliminating the occurrences of a specified value within a given data set. It was proposed based upon the results from the percent of zeros statistics, which showed that on average approximately 43 to 76 percent of the data processed on the F/A-18 was composed of word locations containing 0x0000. For this reason common value tracking is also referred to as Zero Tacking.

The implementation of Zero Tracking is fairly straightforward. This algorithm ensures that only non-zero data and the encoded position words are transmitted on the data bus. Each bit in the position word represents the location of an eliminated 0x0000 word from the original data set. This routine can also be utilized to reduce other commonly occurring data word values and is not restricted solely to the value zero.

The concept behind Zero Tracking is to identify the presence and location of zeros within a data set. As illustrated in Figure 23, the compression process starts with a block of n-data words. The n-length block is partitioned into block segments of length 31. Figure 24 shows an example of the encoding process required for zero tracking. Given an input data set of 31, 16-bit words, the first step to processing the data is to identify the zero locations. In the zero tracking (ZT) column all the zeros are indicated by a binary one and non-zero values by a binary zero. Consider for instance the first four word counts, the binary encoded sequence, located in ZT, are as follows: "1, 1, 1, and 0". This signifies that the first three locations are filled zeros followed by a non-zero data value. The zero tracking encoded sequence continues using this same format with the final position word values corresponds to 0xE6EF and 0x0DFE. Following the bit position words, the non-zero word values are inserted into the buffer in sequential order.

During decompression, zeros are substituted for bit positions containing binary one in the position word. The binary zero locations are substituted with the equivalent data values in the encoded data.

In this example, the encoded data reduces the original word size from 31 to 11, yielding a compression ratio of 2.8. The performance of the compression algorithm is

heavily dependent on the amount of zeros contained within the data set. Therefore a data set composed of fewer zeros will result in a lower compression ratio. If no zeros are present in a 31 word block, the compression bit is set to one and the uncompressed message is instead transmitted. This encoding scheme achieves a worst case expansion of 31/32, corresponding to the state in which no zero words are present.

Word Count (Hex)	Input Data (Hex)	ZT	Encoded Data (Hex)
0	0000	1	E6EF
1	0000	1	0DFE
2	0000	1	FFFF
3	FFFF	0	5604
4	5604	0	5604
5	0000	1	B1F4
6	0000	1	FFFF
7	5604	0	1567
8	0000	1	1567
9	0000	1	FFFF
A	0000	1	5604
B	B1F4	0	
C	0000	1	
D	0000	1	
E	0000	1	
F	0000	1	
10	FFFF	0	
11	1567	0	
12	1567	0	
13	FFFF	0	
14	0000	1	
15	0000	1	
16	5604	0	
17	0000	1	
18	0000	1	
19	0000	1	
1A	0000	1	
1B	0000	1	
1C	0000	1	
1D	0000	1	
1E	0000	1	
1F			

Figure 24: Example of Zero Tracking

Modified Run-Length Encoding

Run-Length Encoding (RLE) is commonly used for the purposes of compressing consecutive sequences of the same data value referred to as a "run". This method has widely been utilized in the area of image compression. The technique exploits the common situation in image data where neighboring pixels are the same. In a traditional run-length encoding scheme, the compression process is conducted by replacing runs by the length of the run and a single corresponding data value. The total count references the length of a given run. For example, consider a situation in which a data value *d* is repeated *n* times, the repeated value is substituted with a single value of *d* and the run-length *n* [16].

The traditional run-length encoding process is further illustrated in more detail in Figure 25. The procedure begins by first scanning the input data for consecutive repeated data values, and then recording the corresponding run lengths. The encoded data is then constructed by inserting the run-length followed by the corresponding data value. To decode the encoded data, the process is to simply repeat the data values by the corresponding run-length in sequential order.

For this example, traditional RLE method is able to reduce the input data size by ten word locations, achieving a compression ratio to 1.41. The statistical characterization of the 1553 bus data indicates that the typical runs seen in the data are not sufficiently long enough to achieve substantial compression. However, the performance of a RLE based approach can be enhanced with a few minor modifications. As seen in the example, the insertion of the run length integers requires that a total of 11 word locations be utilized. This is a substantial amount of storage spaces which is

especially wasteful for short runs. In order to achieve a compression ratio equal to or greater than one, the input data must have run lengths averaging two and higher.

Word Count (Hex)	Input Data (Hex)	Encoded Data (Hex)
0	0000	3
1	0000	0000
2	0000	1
3	FFFF	FFFF
4	5604	4
5	5604	5604
6	5604	2
7	5604	0000
8	0000	2
9	0000	B1F4
A	B1F4	4
B	B1F4	0000
C	0000	1
D	0000	FFFF
E	0000	1
F	0000	A345
10	FFFF	7
11	A345	0000
12	0000	2
13	0000	B352
14	0000	4
15	0000	0000
16	0000	
17	0000	
18	0000	
19	B352	
1A	B352	
1B	0000	
1C	0000	
1D	0000	
1E	0000	
1F		

Figure 25: Example of Traditional RLE

Another drawback of this routine is that the worst case expansion for the example is 31/62. This occurs when no repeated values are present. The worst case expansion

effectively doubles the original input size requirements rendering it ineffective for use on the MIL-STD-1553 data bus.

Modified run-length encoding is an adaptation of the traditional RLE approach. The MRLE improves on the traditional version by incorporating position words. The bits within the position words can be used to indicate the beginning and end of a run. This encoding modification makes the MRLE compression method dramatically more efficient for the 1553 bus data. Additionally, it allows for a reduction in the worst case expansion.

Figure 26 illustrates the MRLE technique as compared to the traditional version. The encoding protocol for the position word sequence is defined by indicating the beginning of a run with a binary zero and the continuation of a run as a binary one. Therefore the beginning of any new runs sequence is denoted by a zero. The input data in this example is the same as that in Figure 25, allowing for the ability to compare the compression of the traditional and the modified versions.

When examining the tracking process, in the RL column of the table, the first run of three values 0x0000 are noted by '0,1,1'. At word count three, the value resets to zero because it is the end of a run and a beginning of new series. The value 0xFFFF is not repeated and is therefore not a run. The next word position bit at word count 0x4 is also a zero. Once the tracking sequence has been completed it is then encoded and placed in the position word. The final encoded tracking sequence is indicated by the hex values 0x6757 and 0x1FAE. To complete the encoding process the locations beginning runs, indicated by a binary zero, are placed sequentially into the encoded buffer.

In general, data sets with a larger number of long runs produce improved compression ratios. In the example, see Figure 26, the MRLE compression technique yields a compression ratio of approximately 2.38. The implementation of the MRLE, effectively improved the compression ratio by a factor of approximately 1.69 compared to the traditional version. Additionally, the worst case expansion for the algorithm is 31/32.

Overall, the modified version is a dramatic improvement upon conventional RLE methods. Unlike the traditional method, the example of the MRLE at a minimum must have one run of two to maintain a compression ratio of one or more. Whereas, the traditional method requires an average of at least two run to break even. The position tracking technique makes more efficient use of the fixed space.

Differential Encoding

Differential Encoding is an approach which attempts to reduce bus traffic by only transmitting the new information between successive data sets. The premise behind this method is that if a data word location has not changed since the previous transmission there is no need to transmit the same value. Therefore, selected positions within the new data are transferred to update the changed data values. As before, position words are used to indicate where in the data set changes have occurred.

In the prior sections it was shown that percent of change statistics indicate that the successive changes between data sets, for the 20 Hz and 10 Hz rate groups, are relatively small. The statistics also show that the 5 Hz rate group exhibits a higher rate of change at 69 percent. Although 20 Hz and 10 Hz rate groups should achieve good

performances marks, the statistics imply that this compression method might not be as effective for the 5 Hz rate group.

Word Count (Hex)	Input Data (Hex)	RL	Encoded Data (Hex)
0	0000	0	6757
1	0000	1	1FAE
2	0000	1	0000
3	FFFF	0	FFFF
4	5604	0	5604
5	5604	1	0000
6	5604	1	B1F4
7	5604	1	0000
8	0000	0	FFFF
9	0000	1	A345
A	B1F4	0	0000
B	B1F4	1	B352
C	0000	0	0000
D	0000	1	
E	0000	1	
F	0000	1	
10	FFFF	0	
11	A345	0	
12	0000	0	
13	0000	1	
14	0000	1	
15	0000	1	
16	0000	1	
17	0000	1	
18	0000	1	
19	B352	0	
1A	B352	1	
1B	0000	0	
1C	0000	1	
1D	0000	1	
1E	0000	1	
1F			

Figure 26: Example of MRLE

The encoded output, for Differential Encoding, is dependent on the status of values in the current data set as compared to the old data set. In Figure 27, an example of the encoding process is shown. Although the differential encoding operates on a

variable block size, for the purposes of this example the input data size is specified to be 31 words.

Word Count (Hex)	Old Data (Hex)	Input Data (Hex)	DT	Encoded Data (Hex)
0	0054	0054	0	20DE
1	0815	0815	0	1410
2	*AF58*	*12F8*	1	*12F8*
3	0000	0000	0	*9FB2*
4	0000	0000	0	*FDA9*
5	6542	6542	0	*D51F*
6	FFFF	FFFF	0	*A512*
7	FFFF	FFFF	0	*F586*
8	*2222*	*9FB2*	1	129F
9	*8966*	*FDA9*	1	A18F
A	8966	8966	0	F183
B	0543	A14F	1	
C	*541D*	*D51F*	1	
D	*1F47*	*A512*	1	
E	*0052*	*F586*	1	
F	0000	0000	0	
10	5604	5604	0	
11	5604	5604	0	
12	0000	0000	0	
13	0000	129F	1	
14	B1F4	B1F4	0	
15	FFFF	A18F	1	
16	A345	A345	0	
17	0000	0000	0	
18	0000	0000	0	
19	0000	0000	0	
1A	B352	B352	0	
1B	B352	F183	1	
1C	B352	B352	0	
1D	0000	0000	0	
1E	0000	0000	0	
1F				

Figure 27 : Example of Differential Encoding

The encoding scheme relies upon the identification of the changes between the two successive data sets. Similar to the previous two algorithms, differential encoding employs position words to indicate the location of changes. Word locations which

exhibit changes are denoted with a binary one and non-changing positions are set to zero in the tracking sequence. The word count locations which contain changes are the following: 0x2, 0x8, 0x9, 0xB, 0xC, 0xD, 0xE, 0x13, 0x15, and 0x1B. The final encoded tracking sequence yields two word values of 0x20DE and 0x1410, which are placed into the position words. The position words are then preceded by the six words value exhibiting changes.

Differential encoding for this example yields a compression ratio of approximately 2.81. The worst case expansion of this technique occurs when all word locations requiring updating. Even under this condition, Differential Encoding generates a worst case expansion of 31/32.

CHAPTER SIX

Findings and Implementation Challenges

Fixed versus Variable Transmission Buffer Size

Upon the completion of the compression process, the encoded data is placed into a message transmission buffer for transfer to the receiving mission computer. The data input/output support is implemented by means of a software routine known as Q1PCK contained in the OFP. The Q1PCK routine takes a list of address location and moves the data from these locations to a contiguous block of memory, as shown in Figure 31. This block of memory can be viewed as a transmission buffer since the interface between memory and the 1553 card will result in the serial transmission of the contents of the memory across the bus. The receiving mission computer then acquires the data, by way of another 1553 interface card, placing the message data into memory. The data is unpacked by the OFP routine Q2UPK, which can then be processed as needed [4].

Ideally, the transmission block size will be significantly reduced after the application of the data compression routine. The question then arises as to how the system will transmit the data from the transmission buffer, since this requires knowledge of how much data the transmission buffer contains. As opposed to the case where compression is not used, we note that the length of the transmission buffer is a function of how well the original data compresses and can vary significantly.

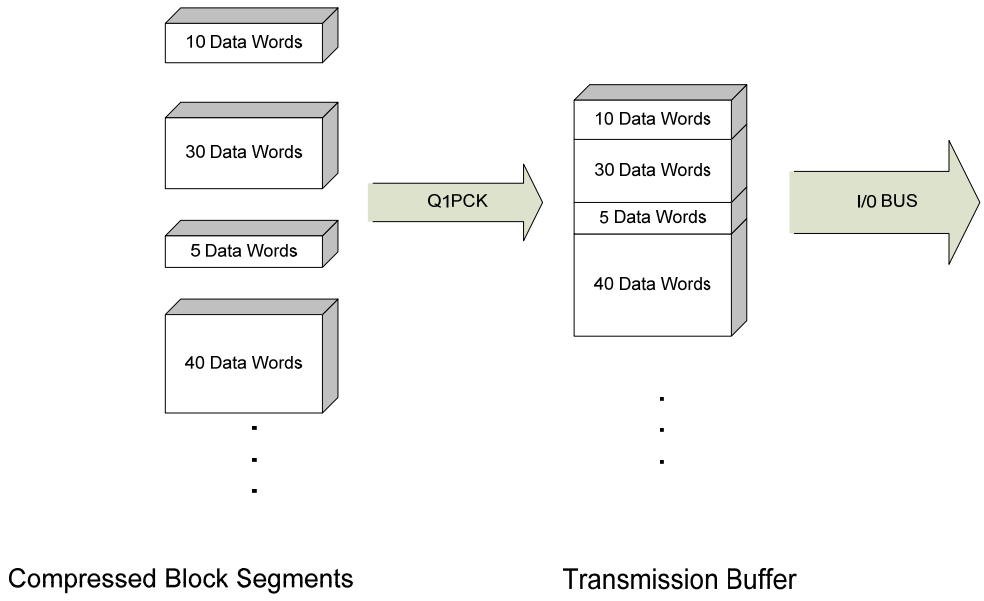

Figure 28 : Q1PCK and 1553 Message Packaging

The original protocol for sending the transmission buffer employed an interrupt based method that is based on a fixed length transmission. The primary advantage of continuing to use a fixed length buffer lies in the ease of implementation. It is believed that changing the length of the transmission buffer to a smaller, fixed value would be a straightforward software change. On the other hand, the implementation of a variable length transmission buffer (variable in the sense that it can change from one transmission to the next) is believed to be a significantly more complicated operational change.

Fixed length transmission describes a procedure in which an upper bound is established and used for the transmission size. In other words, a fixed allocation of word locations is allotted for each message transmission. The bounding of the fixed length buffer is based upon worst case statistics. Since the length of the transmission

buffer is fixed, to ensure that all the data is transmitted one must use a buffer length size that will accommodate the worst case compression.

Even if one relaxes the worst case constraints for start-up conditions, the fact that a fixed-length scheme is driven by worst case statistics is very restrictive. Work has been done which consider an overflow buffer scheme that allows all of the data from the original transmission block data to eventually be transmitted over the bus. This approach would allow a smaller fixed buffer length to be used and is motivated by the notion that worst case conditions are relatively rare. When a worst case condition does occur a portion of the data remains unsent and is placed into an overflow buffer. The non-transmitted word locations are then given priority to be sent with the following message sequence.

The serious handicaps of this method are: performance bottlenecks of latent data, and possible system updates of queued data. System traffic jams could arise during situations where consecutive overflow conditions occur. Assuming this circumstance arises, there is a risk that some data sequences might be delayed by three or more cycles. Finally, it is conceivable that data retained in the overflow buffers might be updated by the system while being tied up. Consequently causing the overflow data to be outdated, yielding it meaningless for continued transfer. Although there are some benefits to utilizing Fixed Length Transmissions the reduction in performance required to avoid latency problems is a serious problem.

The second proposed strategy, Variable Length Transmission, does not limit or bound the length of the encoded data. Instead, the transmission buffer length is dependent on the compression performance for each message. The advantage of

Variable Length Transmission is the fact that there is no overflow buffer required. The one disadvantage that must be addressed is that without a bounded length, low performing compression sequences can cause possible system timing violations due to large message buffer sizes [15]. As mentioned earlier, the MIL-STD-1553 protocol designates a maximum 50ms execution time for each message sequence. The problem arises in that, if the compression length is to long there is a possibility that the execution time (time required to encode, transfer, decode and compute) will exceed its limit.

Given the worst case compression ratios, discussed further in upcoming sections, the conclusion that was made is that a variable length interrupt driven method for transmitting compressed data blocks appears to be the best choice.

Compression Error Propagation and Handling

An inherent problem of utilizing compression algorithms is the increased susceptibility to error propagation in the event of a transmission error. Error propagation is the loss or damage of alternative word locations based upon an initial generated error. All proposed compression algorithms are susceptible to some degree of error propagation, but the overall magnitude varies sharply depending on the method. In each case a discussion and analysis of the error sensitivity will be examined. Error propagation, if not detect, can result in catastrophic system failures which is a major concern. Therefore, it is of critical importance that error rates and effects be minimized.

The two sections of the encoded frame prone to errors are: the position words, and the data words. A generated error in the data word section is for the most part localized to the word locations requiring the insertion of the specific encoded data word. Therefore the scale of the error propagation hinges on the dependency of the encoded

data word location during decompression. In contrast, errors in the position words are more likely to have potentially devastating consequences to the integrity of the message. This is because the position word contains the reference information required to reconstruct of the original message. Lastly, for all three algorithms an undetected bit error of the compression status bit invalidates the entire transmission. This is the worst case scenario because no data is able to be recovered if not corrected.

One of the major challenges related to error problem originates from the manner in which data is processed prior to transmission onto the data bus. In the previous section, a discussion about the 1553 software routine Q1PCK was addressed. Although the compression algorithms might produce individual encoded block segments, prior to transmission all the segments are collected and placed into a single transmission buffer, as depicted in Figure 28. The data is then transmitted onto the bus in individual 32 word segments which are then reconstructed on the receiving end. By placing all the segments in a single buffer it therefore becomes impossible to distinguish the individual units that compose the complete transmission buffer, without referring to the position words.

If a position word location encounters an error while in transmission, in the worst case the number of encoded data word values no longer matches the total number referenced by the corresponding position words. This problem can then possibly falsely identify a position word, for the successive block segment as a data word location. This situation would cause an avalanche effect which invalidates all the following word locations, yielding a corrupted transmission sequence. This problem can not be localized to just a few word locations. To remedy this problem it is advisable that all

position word locations are placed at the beginning of the transmission buffer followed by the data word locations.

Examples of the position and data word errors, for the three algorithms, are illustrated in Figure 29 thru 32. These examples utilize a 16 word data set, which only requires one position word. By exploring smaller examples, it then easier to understand the error complexity for each method and how it relates to the larger problems of transmission. The diagrams are organized in three parts. Section (a) shows an uncorrupted data transmission. Section (b) demonstrates an example of a multi-bit error in the position word and the resulting decompression outcome. While there are no provisions to detect multiple-bit errors, the 1553 data bus fortunately is equipped with a parity bit, allowing for the detection to single bit errors. Chapter 2 further discusses the other error detection and handling capabilities of the 1553 bus. Lastly, section (c) illustrates the error propagation for generated errors in the data words position of an encoded frame.

In regards to Zero Tracking, in the worst case, a single event multi-bit error in the position word can dislocate all subsequent non-zero word values beginning with the initial error location, as illustrated in Figure 29 (b). Although such an error can devastate the non-zero word values, the zero word locations predominately remain uncorrupted with exception to the reference locations containing errors. A similar error in the data word section has less damaging effects than that of the position word. To facilitate understanding, an example of this error is shown in Figure 29 (c). An error in the data word is restricted to the specific word location, and does not propagate throughout the message.

The effects of error propagation within the modified run-length scheme, has slightly worse affects than that of CVT. Specifically, an error in the position word will either wrongly indicate the beginning or continuation of a run. From the run length statistics it was identified that some runs are greater than 32 words in length. In the example, show in Figure 30 (b), a two bit error occurs in the position word in word locations 0x2 and 0x6. In this case, the error condition causes there to be an invalidation of all decoded data words beginning at the primary bit error location of 0x2. On the other hand, an error in the data word would only impact the affected run segment, as illustrated in Figure 30 (c). The magnitude of the error is dependent on the length of the affected run. Under this condition all other run sequences would remain unaffected.

Finally, an analysis of the Differential Encoding scheme reveals that an occurrence of an error in the position word would cause an ignored or unwarranted updating of a data word location. For example, as shown in Figure 31(b), a double error bit error of one(word count 0x3) and zero (word count 0x6) in the position causes all subsequent data values requiring updates to be erroneous. In this case the data word locations have been shifted out of position. As depicted in Figure 31(c), an error in the data word only corrupts the single position, leaving all other locations untouched. The error propagation characteristics of Differential Encoding are very similar to that of the CVT.

An additional continuity check that can easily be applied is to make sure that the number of changes represented by the position words also corresponds to the number of values contained within the data words. For example, in Figure 31, the number of

changes in the position word does not match with the number of data values in the encoded data set. There are not enough data values to complete the decoding process, therefore word locations 0xD and 0xE are empty. If the two total values do not correspond, it is an indication of an error in the transmitted message.

From the analysis of error propagation we can conclude that the most devastating error propagation effect occurs within the position words. This is because the position words act like a map which details the placement and orientation of every value. If this information is corrupted the sequential placement of data values is also corrupted. As state earlier, it is necessary that all position and data word locations are placed in groups of their respective types in order to mitigate possible misidentification caused by an error during transmission. Another possible solution to this error propagation problem is to employ an error detection/correction technique, such as a CRC method specifically to the position word section of the transmission message. It is not as necessary to apply this technique to the data words locations, because the error propagation of this region is relatively localized.

Compression Ratio Statistics

Tables 3 through 5 show the compression ratios for the proposed compression algorithms. The results are evaluated according to their respective rate group and the origin of the message transfer (MC1 or MC2) so that a comprehensive analysis can be provided.

Focusing on the highlights of the compression ratio results, it is unquestionable that the best overall performance was achieved with Differential Encoding.

Word Count (Hex)	Input Data (Hex)	Encoded Data (Hex)
0	0000	**E6EF**
1	0000	FFFF
2	0000	5604
3	FFFF	5604
4	5604	B1F4
5	0000	
6	0000	
7	5604	
8	0000	
9	0000	
A	0000	
B	B1F4	
C	0000	
D	0000	
E	0000	
F	0000	

(a.)

Word Count (Hex)	Encoded Data (Hex)	ZT	Decoded Data (Hex)
0	**F4EF**	1	0000
1	FFFF	1	0000
2	5604	1	0000
3	5604	1	0000
4	B1F4	0	FFFF
5		1	0000
6		0	5604
7		0	0000
8		1	0000
9		1	0000
A		1	0000
B		0	5604
C		1	0000
D		1	0000
E		1	0000
F		1	0000

(b.)

Word Count (Hex)	Encoded Data (Hex)	ZT	Decoded Data (Hex)
0	**E6EF**	1	0000
1	FFFF	1	0000
2	5634	1	0000
3	5604	0	FFFF
4	B1F4	0	5634
5		1	0000
6		1	0000
7		0	5604
8		1	0000
9		1	0000
A		1	0000
B		0	B1F4
C		1	0000
D		1	0000
E		1	0000
F		1	0000

(c.)

Figure 29 : CVT (A.) Uncorrupted Encoded Message (B.) Two-Bit Error in the position word (C.) Two-Bit Error in the data word

Word Count (Hex)	Input Data (Hex)	Encoded Data (Hex)
0	0000	**6757**
1	0000	0000
2	0000	FFFF
3	FFFF	5604
4	5604	0000
5	5604	B1F4
6	5604	0000
7	5604	
8	0000	
9	0000	
A	B1F4	
B	B1F4	
C	0000	
D	0000	
E	0000	
F	0000	

(a.)

Word Count (Hex)	Encoded Data (Hex)	RL	Decoded Data (Hex)
0	**4557**	0	0000
1	0000	1	0000
2	FFFF	**0**	**FFFF**
3	5604	0	**5604**
4	0000	0	**0000**
5	B1F4	1	**0000**
6	0000	**0**	**B1F4**
7		1	**B1F4**
8		0	**0000**
9		1	**0000**
A		0	----
B		1	----
C		0	----
D		1	----
E		1	----
F		1	----

(b.)

Word Count (Hex)	Encoded Data (Hex)	RL	Decoded Data (Hex)
0	**6757**	0	0000
1	0000	1	0000
2	FFFF	1	0000
3	**5724**	0	FFFF
4	0000	0	**5724**
5	B1F4	1	**5724**
6	0000	1	**5724**
7		1	**5724**
8		0	0000
9		1	0000
A		0	B1F4
B		1	B1F4
C		0	0000
D		1	0000
E		1	0000
F		1	0000

(c.)

Figure 30 : MRLE (A.) Uncorrupted Encoded Message (B.) Two-Bit Error in the position word (C.) Two-Bit Error in the data word

Word Count (Hex)	Old Data (Hex)	Input Data (Hex)	Encoded Data (Hex)
0	0054	0054	**20DE**
1	0815	0815	*AF58*
2	*12F8*	*AF58*	*2222*
3	0000	0000	*8966*
4	0000	0000	0543
5	6542	6542	*541D*
6	FFFF	FFFF	*1F47*
7	FFFF	FFFF	*0052*
8	*9FB2*	*2222*	
9	*FDA9*	*8966*	
A	8966	8966	
B	A14F	0543	
C	*D51F*	*541D*	
D	*A512*	*1F47*	
E	*F586*	*0052*	
F	0000	0000	

(a.)

Word Count (Hex)	Encoded Data (Hex)	DT	Decoded Data (Hex)
0	**26FE**	0	0054
1	*AF58*	0	0815
2	*2222*	1	*AF58*
3	*8966*	0	0000
4	0543	1	*2222*
5	*541D*	0	6542
6	*1F47*	0	FFFF
7	*0052*	0	FFFF
8		1	*8966*
9		1	**0543**
A		1	*541D*
B		1	*1F47*
C		1	*0052*
D		1	-----
E		1	-----
F		0	0000

(b.)

Word Count (Hex)	Encoded Data (Hex)	DT	Decoded Data (Hex)
0	**20DE**	0	0054
1	*AF58*	0	0815
2	*2222*	1	*AF58*
3	*8966*	0	0000
4	**1563**	0	0000
5	*541D*	0	6542
6	*1F47*	0	FFFF
7	*0052*	0	FFFF
8		1	*2222*
9		1	*8966*
A		0	8966
B		1	**1563**
C		1	*541D*
D		1	*1F47*
E		1	*0052*
F		0	0000

(c.)

Figure 31 : DE (A.) Uncorrupted Encoded Message (B.) Two-Bit Error in the position word (C.) Two-Bit Error in the data word

Out of the three algorithms, differential encoding was consistently the top performer for the majority of the three rate groups. The algorithm was able to impressive achieve average compression ratios as high as 14.45 to 1. Differential encoding surpassed the compression performances of CVT by a factor of 4.03 and MRLE by a factor of 5.24.

The results indicate that Zero Tracking achieved the second best performance. Although the compression ratios were not as high as those attained by the differential method, the zero tracking scheme was able to slightly edge out the figures attained by MRLE. On average CVT outperformed MRLE by a factor of 1.31. It is clear from the results that modified run-length encoding generated the worst performance results.

The one encouragement that can be taken from the results is that even the poorest performing algorithm was able to achieve a compression ratio greater than one. It should also be noted that all the minimum compression values were also greater than one, which indicates that there is no worst case expansion over an entire message block. Although the compression ratios give good performance measures, it is only one component in discerning the suitability of the algorithms for application on the MIL-STD-1553.

Table 3 : Zero-Tracking Compression Ratios

	20 Hz		10 Hz		5 Hz	
Statistical Method	MC1	MC2	MC1	MC2	MC1	MC2
Maximum	1.80	3.16	3.53	7.60	3.62	1.25
Minimum	1.55	2.12	3.10	3.60	3.08	1.20
Average	1.60	2.51	3.27	4.77	3.49	1.24

Table 4 : Modified Run-Length Encoding Compression Ratios

	20 Hz		10 Hz		5 Hz	
Statistical Method	MC1	MC2	MC1	MC2	MC1	MC2
Maximum	1.44	2.72	2.50	4.58	2.35	1.18
Minimum	1.30	1.79	2.33	2.47	2.16	1.16
Average	1.33	2.11	2.43	3.09	2.32	1.17

Table 5 : Differential Encoding Compression Ratios

	20 Hz		10 Hz		5 Hz	
Statistical Method	MC1	MC2	MC1	MC2	MC1	MC2
Maximum	14.69	12.93	16.00	15.21	15.00	1.34
Minimum	3.21	2.14	1.56	3.60	3.08	1.20
Average	14.45	9.24	13.30	10.44	14.37	1.34

Micro-Instruction Timing Analysis

This section discusses the timing performances for the compression algorithms based upon best case representative compression ratio statistics. Rather than using all 12 bus traces, one trace has been chosen in order to obtain l representative analysis. The calculation of the timing analysis was performed by evaluating the total number of micro-instructions required to run each program. Based upon the timing estimation it is then possible to determine the viability of the proposed algorithms.

The two MIL-STD-1553 components which are critical to processing and transmission of data are the VHSIC Processor Modules (VPMs) and the Discrete and Serial Module (DSM). The VPM is designated as the processing unit (CPU) for the AYK-14 which operates at a machine cycle rate of 62.5 nanoseconds. The DSM coordinates all I/O operations allowing for communication between the two mission

computers on Bus #3. The VPM interfaces with the DSM by way of the XBUS, which is the primary bus utilized for I/O transfers [4].

In a normal 1553 message handling protocol, for example assume that MC1 were to receive a message transmission from the I/O processor. This causes the system interrupt handler to receive a signal indicating the completed message transfer. Prior to processing the data is unpacked. The received data is then processed by the VPM and then the results are packed for transmission to MC2. Once the data has been packed, MC2 sends a Mode Command to MC2 indicating that the data is ready [4].

The time savings achieved through the implementation of a compression method is expressed by the following parameters: the uncompressed data bus transfer *T*, the compressed data bus transfer time T_{tc}, the compression time T_d, and the decompressing time T_d. The compression bus transfer time gains, T_{diff}, is expressed in equation (3). In this equation the variables *T* and T_{tc} are both multiplied by a factor of two in order to account for the receiving and transmission of data on the bus.

$$T_{diff} = 2 * T - 2 * T_{tc} - T_c - T_d \tag{3}$$

$$T = R_{Wt} * W_{total} \tag{4}$$

$$T_{tc} = R_{Wt} * W_c \tag{5}$$

The uncompressed transmission time for the *T*, in equation (4), is given by the transmission rate R_{Wt} (μs/word) times the total uncompressed word size W_{total}. The rate R_{Wt} is equal to 20 μs/word. The compressed transmission time T_{tc} was derived by multiplying R_{Wt} by the compressed word size W_c.

$$T_c = M_C * R_{MI} \quad (6)$$

$$T_d = M_D * R_{MI} \quad (7)$$

The time required to compress T_C is described as the number of compression microinstructions M_C times the execution rate R_{MI} (μs /word). The values of R_{MI} is equal to 0.0625 μs /word. Similarly, the required decompress time T_d is given by the multiplication of the number of decompression M_D by R_{MI}.

Figure 32 illustrates the derivation of T_{diff}, given in equation 3, by showing the timing comparison of uncompressed to compressed data transfer. The value of T_{diff} should be positive if the decompression/compression (T_d, T_c) execution times as well as the compressed data transfer times ($2*T_{tc}$) are less than the time required to transfer uncompressed data ($2*T$). In the case that the value is negative, it is an indication that the compressed data transfer execution time exceeds that of the uncompressed. The MC Processing time is a static value.

For the purposes of the timing evaluation the compression ratio results a single bus trace (fli_4a_v3.txt) was selected. The data from the trace is representative of normal bus transfer conditions on the 1553 data bus, which offers reliable experimental timing results. From the micro-instruction calculations and the bus trace compression results, it was then possible to compute the difference between uncompressed and compressed data transfers T_{diff}. Included in the computations is the additional number of microinstructions that can be processed based upon the timing gains achieved during compression.

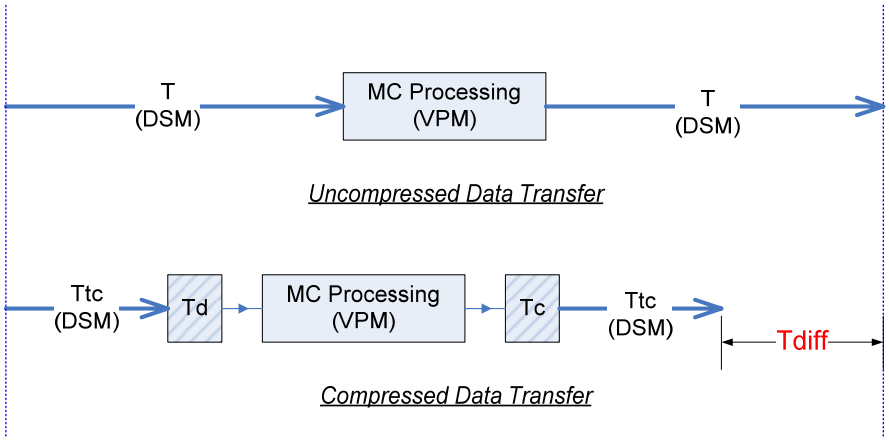

Figure 32 : Transmission Timing Gains (Tdiff) Diagram

Table 6 summarizes the total size of the block set for each respective rate group. Given the block size the total number of position words required to address the message set is included in the table. Table 7-15 review the timing results for each compression algorithm, showing the percentage gains compared to uncompressed data transfer timing simulations. The tables are divided based upon average, minimum, and maximum compression ratios behaviors for each algorithm. Although the results should not be interpreted as a comprehensive analysis, it allows for basic conclusions to be drawn from the timing statistics.

The results from the simulation show that the two best performing algorithm were Differential Encoding and Zero Tracking. As shown in Table 9, the differential algorithm was able to achieve timing gains ranging from 118.41 to 147.60 percent, in relation to average compression ratio statistics. Although for 5 Hz $_{(MC2)}$, the differential compression causes a timing expansion which nullifies any possible benefits of compression. These poor marks also attained for the minimum and maximum timing results. This effectively causes this method to be useless for the 5 Hz rate group. In

regards to the minimum timing statistics the differential method performs well, with exception to 20 Hz $_{(MC1)}$ which has lower percentage gains of 15.95 percent.

On average, although zero tracking did not achieve as high of marks at that of differential encoding, it was able to attain respectable results. The results ranged from timing increases of 14.78% on the low end, and 121.45% on the high end. However the most interesting result from zero tracking is not the average performances but rather the minimum cases. Out of the three methods, zero tracking achieves the best timing marks for the worst case scenarios. The timing increase results range from 14.78% to 120.40%. These results make zero tracking a very credible candidate.

Although, the results for MRLE were not horrible, they were worst of the three algorithms and there are no other compelling factors to support MRLE.

Table 6 : Block Size for Rate Groups – fli_4a_v3.txt

Category	20 Hz	10 Hz	5 Hz
BLOCK SIZE	304	382	105
# OF POSITION WORDS	19	24	7

Table 7 : ZT Timing Analysis – fli _4a_v3.txt (Average)

	20 Hz		10 Hz		5 Hz	
Category	MC1	MC2	MC1	MC2	MC1	MC2
Compression (avg.)	1.594	2.578	3.264	3.665	3.5	1.25
Tdiff (μs)	3012.12	5960.59	8753.99	9279.06	2491.5	310.44
% of Timing Increase	49.54%	98.04%	114.58%	121.45%	118.64%	14.78%
# of extra microinstructions	48193.92	95369.44	140063.84	148464.96	39864	4967.04

Table 8 : MRLE Timing Analysis – fli _4a_v3.txt (Average)

Category	20 Hz		10 Hz		5 Hz	
	MC1	MC2	MC1	MC2	MC1	MC2
Compression (avg.)	1.324	2.14	2.438	2.497	2.348	1.18
Tdiff (μs)	1221.62	4639.54	6663.9	6819.26	1757.4	48.44
% of Timing Increase	20.09%	76.31%	87.22%	89.26%	83.69%	2.31%
# of extra microinstructions	19545.92	74232.64	106622.4	109108.16	28118.4	775.04

Table 9 : DE Timing Analysis – fli _4a_v3.txt (Average)

Category	20 Hz		10 Hz		5 Hz	
	MC1	MC2	MC1	MC2	MC1	MC2
Compression (avg.)	11.185	8.401	14.461	5.205	14.369	1.344
Tdiff (μs)	8654.59	8232.85	11248.67	9046.25	3099.56	-227.5
% of Timing Increase	142.35%	135.41%	147.23%	118.41%	147.60%	-10.83%
# of extra microinstructions	138473.44	131725.6	179978.72	144740	49592.96	-3640

Table 10 : ZT Timing Analysis – fli _4a_v3.txt (Max)

Category	20 Hz		10 Hz		5 Hz	
	MC1	MC2	MC1	MC2	MC1	MC2
Compression (max)	1.626	2.588	3.293	3.723	3.5	1.25
Tdiff (μs)	3173.68	6000.98	8794.38	9319.45	2491.5	310.44
% of Timing Increase	52.20%	98.70%	115.11%	121.98%	118.64%	14.78%
# of extra microinstructions	50778.88	96015.68	140710.08	149111.2	39864	4967.04

Table 11 : MRLE Timing Analysis – fli _4a_v3.txt (Max)

Category	20 Hz		10 Hz		5 Hz	
	MC1	MC2	MC1	MC2	MC1	MC2
Compression (max.)	1.336	2.14	2.453	2.514	2.348	1.18
Tdiff (μs)	1299.3	4639.54	6702.74	6858.1	1757.4	48.44
% of Timing Increase	21.37%	76.31%	87.73%	89.77%	83.69%	2.31%
# of extra microinstructions	20788.8	74232.64	107243.84	109729.6	28118.4	775.04

Table 12 : DE Timing Analysis– fli _4a_v3.txt (Max)

Category	20 Hz		10 Hz		5 Hz	
	MC1	MC2	MC1	MC2	MC1	MC2
Compression (max.)	14.476	9.857	14.692	5.224	15	1.346
Tdiff (μs)	8935.75	8467.15	11248.67	9046.25	3099.56	-227.5
% of Timing Increase	146.97%	139.26%	147.23%	118.41%	147.60%	-10.83%
# of extra microinstructions	142972	135474.4	179978.72	144740	49592.96	-3640

Table 13 : ZT Timing Analysis – fli _4a_v3.txt (Min)

Category	20 Hz		10 Hz		5 Hz	
	MC1	MC2	MC1	MC2	MC1	MC2
Compression (min)	1.551	2.571	3.157	3.608	3.5	1.25
Tdiff (μs)	2810.17	5960.59	8592.43	9198.28	2491.5	310.44
% of Timing Increase	46.22%	98.04%	112.47%	120.40%	118.64%	14.78%
# of extra microinstructions	44962.72	95369.44	137478.88	147172.48	39864	4967.04

Table 14 : MRLE Timing Analysis – fli _4a_v3.txt (Min)

Category	20 Hz		10 Hz		5 Hz	
	MC1	MC2	MC1	MC2	MC1	MC2
Compression (min.)	1.314	2.14	2.365	2.48	2.348	1.18
Tdiff (μs)	1182.78	4639.54	6469.7	6780.42	1757.4	48.44
% of Timing Increase	19.45%	76.31%	84.68%	88.75%	83.69%	2.31%
# of extra microinstructions	18924.48	74232.64	103515.2	108486.72	28118.4	775.04

Table 15 : DE Timing Analysis – fli _4a_v3.txt (Min)

Category	20 Hz		10 Hz		5 Hz	
	MC1	MC2	MC1	MC2	MC1	MC2
Compression (min.)	1.592	2.571	3.265	3.608	3.5	1.25
Tdiff (μs)	969.55	4390.33	6984.41	7499.87	2021.78	-508.66
% of Timing Increase	15.95%	72.21%	91.42%	98.17%	96.28%	-24.22%
# of extra microinstructions	15512.8	70245.28	111750.56	119997.92	32348.48	-8138.56

CHAPTER SEVEN

Conclusions

Motivated by the need to enhance the bandwidth performance of the MIL-STD-1553 Data Bus, this research has attempted to present possible data compression options for use on the current legacy system. In this thesis we have discussed and presented the following topics:

1. We have explored the redundancies inherent within 1553 message transmissions. (Zero Value Compositions, Run-Length, Percent of Change Statistics) Based upon these statistics, three compression algorithms were studied.
2. We have discussed the challenges confronted with implementing the compression algorithms.
3. We have demonstrated that the benefits of effective bandwidth increase that can be achieved through the use of ZT and DE compression methods.

In summary the results obtained from the research show the algorithms provide viable options for use on the 1553 data bus. The performance gains achieved by the Zero Tracking and Differential Encoding are reassuring in that they can possibly be implemented on the 1553 bus. In the case of differential encoding, the poor performance attained in the 5 Hz rate group removes it from consideration of being applied to that specific rate group.

Although the timing gains for the Differential Encoding on average were higher than the other comparable methods, it does automatically place it as the optimal method. Instead, when reviewing the performance of Zero Tracking, it is noticeable that the overall worst-case performances, shown in Table 13, were better than even DE.

It overall performance offers the best stability for all three rate groups. Although it does not achieve the highest marks it performance cannot be overlooked.

In regards to implementation challenges, it was determined that the implementation of the compression methods had to be implemented using a variable length transmission interrupt. This was decided because the minimum compression ratios for the three methods did not seem favorable enough for the implementation of a fixed interrupt.

The problem of error propagation on the 1553 bus ended up being a lot more challenging than initially expected. This is due to the manner in which the compressed data is packaged prior to transmission. As stated in the report, the most vulnerable position in the message block occurs in the position word section. It is recommend that when packing the compressed data it is better to locate all the position words at the top of the transmission buffer. When the position word it placed in the various block segments, it can cause the positions to become indistinguishable because of the packing format. This is an area which will need further study.

The experimental data which has been gathered from this research forms a good basis for determining the initial feasibility of the application of these compression methods to the MIL-STD-1553 data bus. Further considerations must be made in regards to implementation challenges. It is quite possibility that there might be further challenges which might arise during the integration process to the 1553, but for the most part much of this has been addressed in this thesis.

APPENDICES

Appendix A

AKY-14 Assembly Code

Modified Run-Length Encoding Program

```
*ULTRA MYQ1PCK
 OPTIONS AYK14EMR SOURCE,ERROR(C),TAPE
      ORIG 000
      LK R5,0   . First case: 20 Hz, Input
      JLR R0,Q1PCK100
               JLR R0,RLCOMP
      JLR R0,RLUNCOM
DONE   J +$  . Infinite Loop


. . Q1PCK
.     ****************************************************************
.     **                                                            **
.     **  Q1PCK                                                      **
.     **  MC1 INTERCOMPUTER DATA PACKING                             **
.     **                                                            **
.     ****************************************************************
.     **                                                            **
.     **  THIS ROUTINE PACKS DATA WORDS INTO MESSAGE BUFFERS         **
.     **  SO THEY MAY BE SENT TO MC2. THE RATE AND TYPE OF           **
.     **  DATA BEING PACKED IS PROVIDED BY THE CALLING ROUTINE.      **
.     **  TYPES - 0=INPUT, 1=OUTPUT, 2=INTERNAL                      **
.     **  RATES - 0=20HZ, 1=10HZ, 2=5HZ, 3=1HZ                       **
.     **                                                            **
.     **  INVOCATION:                                               **
.     **    L   R5,------  . R5 SHOULD CONTAIN (4*TYPE) + RATE      **
.     **    JLR R4,Q1PCK                                            **
.     **                                                            **
.     **  REGISTERS USED:                                           **
.     **    R0 - *LINK ADDRESS                                      **
.     **    R1 - EMPTY                                              **
.     **    R2 - EMPTY                                              **
.     **    R3 - EMPTY                                              **
.     **    R4 - EMPTY                                              **
.     **    R5 - SEE INVOCATION                                     **
.     **    R6 - NUMBER OF LOCATIONS TO BE PACKED                   **
.     **    R7 - SOURCE ADDRESS LIST POINTERS                       **
.     **    R8 - DESTINATION ADDRESS LIST POINTERS                  **
.     **    R9 - ADDRESS OF DATA WORD                               **
.     **    R10 - DATA WORD                                         **
.     **    R11 - EMPTY                                             **
.     **    R12 - EMPTY                                             **
.     **    R13 - EMPTY                                             **
.     **    R14 - EMPTY                                             **
```

```
.      **   R15 - EMPTY                                                  **
.      **                                                                **
.      **  PARAMETERS PASSED:                                            **
.      **   R5 - SEE INVOCATION                                          **
.      **                                                                **
.      **  NOTES:                                                        **
.      **   NONE                                                         **
.      **                                                                **
.      ****************************************************************

. ********************************************************************
. * MYQ1PCK*
. * GET COUNT AND SOURCE/DESTINATION ADDRESS LIST POINTERS
. ********************************************************************

Q1PCK100
      L    R6,QSIZE1,R5   . PICK UP THE NUMBER OF LOCATIONS TO BE
                          . PACKED
      DROR  R6            . DECREMENT COUNT BY ONE
      L    R7,Q1ADR1,R5   . PICK UP THE ADDRESS OF THE TABLE OF
                          . ADDRESSES OF DATA TO BE PACKED
      L    R8,QBUFR1,R5   . PICK UP THE ADDRESS OF THE MESSAGE BUFFER TO BE USED

Q1PCK200
      LXI   R9,R7         . FETCH TARGET ADDRESS AND INCREMENT
                          . ADDRESS POINTER //Load and index by 1 (indirect)GO
      LI    R10,R9        . FETCH DATA //Load Indirect
      SXI   R10,R8         . STORE DATA INTO TARGET ADDRESS AND INCREMENT DATA
                          .POINTER //Store and index by 1 (indirect)

Q1PCK400
      XJ    R6,Q1PCK200   . DECREMENT COUNT AND LOOP //Index Jump

Q1PCK990
      LPR   R0            . RETURN AFTER PACKING IS COMPLETE


. ---------------------------------------------------------------
. Run Length Stuff here.
. ---------------------------------------------------------------
.      ************************************************************
.      **   This routine implements a lossless data compression  **
.      **   scheme.                                              **
.      **                                                        **
.      **  INVOCATION:                                           **
.      **    L   R5,------  . R5 SHOULD CONTAIN (4*TYPE) + RATE  **
.      **    JLR R4,Q1PCK   . (See invocation for Q1PCK)         **
.      **    JLR R4,ZTCOMP                                       **
.      **                                                        **
.      **  REGISTERS USED:                                       **
.      **    R0 - LINK ADDRESS                                   **
.      **    R1 - COUNTER FOR COMPRESSED DATA                    **
.      **    R2 -                                                **
.      **    R3 -                                                **
.      **    R5 - SEE INVOCATION FOR Q1PCK                       **
.      **    R6 - SIZE OF THE DATA TO BE COMPRESSED (source data)**
```

```
.     **   R7 - POINTER TO DATA BUFFER (QBUFR1)                    **
.     **   R8 - POINTER TO TEMPORARY BUFFER (TEMPDAT)              **
.     **   R9 - CURRENT ADDRESS STATUS BIT (1 or 0)                **
.     **   R10 - DATA (N)                                          **
.     **   R11 - DATA (N+1)                                        **
.     **   R12 - RUN TRACKING REG                                  **
.     **   R13 - POINTER TO TEMP BUFFER BIT POSITION ADDRESS       **
.     **   R14 -                                                   **
.     **   R15 -                                                   **
.     **                                                           **
.     **  PARAMETERS PASSED:                                       **
.     **   R5 - SEE INVOCATION                                     **
.     **                                                           **
.     **                                                           **
.     **  NOTES:                                                   **
.     **    NONE                                                   **
.     **                                                           **
.     ***************************************************************
.
.
RLCOMP  L R7,QBUFR1,R5     . LOAD DATA POINTER INTO R7
      L R8,TEMPDAT         . LOAD POINTER TO R8
            A R8,BITPOS    . BUFFER LOADED TO R8 (DATA WORDS)
            L R13,TEMPDAT  . LOAD POINTER TO TEMP BUFFER (POSITION WORDS)

            LK R1,1         . INIT COUNTER TO ONE
            LK R9,0        . INIT STATUS BIT TO ZERO
            LI R10,R7      . LOAD DATA FROM MEM TO R10
            SXI R10,R8      . STORE FIRST DATA WORD INTO TRANSMIT
BUFF/INCREMENT ADDR
            IROR R1         . INCREMENT COUNTER
            IROR R7        . INCREMENT DATA POINTER

CMPR        LI R11,R7      . LOAD DATA FROM MEM TO R11
            CR R10,R11     . COMPARE R11 TO R13
            JE EQU         . JUMP IF R10 AND R11 ARE EQUAL

NEQU        LK R9,0        . SET STATUS BIT TO ZERO
            J PCKTX        . JUMP TO PCKTX TO PACK INTO TEMPDAT

EQU         LK R9,32768    . SET BIT 15 TO ONE
            J ENCODE      . JUMP TO ENCODE BIT POSITION ADDR SECTION


. *******************************************************
. *This section of code stores the data words into the
. *transmission buffer. Successive word locations which are
. *unequal to previous locations are store into the
. *transmission buffer.
. *******************************************************
PCKTX       SXI R11,R8     . STORE DATA INTO TEMPDAT(DATA WORDS)/INCREMENT
ADDR
            J NEXT         . JUMP TO NEXT
```

```
. ********************************************************
. *This section of code encodes the Bit Position Words
. ********************************************************

ENCODE  LK R2,0              . ZERO FOR DIVISION
        LR R3,R1             . COPY COUNTER INTO R3
        DK R2,16             . DIVIDE R2 BY CONSTANT Ox16
        S R2,REMAIN          . STORE REMAINDER
        S R3,QUOTINT         . STORE QUOTIENT

        LR R3,R9             . STORE CONSTANT STATUS BIT FOR SHIFTING
        LRSR R3,R2           . SHIFT R3 (STATUS BIT) by R2

        L R13,TEMPDAT        . LOAD BIT POSITION WORD POINTER
        L R14,QUOTINT        . LOAD QUOTINT

        AR R13,R14           . ADD QOUTINT TO BIT POSITION WORD POINTER
        LI R4,R13            . LOAD CONTENTS OF BIT POSITION WORD INTO R4
        ORR R3,R4            .LOGIC OR R3(SHIFT STATUS BIT) & R4(POSITION WORD)


        SI R3,R13            . STORE R3 INTO R13(BIT POS WORD)

NEXT    LR R10,R11           . COPY DATA IN R11 TO R10
        IROR R7              . INCREMENT CURRENT DATA POINTER
        IROR R1              . INCREMENT COUNTER
        C R1,QSIZE1,R5       . CHECK COUNTER WITH BLOCK SIZE
        JNE CMPR             . LOOP AND COMPARE NEXT WORD LOCATIONS IF NOT EQU


. ************************************************************
. * This section of code copies the temporary data buffer
. *(which now holds the compressed data) back to the data buffer.
. ************************************************************
      L R8,TEMPDAT       . LOAD ADDR OF TEMPORARY BUFFER TO R8
      L R6,QSIZE1,R5     . LOAD THE SIZE OF THE DATA BUFFER TO R6
DONEST  L R7,QBUFR1,R5     . GET THE STARTING ADDRESS OF THE DATA BUFFER
      AK R1,8
      DROR R1
WLOOP2  LXI R10,R8         . GET THE NEXT DATA POINT FROM THE TEMPORARY BUFFER
      SXI R10,R7         . PUT IT INTO THE NEXT LOCATION IN THE DATA BUFFER
      XJ R1,WLOOP2       . DECREMENT THE COMPRESSED DATA COUNTER
DONE2   LPR R0             . RETURN FROM THE CALL

. ---------------------------------------------------------------------
. rl uncompress stuff here
. ---------------------------------------------------------------------
.       ************************************************************
.       **   This routine implements the uncompress for the run   **
.       **   length scheme.  It should be called after the        **
.       **   compressed data is transmitted over the bus and      **
.       **   BEFORE calling Q2UPK.                                **
.       **                                                        **
.       **  INVOCATION:                                           **
.       **    L   R5,------  . R5 SHOULD CONTAIN (4*TYPE) + RATE  **
.       **    JPR R4,ZTUNCOM                                      **
```

```
.     **                                                              **
.     **  REGISTERS USED:                                             **
.     **    R0 - LINK MAIN ADDRESS                                    **
.     **    R1 - LINK ADDRESS #2                                      **
.     **    R2 -                                                      **
.     **    R3 -                                                      **
.     **    R5 - SEE INVOCATION FOR Q1PCK                             **
.     **    R6 - SIZE OF THE DATA OF UNCOMPRESSED DATA                **
.     **    R7 - POINTER TO DATA BUFFER(data section)                 **
.     **    R8 - POINTER TO DATA BUFFER(address section)              **
.     **    R9 -  LOADED ADDRESS                                      **
.     **    R10 - LOADED DATA                                         **
.     **    R11 - POINTER TO TEMP BUFFFER                             **
.           **    R12 - SET BIT POSITION                              **
.     **    R13 -                                                     **
.     **                                                              **
.     **  PARAMETERS PASSED:                                          **
.     **    R5 - SEE INVOCATION                                       **
.     **                                                              **
.     **                                                              **
.     **  NOTES:                                                      **
.     **  NONE                                                        **
.     **                                                              **
.     **                                                              **
.     **************************************************************

RLUNCOM   L  R6,QSIZE1,R5    . LOAD SIZE OF UNCOMPRESSED DATA
          DROR  R6           . DECREMENT UNCOMPRESSED DATA COUNT
          L    R11,TEMPDAT   . LOAD POINTER FOR TEMP DATA

          L    R7,QBUFR1,R5  . LOAD POINTER TO DATA WORDS
          A    R8,BITPOS     . BUFFER LOADED TO R8 (DATA WORDS)

          L    R8,QBUFR1,R5  . LOAD POINTER TO POSITION WORD

          LI   R10,R7        . LOAD 1ST COMPRESSED DATA VALUE TO R10
          LI   R9,R8         . LOAD POSITION WORD VALUE TO R8


          SI R10,R11           . STORE 1ST COMPRESSED DATA VALUE TO TEMPDAT
          IROR R11             . INCREMENT TEMP DATA POINTER
          DROR R6              . DECREMENT COUNTER

          LK R12,0       . SET BIT STATUS TO ZERO
          J BIT2               . JUMP TO BIT2

. ********************************************************
. **Compare address bits
. ********************************************************
BIT0
               DROR R6                 . DECREMENT COUNTER
               CK R6,0                 . COMPARE R6 to Zero
               JE DONE

               LK R1,BIT1            . LOAD LINK ADDR TO R1
               CBR R9,15             . COMPARE BIT 15 TO ZERO
```

```
        JE STORE0           . IF ZERO JUMP TO STORE0
        J STORE1            . JUMP TO STORE1

BIT1
        DROR R6             . DECREMENT COUNTER
        CK R6,0             . COMPARE R6 to Zero
        JE DONE

        LK R1,BIT2          . LOAD LINK ADDR TO R1
        CBR R9,14           . COMPARE BIT 14 TO ZERO
        JE STORE0           . IF ZERO JUMP TO STORE0
        J STORE1            . JUMP TO STORE1

BIT2
        DROR R6             . DECREMENT COUNTER
        CK R6,0             . COMPARE R6 to Zero
        JE DONE

        LK R1,BIT3          . LOAD LINK ADDR TO R1
        CBR R9,13           . COMPARE BIT 13 TO ZERO
        JE STORE0           . IF ZERO JUMP TO STORE0
        J STORE1            . JUMP TO STORE1

BIT3
        DROR R6             . DECREMENT COUNTER
        CK R6,0             . COMPARE R6 to Zero
        JE DONE

        LK R1,BIT4          . LOAD LINK ADDR TO R1
        CBR R9,12           . COMPARE BIT 12 TO ZERO
        JE STORE0           . IF ZERO JUMP TO STORE0
        J STORE1            . JUMP TO STORE1

BIT4
        DROR R6             . DECREMENT COUNTER
        CK R6,0             . COMPARE R6 to Zero
        JE DONE

        LK R1,BIT5          . LOAD LINK ADDR TO R1
        CBR R9,11           . COMPARE BIT 11 TO ZERO
        JE STORE0           . IF ZERO JUMP TO STORE0
        J STORE1            . JUMP TO STORE1

BIT5
        DROR R6             . DECREMENT COUNTER
        CK R6,0             . COMPARE R6 to Zero
        JE DONE

        LK R1,BIT6          . LOAD LINK ADDR TO R1
        CBR R9,10           . COMPARE BIT 10 TO ZERO
        JE STORE0           . IF ZERO JUMP TO STORE0
        J STORE1            . JUMP TO STORE1

BIT6
        DROR R6             . DECREMENT COUNTER
        CK R6,0             . COMPARE R6 to Zero
```

```
        JE DONE

        LK R1,BIT7          . LOAD LINK ADDR TO R1
        CBR R9,9            . COMPARE BIT 9 TO ZERO
        JE STORE0           . IF ZERO JUMP TO STORE0
        J STORE1            . JUMP TO STORE1

BIT7
        DROR R6             . DECREMENT COUNTER
        CK R6,0             . COMPARE R6 to Zero
        JE DONE

        LK R1,BIT8          . LOAD LINK ADDR TO R1
        CBR R9,8            . COMPARE BIT 8 TO ZERO
        JE STORE0           . IF ZERO JUMP TO STORE0
        J STORE1            . JUMP TO STORE1

BIT8
        DROR R6             . DECREMENT COUNTER
        CK R6,0             . COMPARE R6 to Zero
        JE DONE

        LK R1,BIT9          . LOAD LINK ADDR TO R1
        CBR R9,7            . COMPARE BIT 7 TO ZERO
        JE STORE0           . IF ZERO JUMP TO STORE0
        J STORE1            . JUMP TO STORE1

BIT9
        DROR R6             . DECREMENT COUNTER
        CK R6,0             . COMPARE R6 to Zero
        JE DONE

        LK R1,BIT10         . LOAD LINK ADDR TO R1
        CBR R9,6            . COMPARE BIT 6 TO ZERO
        JE STORE0           . IF ZERO JUMP TO STORE0
        J STORE1            . JUMP TO STORE1

BIT10
        DROR R6             . DECREMENT COUNTER
        CK R6,0             . COMPARE R6 to Zero
        JE DONE

        LK R1,BIT11         . LOAD LINK ADDR TO R1
        CBR R9,5            . COMPARE BIT 5 TO ZERO
        JE STORE0           . IF ZERO JUMP TO STORE0
        J STORE1            . JUMP TO STORE1

BIT11
        DROR R6             . DECREMENT COUNTER
        CK R6,0             . COMPARE R6 to Zero
        JE DONE

        LK R1,BIT12         . LOAD LINK ADDR TO R1
        CBR R9,4            . COMPARE BIT 4 TO ZERO
        JE STORE0           . IF ZERO JUMP TO STORE0
        J STORE1            . JUMP TO STORE1
```

```
BIT12
            DROR R6             . DECREMENT COUNTER
            CK R6,0             . COMPARE R6 to Zero
            JE DONE

            LK R1,BIT13         . LOAD LINK ADDR TO R1
            CBR R9,3            . COMPARE BIT 3 TO ZERO
            JE STORE0           . IF ZERO JUMP TO STORE0
            J STORE1            . JUMP TO STORE1

BIT13
            DROR R6             . DECREMENT COUNTER
            CK R6,0             . COMPARE R6 to Zero
            JE DONE

            LK R1,BIT14         . LOAD LINK ADDR TO R1
            CBR R9,2            . COMPARE BIT 2 TO ZERO
            JE STORE0           . IF ZERO JUMP TO STORE0
            J STORE1            . JUMP TO STORE1

BIT14
            DROR R6             . DECREMENT COUNTER
            CK R6,0             . COMPARE R6 to Zero
            JE DONE

            LK R1,BIT15         . LOAD LINK ADDR TO R1
            CBR R9,1            . COMPARE BIT 1 TO ZERO
            JE STORE0           . IF ZERO JUMP TO STORE0
            J STORE1            . JUMP TO STORE1

BIT15
            DROR R6             . DECREMENT COUNTER
            CK R6,0             . COMPARE R6 to Zero
            JE DONE

            LK R1,CHK           . LOAD LINK ADDR TO R1
            CBR R9,0            . COMPARE BIT 0 TO ZERO
            JE STORE0           . IF ZERO JUMP TO STORE0
            J STORE1            . JUMP TO STORE1

CHK
            C R8,QBUFR1,8       . COMPARE ADDR TO LOCATION TO END
            JE DONE             . IF EQU JUMP TO DONE
            IROR R8             . INCREMENT POINTER FOR BIT POSITION
            LI R9,R8            . LOAD BIT POSITION WORD
            J BIT0

. ********************************************************
. **Store to Buffer
. ********************************************************

STORE0
            IROR R7             . INCREMENT DATA WORD POINTER
            LI R10,R7           . LOAD DATA WORD
            SI R10,R11          . STORE DATA WORD
```

```
            IROR R11            . INCREMENT TEMP BUFFER POINTER
            LPR R1              .RETURN TO LINK ADDR


STORE1
            SI R10,R11          . STORE DATA WORD
            IROR R11            . INCREMENT TEMP BUFFER POINTER
            LPR R1              . RETURN TO LINK ADDR



 . Set up data table
 .
 .
 .  Data Examples using q1pck
      ORIG 010000
QSIZE1 125  . Input 20 Hz
 92    . Input 10 Hz
 120   . Input 5 Hz
 130   . Input 1 Hz
 140   . Output 20 Hz
 150   . Output 10 Hz
 160   . Output 5 Hz
 170   . Output 1 Hz
 200   . Internal 20 Hz
 80    . Internal 10 Hz
 220   . Internal 5 Hz
 230   . Internal 1 Hz
Q1ADR1 04000  .Address locations:  Input 20 Hz
 04000  .Address locations:  Input 10 Hz
 04000  .Input 5
 04000  .Input 1
 04000  .Output 20
 04000  .Output 10
 04000  .Output 5
 04000  .Output 1
 04000  .Internal 20
 04000  .Internal 10
 04000  .Internal 5
 04000  .Address locations:  Internal 1 Hz
QBUFR1 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
```

```
 06000


. ***********************************************************
. **Variable Constants
. ***********************************************************
REMAIN  00000           .Remainder
QUOTINT 00000           .Quotient
TEMPDAT 07000
BITPOS  00008   .Indicates the total number of bit position words required

    ORIG 02000
 05000   .Table of addresses starts here at 02000+2000_offset=4000 =
 05001
 05002
 05003
 05004
 05005
 05006
 05007
 05010
 05011
 05012
 05013
 05014
 05015
 05016
 05017
 05020
 05021
 05022
 05023
 05024
 05025
 05026
 05027
 05030
 05031
 05032
 05033
 05034
 05035
 05036
 05037
 05040
 05041
 05042
 05043
 05044
 05045
 05046
 05047
 05050
 05051
 05052
 05053
```

05054
05055
05056
05057
05060
05061
05062
05063
05064
05065
05066
05067
05070
05071
05072
05073
05074
05075
05076
05077
05100
05101
05102
05103
05104
05105
05106
05107
05110
05111
05112
05113
05114
05115
05116
05117
05120
05121
05122
05123
05124
05125
05126
05127
05130
05131
05132
05133
05134
05135
05136
05137
05140
05141
05142
05143

```
05144
05145
05146
05147
05150
05151
05152
05153
05154
05155
05156
05157
05160
05161
05162
05163
05164
05165
05166
05167
05170
05171
05172
05173
05174
05175
05176
05177

.************************************
.* Data to be compressed
.************************************
      ORIG 03000  . Data is at addresses 3000+2000=5000
07777
07777
07777
01111
01111
01111
01111
00001
01111
01111
01111
07777
06600
06600
06600
06600
06600
06600
06600
07777
00001
00001
07777
```

07777
03300
03300
07777
00001
07777
07777
00550
00550
00550
07777
07777
07777
06600
06600
06600
00550
00550
07777
00001
00001
00001
00001
00001
00001
00001
07010
00001
07012
00001
00001
06015
06016
00001
00001
01021
00000
01023
00001
00001
00001
01027
00001
05031
05032
05033
05034
00001
00001
00001
05040
00001
05042
00043
00001
07045

00001
00001
00050
04051
00001
04053
04054
00001
00001
03057
02060
07777
07777
07777
01111
01111
01111
01111
00001
01111
01111
01111
07777
00001
00001
00001
00001
00001
00001
00001
00001
06071
00001
06074
06075
00001
00001
00001
05101
00001
00001
02104
02105
02106
00001
00001
00001
00001
00001
06114
06115
06116
00001
00001
00001
00001

```
03123
03124
00001
00001
03127
00001
00131
05132
00001
00001
00001
00001
07137
00000
07141
06144
06144
06144
     ORIG 07000
 END
*EOF
```

```
*ULTRA MYQ1PCK
 OPTIONS AYK14EMR SOURCE,ERROR(C),TAPE
      ORIG 000
      LK R5,0   . First case: 20 Hz, Input
                LK R15,8
                JLR R0,Q1PCK100
LOOPT1      JLR R0,ZERO1
      JLR R0,DUNCOM
                XJ R15,LOOPT1
                JLR R0,ZERO1
      JLR R0,DUNCOM
DONE    J +$  . Infinite Loop


. . Q1PCK
.      ****************************************************************
.      **                                                            **
.      **  Q1PCK                                                     **
.      **  MC1 INTERCOMPUTER DATA PACKING                            **
.      **                                                            **
.      ****************************************************************
.      **                                                            **
.      **  THIS ROUTINE PACKS DATA WORDS INTO MESSAGE BUFFERS        **
.      **  SO THEY MAY BE SENT TO MC2. THE RATE AND TYPE OF          **
.      **  DATA BEING PACKED IS PROVIDED BY THE CALLING ROUTINE.     **
.      **  TYPES - 0=INPUT, 1=OUTPUT, 2=INTERNAL                     **
.      **  RATES - 0=20HZ, 1=10HZ, 2=5HZ, 3=1HZ                      **
.      **                                                            **
.      **  INVOCATION:                                               **
.      **    L   R5,------  . R5 SHOULD CONTAIN (4*TYPE) + RATE      **
.      **    JLR R4,Q1PCK                                            **
.      **                                                            **
.      **  REGISTERS USED:                                           **
.      **    R0 - *LINK ADDRESS                                      **
.      **    R1 - EMPTY                                              **
.      **    R2 - EMPTY                                              **
.      **    R3 - EMPTY                                              **
.      **    R4 - EMPTY                                              **
.      **    R5 - SEE INVOCATION                                     **
.      **    R6 - NUMBER OF LOCATIONS TO BE PACKED                   **
.      **    R7 - SOURCE ADDRESS LIST POINTERS                       **
.      **    R8 - DESTINATION ADDRESS LIST POINTERS                  **
.      **    R9 - ADDRESS OF DATA WORD                               **
.      **    R10 - DATA WORD                                         **
.      **    R11 - EMPTY                                             **
.      **    R12 - EMPTY                                             **
.      **    R13 - EMPTY                                             **
.      **    R14 - EMPTY                                             **
.      **    R15 - EMPTY                                             **
.      **                                                            **
.      **  PARAMETERS PASSED:                                        **
.      **    R5 - SEE INVOCATION                                     **
```

```
.     **                                                              **
.     ** NOTES:                                                       **
.     **   NONE                                                       **
.     **                                                              **
.     ******************************************************************

. ******************************************************************
. * MYQ1PCK*
. * GET COUNT AND SOURCE/DESTINATION ADDRESS LIST POINTERS
. ******************************************************************

Q1PCK100
      L    R6,QSIZE1,R5  . PICK UP THE NUMBER OF LOCATIONS TO BE
                         . PACKED
      DROR  R6           . DECREMENT COUNT BY ONE
      L    R7,Q1ADR1,R5  . PICK UP THE ADDRESS OF THE TABLE OF
                          . ADDRESSES OF DATA TO BE PACKED
      L    R8,QBUFR1,R5  . PICK UP THE ADDRESS OF THE MESSAGE BUFFER TO BE USED

Q1PCK200
      LXI   R9,R7        . FETCH TARGET ADDRESS AND INCREMENT
                         . ADDRESS POINTER //Load and index by 1 (indirect)GO
      LI    R10,R9       . FETCH DATA //Load Indirect
      SXI   R10,R8       . STORE DATA INTO TARGET ADDRESS AND INCREMENT DATA
POINTER //Store and index by 1 (indirect)

Q1PCK400
      XJ    R6,Q1PCK200   . DECREMENT COUNT AND LOOP //Index Jump

Q1PCK990
      LPR   R0           . RETURN AFTER PACKING IS COMPLETE


. ------------------------------------------------------------------
. DIFFERENTIAL ENCODING STUFF HERE.
. ------------------------------------------------------------------
.     ************************************************************
.     **   This routine implements a lossless data compression    **
.     **   scheme.                                                **
.     **                                                          **
.     ** INVOCATION:                                             **
.     **    L   R5,------  . R5 SHOULD CONTAIN (4*TYPE) + RATE    **
.     **    JLR R4,Q1PCK   . (See invocation for Q1PCK)           **
.     **    JLR R4,ZTCOMP                                         **
.     **                                                          **
.     ** REGISTERS USED:                                         **
.     **    R0 - LINK ADDRESS                                     **
.     **    R1 - COUNTER for compressed data                      **
.     **    R2 -                                                  **
.     **    R3 -                                                  **
.     **    R5 - SEE INVOCATION FOR Q1PCK                         **
.     **    R6 - Size of the data to be compressed (source data)  **
.     **    R7 - Pointer to data buffer (QBUFR1)                  **
.     **    R8 - Pointer to temporary buffer (TEMPDAT)            **
.     **    R9 - (POINTER) Old Data Buffer                        **
.     **    R10 - New Data                                        **
```

```
.     **   R11 - Old Data                                              **
.     **   R12 -                                                       **
.     **   R13 - Pointer to temporary buffer (address section)         **
.     **   R14 -                                                       **
.     **   R15 -                                                       **
.     **                                                               **
.     ** PARAMETERS PASSED:                                           **
.     **   R5 - SEE INVOCATION                                         **
.     **                                                               **
.     **                                                               **
.     ** NOTES:                                                       **
.     **   NONE                                                        **
.     **                                                               **
.     ***************************************************************
.
.
ZERO1   L R6,TEMPDAT            .LOAD ADDR  INTO REG
            L R1,QSIZE1,R5      .LOAD THE SIZE OF THE DATA BUFFER TO R6
            DROR R1             .DECREMENT COUNTER

ZEROPK      SZI   R6            .STORE ZERO INTO ADDR
            IROR R6             .INCREMENT REGISTER ADDR
            XJ R1,ZEROPK

. ********************************************************
. **
. ********************************************************

DCOMP   L R7,QBUFR1,R5          . POINTER TO INPUT DATA LOADED TO R7
     L R8,TEMPDAT               . POINTER TO TEMPORARY
            A R8,BITPOS          . BUFFER LOADED TO R8 (DATA WORDS)
            L R13,TEMPDAT       . POINTER TO TEMP BUFFER POSITION WORDS)
            L R9,OLDBUF          . POINTER TO OLD BUFFER

            LK R1,0             . INIT COUNTER TO ONE

. ********************************************************
. **COMPARE OLD BUFFER WITH NEW BUFFER
. ********************************************************

CMPR        LI R10,R7       . LOAD DATA FROM NEW BUFFER TO R10
     LI R11,R9              . LOAD DATA FROM OLD BUFFER TO R11
            CR R10,R11      . COMPARE R10 TO R11
            JE EQU          . JUMP TO EQU IF EQUAL

NEQU        J PCKTMB        . IF NOT EQUAL JUMP TO DATA PACKING SEQ

EQU         IROR R9         . INCREMENT OLD BUFFER POINTER
            IROR R7         . INCREMENT NEW BUFFER POINTER
            IROR R1         . INCREMENT COUNTER
            C R1,QSIZE1,R5      . CHECK COUNTER WITH BLOCK SIZE
            JNE CMPR        . LOOP IF NOT EQUAL
            J DONEST
```

```
. ********************************************************
. **PACK THE TRANSMIT BUFFER/ADDRESSING ENCODING
. ********************************************************

PCKTMB    SXI R10,R8     . STORE INTO TRANSMIT BUFFER AND INCREMENT POINTER

ENCODE    LK R2,0        . LOAD CONSTANT ZERO FOR DIVISION PROCESS
          LR R3,R1       . COPY COUNTER TO R3
          DK R2,16       . DIVIDE R2(COUNTER) BY CONSTANT Ox16
          S R2,REMAIN    . STORE REMAINDER
          S R3,QUOTINT   . STORE QUOTIENT

          LK R3,32768    . STORE CONSTANT 1 (BIT 15) FOR SHIFTING
          LRSR R3,R2     . SHIFT R3 by R2

          L R13,TEMPDAT  . LOAD BIT POSITION POINTER TO R13
          L R12,QUOTINT  . LOAD QUOTIENT TO R12
          AR R13,R12     . ADD QUOTIENT TO BIT POSITION POINTER
          LI R4,R13      . LOAD R6 FROM MEM TO R4
          ORR R3,R4      . LOGIC OR R3 AND R4

          SI R3,R13      . STORE BIT POSITION WORD TO TEM BUFFER
          J EQU          . CONTINUE COMPARING OLD/NEW DATA


. ************************************************************
. * THIS SECTION OF CODE COPIES THE TEMP. DATA BUFFER
. * (WHICH NOW HOLDS THE COMPRESSED DATA) BACK TO THE DATA BUFFER.
. ************************************************************
DONEST  L R8,TEMPDAT     . LOAD ADDRESS OF TEMPORARY BUFFER TO R8
     L R6,QSIZE1,R5      . LOAD THE SIZE OF THE DATA BUFFER TO R6
     L R7,QBUFR1,R5      . GET THE STARTING ADDRESS OF THE DATA BUFFER
          DROR R6
WLOOP2  LXI R10,R8     . GET THE NEXT DATA POINT FROM THE TEMPORARY BUFFER
     SXI R10,R7        . PUT IT INTO THE NEXT LOCATION IN THE DATA BUFFER
     XJ R6,WLOOP2      . DECREMENT THE COMPRESSED DATA COUNTER
     LPR R0            . RETURN FROM THE CALL


. ---------------------------------------------------------------
. rl uncompress stuff here
. ---------------------------------------------------------------
.      ************************************************************
.      **  This routine implements the uncompress for the run    **
.      **  length scheme.  It should be called after the         **
.      **  compressed data is transmitted over the bus and       **
.      **  BEFORE calling Q2UPK.                                 **
.      **                                                        **
.      **  INVOCATION:                                           **
.      **    L   R5,------  . R5 SHOULD CONTAIN (4*TYPE) + RATE  **
.      **    JPR R4,ZTUNCOM                                      **
.      **                                                        **
.      **  REGISTERS USED:                                       **
.      **    R0 - Link Main address                              **
.      **    R1 - Link Address 2/**                              **
.      **    R2 -                                                **
```

```
.      **   R3 -                                                    **
.      **   R5 - SEE INVOCATION FOR Q1PCK                           **
.      **   R6 - Size of the data of uncompressed data              **
.      **   R7 - Pointer to data buffer (data section)              **
.      **   R8 - Pointer to data buffer (address section)           **
.      **   R9 -  Loaded Address                                    **
.      **   R10 - Loaded Data                                       **
.      **   R11 - Pointer to temporary buffer                       **
.               **   R12 - Set Bit Position                         **
.      **   R13 - Pointer old buffer                                **
.      **   R14 - Loaded Old Data                                   **
.      **   R15 -                                                   **
.      **                                                           **
.      **  PARAMETERS PASSED:                                       **
.      **   R5 - SEE INVOCATION                                     **
.      **                                                           **
.      **                                                           **
.      **  NOTES:                                                   **
.      **  NONE                                                     **
.      **                                                           **
.      ***********************************************************
.

DUNCOM   L     R6,QSIZE1,R5  . LOAD SIZE OF UNCOMPRESSED DATA
         L     R11,TEMPDAT   . LOAD POINTER FOR TEMP BUFFER

         L     R7,QBUFR1,R5  . LOAD POINTER TO DATA WORDS
         A     R7,BITPOS     . (SKIP BIT POSITION WORDS)
         LI    R10,R7        . Load Data of temporary buffer to R8

        L     R8,QBUFR1,R5  . LOAD POINTER TO BIT POSITION WORDS
        LI    R9,R8         . LOAD BIT POSITION WORD

        L     R13,OLDBUF    . POINTER TO OLD BUFFER


. ********************************************************
. **COMPARE ADDRESS BITS
. ********************************************************

BIT0
                DROR R6                 . DECREMENT COUNTER
                CK R6,0                 . COMPARE R6 to Zero
                JE DONE3

                LK R1,BIT1      . LOAD LINK ADDR TO R1
                CBR R9,15       . COMPARE BIT 15 TO ZERO
                JNE STORE       . JUMP TO STORE (CHANGE IN DATA)
                J INCRE          . INCREMENT TO NEXT DATA WORD (NO CHANGE)

BIT1
                DROR R6                 . DECREMENT COUNTER
                CK R6,0                 . COMPARE R6 to Zero
                JE DONE3

                LK R1,BIT2              . LOAD LINK ADDR TO R1
```

```
        CBR R9,14       . COMPARE BIT 14 TO ZERO
        JNE STORE       . JUMP TO STORE (CHANGE IN DATA)
        J INCRE         . INCREMENT TO NEXT DATA WORD (NO CHANGE)

BIT2
        DROR R6           . DECREMENT COUNTER
        CK R6,0           . COMPARE R6 to Zero
        JE DONE3

        LK R1,BIT3      . LOAD LINK ADDR TO R1
        CBR R9,13       . COMPARE BIT 13 TO ZERO
        JNE STORE       . JUMP TO STORE (CHANGE IN DATA)
        J INCRE         . INCREMENT TO NEXT DATA WORD (NO CHANGE)

BIT3
        DROR R6           . DECREMENT COUNTER
        CK R6,0           . COMPARE R6 to Zero
        JE DONE3

        LK R1,BIT4      . LOAD LINK ADDR TO R1
        CBR R9,12       . COMPARE BIT 12 TO ZERO
        JNE STORE       . JUMP TO STORE (CHANGE IN DATA)
        J INCRE         . INCREMENT TO NEXT DATA WORD (NO CHANGE)

BIT4
        DROR R6           . DECREMENT COUNTER
        CK R6,0           . COMPARE R6 to Zero
        JE DONE3

        LK R1,BIT5      . LOAD LINK ADDR TO R1
        CBR R9,11       . COMPARE BIT 11 TO ZERO
        JNE STORE       . JUMP TO STORE (CHANGE IN DATA)
        J INCRE         . INCREMENT TO NEXT DATA WORD (NO CHANGE)

BIT5
        DROR R6           . DECREMENT COUNTER
        CK R6,0           . COMPARE R6 to Zero
        JE DONE3

        LK R1,BIT6      . LOAD LINK ADDR TO R1
        CBR R9,10       . COMPARE BIT 10 TO ZERO
        JNE STORE       . JUMP TO STORE (CHANGE IN DATA)
        J INCRE         . INCREMENT TO NEXT DATA WORD (NO CHANGE)

BIT6
        DROR R6           . DECREMENT COUNTER
        CK R6,0           . COMPARE R6 to Zero
        JE DONE3

        LK R1,BIT7      . LOAD LINK ADDR TO R1
        CBR R9,9        . COMPARE BIT 9 TO ZERO
        JNE STORE       . JUMP TO STORE (CHANGE IN DATA)
        J INCRE         . INCREMENT TO NEXT DATA WORD (NO CHANGE)

BIT7
        DROR R6           . DECREMENT COUNTER
```

```
        CK R6,0             . COMPARE R6 to Zero
        JE DONE3

        LK R1,BIT8          . LOAD LINK ADDR TO R1
        CBR R9,8            . COMPARE BIT 8 TO ZERO
        JNE STORE           . JUMP TO STORE (CHANGE IN DATA)
        J INCRE             . INCREMENT TO NEXT DATA WORD (NO CHANGE)

BIT8
        DROR R6             . DECREMENT COUNTER
        CK R6,0             . COMPARE R6 to Zero
        JE DONE3

        LK R1,BIT9          . LOAD LINK ADDR TO R1
        CBR R9,7            . COMPARE BIT 7 TO ZERO
        JNE STORE           . JUMP TO STORE (CHANGE IN DATA)
        J INCRE             . INCREMENT TO NEXT DATA WORD (NO CHANGE)

BIT9
        DROR R6             . DECREMENT COUNTER
        CK R6,0             . COMPARE R6 to Zero
        JE DONE3

        LK R1,BIT10         . LOAD LINK ADDR TO R1
        CBR R9,6            . COMPARE BIT 6 TO ZERO
        JNE STORE           . JUMP TO STORE (CHANGE IN DATA)
        J INCRE             . INCREMENT TO NEXT DATA WORD (NO CHANGE)

BIT10
        DROR R6             . DECREMENT COUNTER
        CK R6,0             . COMPARE R6 to Zero
        JE DONE3

        LK R1,BIT11         . LOAD LINK ADDR TO R1
        CBR R9,5            . COMPARE BIT 5 TO ZERO
        JNE STORE           . JUMP TO STORE (CHANGE IN DATA)
        J INCRE             . INCREMENT TO NEXT DATA WORD (NO CHANGE)

BIT11
        DROR R6             . DECREMENT COUNTER
        CK R6,0             . COMPARE R6 to Zero
        JE DONE3

        LK R1,BIT12         . LOAD LINK ADDR TO R1
        CBR R9,4            . COMPARE BIT 4 TO ZERO
        JNE STORE           . JUMP TO STORE (CHANGE IN DATA)
        J INCRE             . INCREMENT TO NEXT DATA WORD (NO CHANGE)

BIT12
        DROR R6             . DECREMENT COUNTER
        CK R6,0             . COMPARE R6 to Zero
        JE DONE3

        LK R1,BIT13         . LOAD LINK ADDR TO R1
        CBR R9,3            . COMPARE BIT 3 TO ZERO
        JNE STORE           . JUMP TO STORE (CHANGE IN DATA)
```

```
            J INCRE                 . INCREMENT TO NEXT DATA WORD (NO CHANGE)

BIT13
            DROR R6                   . DECREMENT COUNTER
            CK R6,0                   . COMPARE R6 to Zero
            JE DONE3

            LK R1,BIT14             . LOAD LINK ADDR TO R1
            CBR R9,2                . COMPARE BIT 2 TO ZERO
            JNE STORE               . JUMP TO STORE (CHANGE IN DATA)
            J INCRE                 . INCREMENT TO NEXT DATA WORD (NO CHANGE)

BIT14
            DROR R6                   . DECREMENT COUNTER
            CK R6,0                   . COMPARE R6 to Zero
            JE DONE3

            LK R1,BIT15             . LOAD LINK ADDR TO R1
            CBR R9,1                . COMPARE BIT 1 TO ZERO
            JNE STORE               . JUMP TO STORE (CHANGE IN DATA)
            J INCRE                 . INCREMENT TO NEXT DATA WORD (NO CHANGE)

BIT15
            DROR R6                  . DECREMENT COUNTER
            CK R6,0                 . COMPARE R6 to Zero
            JE DONE3

            LK R1,CHK               . LOAD LINK ADDR TO R1
            CBR R9,0                . COMPARE BIT 0 TO ZERO
            JNE STORE               . JUMP TO STORE (CHANGE IN DATA)
            J INCRE                 . INCREMENT TO NEXT DATA WORD (NO CHANGE)

CHK
            IROR R8                 . INCREMENT BIT POSITION POINTER
            LI R9,R8                . LOAD BIT POSITION WORD
            J BIT0                   . LOOP TO BIT0

. ********************************************************
. **Store to Buffer
. ********************************************************

STORE       SI R10,R11              . STORE
            IROR R11                . INCREMENT POINTER TO TEMP BUFFER

            IROR R13                . INCREMENT POINTER TO OLD BUFF
            LI R14,R13              . LOAD NEXT OLD BUFF WORD INTO R14

            IROR R7                   . INCREMENT DATA WORD POINTER
      LI R10,R7                     . LOAD NEXT DATA WORD LOCATION INTO R10
            LPR R1                   . RETURN TO LINK ADDR

INCRE       LI R14,R13              . LOAD NEXT OLD BUFF WORD INTO R14
            SI R14,R11              . STORE
            IROR R11                . INCREMENT POINTER TO TEMP BUFFER

            IROR R13                . INCREMENT POINTER TO OLD BUFF
```

```
        LPR R1                  . RETURN TO LINK ADDR


. ************************************************************
. * This section of code copies the temporary data buffer
. * (which now holds the compressed data) back to the data buffer.
. ************************************************************
DONE3   L R8,OLDBUF     . LOAD ADDR OF TEMP BUFFER TO R8
        L R6,QSIZE1,R5  . LOAD THE SIZE OF THE DATA BUFFER TO R6
        L R7,QBUFR1,R5  . GET THE STARTING ADDRESS OF THE DATA BUFFER
        DROR R6
LOOP5   LXI R10,R8      . GET THE NEXT DATA POINT FROM THE TEMPORARY BUFFER
        SXI R10,R7      . PUT IT INTO THE NEXT LOCATION IN THE DATA BUFFER
        XJ R6,LOOP5     . DECREMENT THE COMPRESSED DATA COUNTER

        L R2,OLDBUF     . POINTER TO OLD BUFFER
        CK R2,2560
        JE CHG

        LK R2,2560
        S R2,OLDBUF
        LPR R0

CHG     LK R2,2880
        S R2,OLDBUF
        LPR R0


 . Set up data table
 .
 .
 .   Data Examples using q1pck
        ORIG 010000
QSIZE1 100  . Input 20 Hz
 92   . Input 10 Hz
 120   . Input 5 Hz
 130   . Input 1 Hz
 140    . Output 20 Hz
 150   . Output 10 Hz
 160   . Output 5 Hz
 170   . Output 1 Hz
 200   . Internal 20 Hz
 80   . Internal 10 Hz
 220   . Internal 5 Hz
 230   . Internal 1 Hz
Q1ADR1 04000  .Address locations:  Input 20 Hz
 04000  .Address locations:  Input 10 Hz
 04000  .Input 5
 04000  .Input 1
 04000  .Output 20
 04000  .Output 10
 04000  .Output 5
 04000  .Output 1
 04000  .Internal 20
 04000  .Internal 10
 04000  .Internal 5
```

```
 04000 .Address locations:  Internal 1 Hz
QBUFR1 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000
 06000


. ***********************************************************
. **Variable Constants
. ***********************************************************
REMAIN  00000            .Remainder
QUOTINT 00000            .Quotient
OLDBUF  05500   .Old Buffer
BITPOS  00010   .Size of bit position section

TEMPDAT  07000

     ORIG 02000
 05000    .Table of addresses starts here at 02000+2000_offset=4000 =
 05001
 05002
 05003
 05004
 05005
 05006
 05007
 05010
 05011
 05012
 05013
 05014
 05015
 05016
 05017
 05020
 05021
 05022
 05023
 05024
 05025
 05026
 05027
 05030
```

05031
05032
05033
05034
05035
05036
05037
05040
05041
05042
05043
05044
05045
05046
05047
05050
05051
05052
05053
05054
05055
05056
05057
05060
05061
05062
05063
05064
05065
05066
05067
05070
05071
05072
05073
05074
05075
05076
05077
05100
05101
05102
05103
05104
05105
05106
05107
05110
05111
05112
05113
05114
05115
05116
05117
05120

```
05121
05122
05123
05124
05125
05126
05127
05130
05131
05132
05133
05134
05135
05136
05137
05140
05141
05142
05143
05144



    ORIG 02500
05500    .Table of addresses starts here at 02000+2000_offset=4000 =
05501
05502
05503
05504
05505
05506
05507
05510
05511
05512
05513
05514
05515
05516
05517
05520
05521
05522
05523
05524
05525
05526
05527
05530
05531
05532
05533
05534
05535
05536
05537
```

05540
05541
05542
05543
05544
05545
05546
05547
05550
05551
05552
05553
05554
05555
05556
05557
05560
05561
05562
05563
05564
05565
05566
05567
05570
05571
05572
05573
05574
05575
05576
05577
05600
05601
05602
05603
05604
05605
05606
05607
05610
05611
05612
05613
05614
05615
05616
05617
05620
05621
05622
05623
05624
05625
05626
05627

```
05630
05631
05632
05633
05634
05635
05636
05637
05640
05641
05642
05643
05644


.************************************
.* Data to be compressed
.************************************
      ORIG 03000  . Data is at addresses 3000+2000=5000
07777
01111
05555
00001
00001
00001
04444
01111
00001
00001
00001
01444
00001
00001
07666
07111
00001
00001
01111
00001
06611
07771
00001
00001
00001
07775
07774
07776
00001
00001
00001
07777
00001
00001
07777
07777
07777
```

01111
01111
01111
01111
00001
01111
01111
01111
07777
06600
06600
06600
06600
06600
06600
06600
07777
00001
00001
07777
07777
03300
03300
07777
00001
07777
07777
00550
00550
00550
07777
07777
07777
06600
06600
06600
00550
00550
07777
00001
00001
00001
00001
00001
00001
00001
07010
00001
07012
00001
00001
06015
06016
00001
00001
01021

00001
01023
00001
00001
00001
01027
01027
05031
05032
05033
05034
00001
00001
00001
05040
00000
05042
00043
00001
07045
00001
00001
00050
04051
00001
04053
04054
00001
00001
03057
02060
07777
07777
07777
01111
01111
01111
01111
00001
01111
01111
01111
07777
00001
00001
00001
00001
00001
00001
00001
00001
00001
06071
00001
06074
06075
00001

```
00001
00001
05101
00001
00001
02104
02105
02106
00001
00001
00001
00001
00001
06114
06115
06116
00001
00001
00001
00001
03123
03124
00001
00001
03127
00001
00131
05132
00001
00001
00001
00001
07137
00001
07141
06144
06144
06144

/***********************************
*
***********************************/
                ORIG 03500
07777
07777
00001
00001
07777
00001
07777
07777
00001
00001
00001
07777
07777
```

00001
00001
00001
00001
07777
00001
00001
07777
07777
00001
00001
07777
07777
07777
00001
00001
00001
00001
07777
07777
00001
00001
00001
00001
07777
01111
01111
01111
01111
00001
01111
01111
01111
07777
06600
06600
06600
06600
06600
06600
06600
07777
00001
00001
07777
07777
03300
00001
00001
00001
00001
00001
00001
06071
00001
06074

```
 06075
 00001
 00001
 00001
 05101
 00001
 00001
 02104
 02105
 02106
 00001
 00001
 00001
 00001
 00001
 06114
 06115
 06116
 00001
 00001
 00001
 00001
 03123
 03124
 00001
 00001
 03127
 00001
 00131
 05132
 00001
 00001
 00001
 00001
 07137
 00001
 07141

 ORIG 07000
 END
*EOF
```

Appendix B

Run-Length Statistics

These run-length statistics were acquired from previous research in [4].

Table B.1: 20 Hz MC1 Dump 1 Run-length Statistics

Run Length	1	2	3	4	5	6	7	8	9	10
# of Runs	10605	1217	1697	644	820	215	611	244	1	238
Run Length	11	12	14	15	17	19	20	22	29	31
# of Runs	213	424	213	1	213	212	423	1	213	2
Run Length	32	39	41	49	54	55	195			
# of Runs	1	213	1	213	1	1	1			

Table B.2: 20 Hz MC1 Dump 2 Run-length Statistics

Run Length	1	2	3	4	5	6	7	8	10	11
# of Runs	10583	1294	1414	1008	404	202	606	201	202	284
Run Length	12	13	14	17	20	23	28	39	49	
# of Runs	202	202	524	202	202	202	201	202	202	

Table B.3: 20 Hz MC2 Dump 1 Run-length Statistics

Run Length	1	2	3	4	5	8	9	14	15	17
# of Runs	27057	3200	3086	429	383	408	214	173	4	210
Run Length	20	23	25							
# of Runs	40	214	1							

Table B.4: 20 Hz MC2 Dump 2 Run-length Statistics

Run Length	1	2	3	4	5	6	7	8	9	13
# of Runs	29146	3644	1818	807	286	202	84	202	202	118
Run Length	17	19								
# of Runs	202	202								

Table B.5: 10 Hz MC1 Dump 1 Run-length Statistics

Run Length	1	2	3	4	5	6	8	9	11	12
# of Runs	3449	633	213	106	204	2	107	213	204	1
Run Length	17	18	27	30	35	42	48	56	88	118
# of Runs	115	106	53	1	107	1	1	106	54	50
Run Length	121	122								
# of Runs	2	0								

Table B.6: 10 Hz MC1 Dump 2 Run-length Statistics

Run Length	1	2	3	4	5	7	8	9	11	17
# of Runs	3630	605	202	202	101	100	101	200	101	101
Run Length	18	32	56	122						
# of Runs	101	101	101	101						

Table B.7: 10 Hz MC2 Dump 1 Run-length Statistics

Run Length	1	2	3	4	5	6	7	8	9	10
# of Runs	6726	1709	321	214	426	214	106	107	428	204
Run Length	11	12	13	19	22	27	29	30	35	56
# of Runs	98	214	107	107	9	104	105	3	2	107

Table B.8: 10 Hz MC2 Dump 2 Run-length Statistics

Run Length	1	2	3	4	5	6	7	8	9	10
# of Runs	7367	1717	404	503	301	101	200	201	404	101
Run Length	11	12	13	19	22	56				
# of Runs	201	101	2	303	101	101				

Table B.9: 5 Hz MC1 Dump 1 Run-length Statistics

Run Length	1	3	5	9	13	16	20
# of Runs	914	53	54	53	1	54	53

Table B.10: 5 Hz MC1 Dump 2 Run-length Statistics

Run Length	1	5	13	20	32
# of Runs	649	50	1	49	49

Table B.11: 5 Hz MC1 Dump 2 Run-length Statistics

Run Length	1	2	4	5	6	7	14	15	16	43
# of Runs	2359	160	54	53	43	10	54	54	54	54

Table B.12: 5 Hz MC1 Dump 2 Run-length Statistics

Run Length	1	2	3	4	6	7	9	14	15	45
# of Runs	2090	148	50	81	49	31	49	19	100	50

Appendix C

MATLAB M-files

ENCODE_ZERO.m

```
function [tx_buffer stat_txbuff addr_words] = ENCODE_ZERO(new)
%% Initializes address bits for zero tracking
addr_words = ceil(length(new)/16);
tx_addr = zeros(addr_words,16);
tx_buff = [];

%% FIND ZERO LOCATIONS AND SET ADDR BITS (SETS THE TX_ADDR BITS)
for i=1:1:length(new)
    if(new(i) == 0) %compare old and new data, if different...
        row = ceil(i/16);
        bit = mod(i,16);
        if(bit == 0) bit = 16; end
        tx_addr(row,bit) = 1;
    end
end

%% PACKS THE TX_BUFFER
count = 1;              %indicates amount of space in TX Buffer

for i=1:1:length(new)
    if(new(i) ~= 0)
         tx_buff =[tx_buff ; new(i)];
         count = count + 1;
    end
end

%converts the tx_address bits to decimal and puts at top of tx_buffer
tx_address = zeros(length(tx_addr(:,1)),1);
count = 1;
for i = 1:length(tx_addr(:,1))
    for j = 16*i-15:16*i
        if(tx_addr(i,count) == 1)
            tx_address(i) = 2^(16*i-j) + tx_address(i);
        end
        count = count + 1;
        if(count > 16) count = 1; end
    end
end
tx_buffer = [tx_buff];


%% Statistiscs
new_stat =  length(tx_buff) + length(tx_address);
```

```
orig_stat = length(new);
stat_txbuff = orig_stat/new_stat;          % compression ratio
```

ENCODE_RUNMOD.m

```
function [tx_buffer stat_txbuff addr_words] = ENCODE_RUNMOD(new)
%% Initializes address bits for run-length tracking
addr_words = ceil(length(new)/16);
block_set = ceil(length(new)/32); % defines number of block sets

tx_addr   = zeros(2,16);
% tx_data   = zeros(32,1);
tx_data = [];
tx_buffer = [];
tx_b = [];

set_buff = zeros(32,1);


%% SET RUN LENGTH ADDRESS BITS
temp_bit = 1;
tx_addr(1,2) = temp_bit;
orig_stat = 0;              %origninal compression
new_stat = 0;
count = 1;

for j=1:29:length(new)

    if((j+29) < length(new))
        set_buff = new(j:j+29);
    else
        set_buff = new(j:end);
    end

    orig_stat = orig_stat + length(set_buff);

    for i=1:(length(set_buff)-1)
       if(i <= 14)
           row = 1;
           bit = mod((i+1),16) + 1;
       else
           row = 2;
           bit = mod((i+1),16) + 1;
       end

       if(set_buff(i) ~= set_buff(i+1))
            temp_bit = not(temp_bit);
            count = count + 1;
       end
       tx_addr(row,bit) = temp_bit;
    end
```

```
%Check if data should be compressed(Compress Status Bit)

if(((count+2) >= length(set_buff)))
   comp_bit = 1;
   tx_addr(1,1) = comp_bit;
else
   comp_bit = 0;
   tx_addr(1,1) = comp_bit;
end

% PACKS THE TX_BUFFER
%packs the tx buffer with the address bits that were set above

% i = 1; j = 1;
if(tx_addr(1,1) == 0)
    count = 1;              %indicates amount of space in TX Buffer
    word = 1;               %indicates position in buffer
    tx_data(count,1) = set_buff(word);
    for i=1:(length(set_buff)-1)
       word = word + 1;
       if(set_buff(i) ~= set_buff(i+1))
            count = count + 1;
            tx_data(count,1) = set_buff(word);
       end
    end

    new_stat = new_stat + count + 2; % data plus address words
compression statistics
else
    tx_addr = tx_addr(1,:);
    tx_data = set_buff;
    new_stat = new_stat + count + 1; % data plus address words
compression statistics
end



%converts the tx_address bits to decimal and puts at top of tx_buffer
tx_address = zeros(length(tx_addr(:,1)),1);
count = 1;
for i = 1:length(tx_addr(:,1))
    for j = 16*i-15:16*i
        if(tx_addr(i,count) == 1)
            tx_address(i) = 2^(16*i-j) + tx_address(i);
        end
        count = count + 1;
        if(count > 16) count = 1; end
    end
end

tx_b = [tx_address; tx_data];
tx_buffer = [tx_buffer ; tx_b];

set_buff = [];
tx_addr   = zeros(2,16);
```

```
tx_data   = [];
tx_b = [];

end

stat_txbuff = orig_stat/new_stat;          % Compression Ratio
```

ENCODE_DIFFMOD.m

```
function [tx_buffer stat_txbuff addr_words plot_changes] =
ENCODE_DIFFMOD(old, new, Nbuff)
%% INITIALIZING NEW OLD AND OVFLOW DATA BUFFERS
addr_words = ceil(length(old)/16);
tx_buffer_size = Nbuff;
tx_size  = tx_buffer_size - addr_words; %size of tx_buff - address
tx_addr = zeros(addr_words,16);
% tx_buff = zeros(tx_size,1);
tx_buff = [];

%% Percent Change Statistics (For Plots)
plot_changes = zeros(length(new),1);
for i=1:1:length(new)
    if(old(i) ~= new(i)) %compare old and new data, if different...
        plot_changes(i)=i;
    end
end

%% COMPARE DIFFERENCES OF NEW AND OLD BUFFS AND SETS ADDR BITS
for i=1:1:length(new)
    if(old(i) ~= new(i)) %compare old and new data
        row = ceil(i/16);
        bit = mod(i,16);
        if(bit == 0) bit = 16; end
        tx_addr(row,bit) = 1;
    end
end

%% PACKS THE TX_BUFFER
i = 1; j = 1;
count = 1;              %indicates amount of space in TX Buffer
word = 1;               %indicates position in buffer

while(count <= tx_size && i <= length(tx_addr(:,1)))
    if(tx_addr(i,j) == 1)
         tx_buff(count,1) = new(word);
         count = count + 1;
    end

    word = word + 1;
    j = j + 1;
    if(j > 16) i = i + 1; j = 1; end
end
```

```
%converts the tx_address bits to decimal and puts at top of tx_buffer
tx_address = zeros(length(tx_addr(:,1)),1);
count = 1;
for i = 1:length(tx_addr(:,1))
    for j = 16*i-15:16*i
        if(tx_addr(i,count) == 1)
            tx_address(i) = 2^(16*i-j) + tx_address(i);
        end
        count = count + 1;
        if(count > 16) count = 1; end
    end
end
% tx_buffer = [tx_address; tx_buff];
tx_buffer = [tx_buff];


%% Statistiscs
new_stat =  length(tx_buff) + length(tx_address);
orig_stat = length(new);
stat_txbuff = orig_stat/new_stat;          % Compression Ratio
```

LIST OF REFERENCES

[1] Aging Avionics in Military Aircraft. Washington, D.C.: National Academy Press, 2001.

[2] V.K. Madisetti, Y. Jung, M.H. Khan, J.W. Kim, T. Finnessy, "Reengineering Legacy Embedded System," *IEEE Design and Testing of Computers*, pp. 38-47, 1999.

[3] R. Duren, "Options for Upgrading Legacy Avionics Systems," *AIAA Journal of Aerospace Computing, Information, and Communication*, vol. 3, no. 6, pp. 251-259, 2006.

[4] B.O. Weston, "Data Compression Application to the MIL-STD-1553 Avionics Data Bus", M.S. Thesis, Baylor University, Waco, TX, USA, 2005

[5] G. Motta, J. Gustafson and S. Chen, "Differential Compression of Executable Code," *2007 Data Compression Conference*, 2007.

[6] G.M. Lundy, P.H. Christensen, "Specification of the MIL-Standard 1553 protocol using systems of communicating machines," Military Communications Conference, vol. 3, pp.1049-1053, 1990.

[7] *MIL-STD-1553 Tutorial*, Condor Engineering Inc., Santa Barbara, CA, 2000.

[8] C.R. Spitzer, *The Avionics Handbook.* Boca Raton, FL: CRC Press, 2000, 1-1: 1-25.

[9] E. C. Gangl, "Evolution from analog to digital integration in aircraft avionics - a time of transition," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 42, no.3, pp.1163-1170, July 2006.

[10] S.R. Mackey, L.M. Meredith, "Software Migration and Reengineering: A Pilot Project in Reengineering," *Journal of Systems and Software*, vol.30, pp. 137-150, 1995.

[11] T. Shepard, M. Gagne, "A model of the F18 mission computer software for pre-run-time scheduling," *IEEE 10th Int'l Conf. Distributed Computing Systems*, pp.62-69, May 1990.

[12] R.A. Schuh, "An Overview of the 1553 Bus with Testing and Simulation Considerations," *IEEE Instrumentation and Measurement Technology Conference*, pp. 22-25, 1998.

[13] J.R. Murdock, J.R. Koenig, "Open systems avionics network to replace MIL-STD-1553," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 16, no. 8, pp. 15-19, Aug 2001.

[14] S.N. Griedman, "A multiprotocol multiplexed superhybrid terminal," *Aerospace and Electronics Conference*, *Proceedings of the IEEE National*, vol.1, pp.157-162, May 1990.

[15] R. Duren, M.W. Thompson, "Application of Data Compression to the MIL-STD-1553 Data Bus" *IEEE Aerospace Conference*, 2008.

[16] D. Salomon, *Data Compression: The Complete Reference.* New York, NY: Springer-Verlag New York Inc., 2004, 1:40.