Linköping University | Department of Computer and Information Science
Bachelor Thesis, 16 hp | Högskoleingenjör i datateknik
Spring term 2017 | LIU-IDA/LITH-EX-G--17/076--SE

Conversion and Analysis of Telemetric Data from the CCSDS Standard

Simon Ahlgren
Daniel Aini

Supervisor: Sebastian Sundqvist
Examinator: Petru Eles

Linköpings universitet
SE-581 83 Linköping
013-28 10 00, www.liu.se

Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© Simon Ahlgren, Daniel Aini

Abstract

When communicating with spacecrafts, the international standard is to use the protocols defined by CCSDS. In this study, the Space Packet Protocol from CCSDS is converted to the Digital Recording Standard used in aviation. The goal of the study is to find out in what way such a conversion can be made, as well as analyzing the efficiency of different packing methods for the Digital Recording Standard. An application is developed in order to perform the conversion, and the performance of said application is profiled using different packet sizes. In the end the results are evaluated and an optimal packet size is found in terms of runtime and memory usage. In the end we conclude that a packet size of 2^{16} bytes is best when prioritizing speed, and a packet size of 2^{19} bytes is best when prioritizing memory.

Acknowledgement

We would like to thank Instrument Control Sweden for providing us with this thesis. We also want to specifically thank Sebastian Sundqvist from ICS for helping us throughout this study and finally Professor Petru Eles for approving the study.

Daniel Aini
Simon Ahlgren

Table of Contents

List of Abbreviations

RCC	Range Commanders Council
IRIG	Inter Range Instrumentation Group
CCSDS	Consultive Committee for Space Data Systems
ICS	Instrument Control Sweden
LDP	Logical Data Path
APID	Application Process Identifier
IPTS	Intra Packet Time Source
RTC	Relative Time Counter
PCM	Pulse Code Modulation
RMM	Removable Memory Module
TMATS	Telemetry Attributes Transfer Standard
SRCC	Setup Record Configuration Change
CSV	Comma Separated Value
OTOC	One-To-One Conversion
FSC	Full Size Conversion

1. Introduction

The different conditions on earth and space has given birth to separate protocols and methods used to communicate in these environments. The Range Commanders Council (RCC) created a standard called IRIG-106 Chapter 10 which catered to digital recording applications [1]. This standard could not and still can not be used for space communication since it lacks the necessary security and reliability that space telemetry requires. It is however widely used on earth. A space protocol needs to be able to handle the large latencies and eventual disruptions that comes with deep space communication.

The Consultative Committee for Space Data Systems (CCSDS) was created as a collaboration between major space agencies across the globe. This cooperation came as a solution to the lack of an international standard. The protocols used in space mission varied immensely since they targeted different environments. They had to be configured specifically for each project which made it costly to design and maintain. This was recognised in 1982 and nations all over the world came together to create CCSDS. The organisation has published over 300 books and has a total of eleven member agencies to this day.

Instrument Control Sweden (ICS) is a company that specializes in displaying telemetric data. Their own developed software, Netview, can read and display telemetric data according to the IRIG-106 Chapter 10 standard. They want to improve the functionality of Netview and expand into space territorium. The software is however limited by the restraints put on the Chapter 10 standard. In order to handle data coming in from space, ICS needs an application that translates CCSDS packets to Chapter 10 files. Said conversion will handle Pulse Code Modulation (PCM) data. This thesis will focus on writing such an application.

Lastly, the application is evaluated using the ANTS performance profiler. The profiler measures the runtime of the application and provides detailed performance statistics about individual functions. These results will be used to figure out which packet size allows the conversion algorithm to perform better.

1.1 Motivation

Instrument Control Sweden (ICS) can currently handle Chapter 10 files. ICS want to be able to handle CCSDS packets as well to be able to also view the data coming in from spacecrafts in addition to aircrafts. The company wants the ability to view real time data coming in from spacecrafts. In order to do so, the software needs to be able to convert the data in a quick manner which is why the speed of the application will be examined.

1.2 Purpose

The focus of this thesis is to develop a software that translates CCSDS packets to Chapter 10 files. The optimal way of creating Chapter 10 files will be explored, with the metric being performance and memory usage.

1.3 Research Question

- What packet size proves to be optimal in terms of runtime and memory usage when converting CCSDS telemetry data to IRIG-106 Chapter 10 Digital Recording Standard?

1.4 Scope

The CCSDS conversion will not handle every data packet available since the Digital Recording Standard (Chapter 10) is overwhelmingly big to be included in its entirety for this project. This thesis will instead focus on converting CCSDS packets to Chapter 10 PCM packet format. This thesis will not focus on establishing a space data link (connection) between the spacecraft and the ground. The CCSDS packets will be created beforehand and injected into the application.

2. Theory

This chapter gives insight into the Irig-106 Chapter 10 standard and the CCSDS Space Packet protocol.

2.1 Space Packet Protocol

The Space Packet Protocol defines a way to transfer data between earth and space. This also includes communications between onboard relay systems in space. It accomplishes this by the help of a *Logical Data Path* [2]. The LDP is the path between the source and the destination(s). The source and destinations are end systems, i.e devices connected to the edge of the network. In order for the data to travel through space, it gets sent through onboard relay points and subnetworks. The LDP can of course be reconfigured as well. Since a spacecraft or a ground end system can have multiple LDPs, a way to easily identify each of them is needed. This is done with a *Path Id*.

This *Path Id* consists of an application process id (APID) and an APID Qualifier, which is optional. The APID acts as the main identifier. It singles out the LDP that the packet can take. This identifier is present in the packet primary header, which figure 1 describes.

The Space Packet protocol is defined in *CCSDS 133.0-B-1* [2]. The data that this protocol handles is the Space Packet, which is part of the network layer. It will be referred to as CCSDS packet. The CCSDS packet is divided into two parts: a packet primary header and a packet data field [3]. The packet primary header contains information about the packet in general. It includes four major fields, some of which can be broken down even further. The data contained in the primary header is described in figure 1.

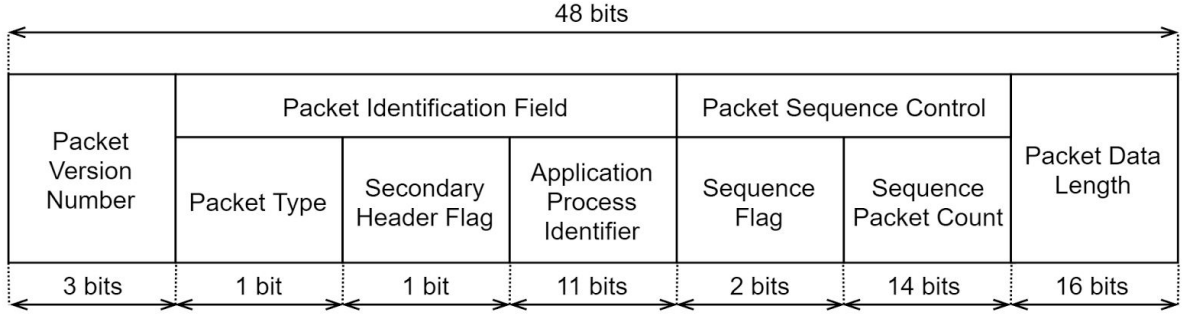

Figure 1: Format of the CCSDS packet header.

1. Packet Version Number. The packet version number identifies the data unit of the protocol. It is three bits and is usually set to all zeros

2. Packet Identification Field. Packet identification concerns bits 3-15 and is done in three steps.
    a. Packet Type. A bit deciding if the packet relates to telemetry or telecommand.
    b. Secondary Header Flag. A single bit that acts as a flag. If set, a secondary header exists.
    c. Application Process Identifier. As described above, the APID provides a way to identify the LDP. There are eleven bits in the APID. In case of an idle packet, the APID should be set to "all ones".

3. Packet Sequence Control. The Packet Sequence Control handles the sequence flag and the sequence packet count. Involves bits 16-31.
    a. Sequence Flag. Two bits that determine whether the data is part of a larger set or if it is independent.
    b. Sequence Packet Count: 14 bits that provide order to the packets being sent by an application. This is useful since they may arrive out of order.

4. Packet Data Length. Bits 32-47 reveals the packed data length. It should be the length expressed in octets subtracted by one.

What follows the primary header is the packet data field, see figure 2. Within the packet data field lies a possible secondary header and the user data field, which is the actual data that the application wants to transfer. This data could be anything ranging from basic arithmetic calculations to images or even files. This is up to the application. The secondary header is mandatory if there exists no user data. The two components that create the packet data field, vary in length. When the packet is ready for transmission, it is put inside a transfer frame and transported through the LDP [4].

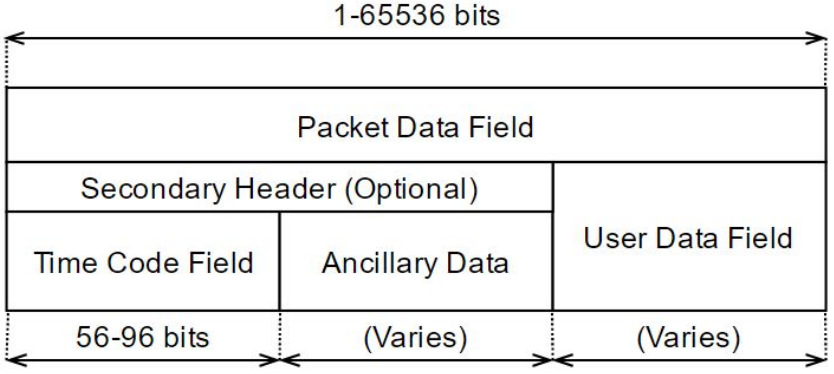

Figure 2: Format of the CCSDS Packet Data Field.

1. Secondary Header. The secondary header may provide even more details about the packet. It can contain two fields. The time code field and ancillary data.
    a. Time Code Field. This field contains the timestamp of the packet as well as metadata describing the format of the timestamp.
    b. Ancillary Data. This optional field contains custom data decided by the user in advance.

2. User Data Field. The data being sent is placed in this field. If the application sends no data, this field will be absent but the secondary header must be present.

2.2 Time Code Format

The time code, that may be present in the secondary header, supplies the user with a timestamp. There are a few different ways to format the timestamp and this needs to be predetermined by the corporation using the protocol [5]. There are two fields in the time code that are of significance. They are the preamble field and the time specification field.

The preamble is eight bits and can be seen as a tiny header. It is optional to have the preamble but since it gives necessary information regarding which format the time specification uses, it is included more often than not. There are four defined time code formats available.

- CCSDS Unsegmented Time Code
- CCSDS Day Segmented Time Code
- CCSDS Calendar Segmented Time Code
- CCSDS Ascii Calendar Segmented Time Code

There are three bits in the preamble that decide which of these formats the time specification follows. The extra combinations that are usable from these three bits are there to provide the selection of custom modified time codes. If the most significant bit in the preamble is set, there should follow an extension of eight more bits that contain additional metadata for the time specification. The final four bits are based on the time code used and give extra details that are vital for the specific time code in question.

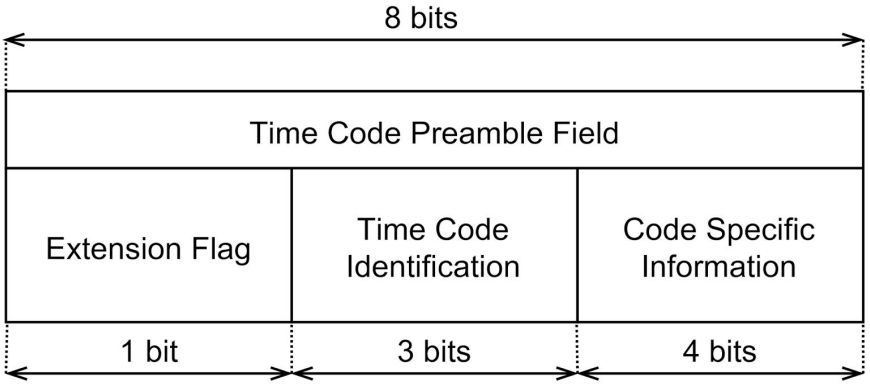

Figure 3: General format of the Time Code Preamble Field in a CCSDS packet.

As mentioned, the time specification can follow different structures based on the information from the preamble.

2.2.2 CCSDS Day Segmented Time Code

The Day Segmented Time Code has two mandatory segments: day and time of day in ms, see figure 4. These segments are binary counters which complement each other. The day format can be either a 16 bit or a 24 bit segment. It is dependent on bit five in the preamble. The time of day in ms consists of 32 bits and is self explanatory. The user also has the choice to include an additional segment that can specify the time even further with the use of fractional time units. It has a size specified by bit 6-7 in the preamble.

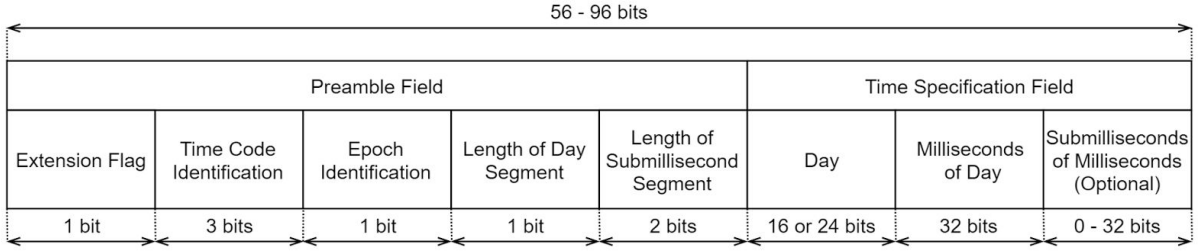

Figure 4: Format of the Timecode Field using a CCSDS Day Segmented Time Code.

2.2.3 CCSDS Calendar Segmented Time Code

The calendar segmented format divides the time code into either six or five different segments depending on the variation, see figure 5. These segments are normally eight bits (with the exception of year and day of year) and they act as binary counters. There exist two different variations explained in the protocol. Which one to use is up to the user. It is once again the preamble that provides this information. They have at least four segments in common.

The day of month/month of year variation gives a more modern representation of time according to the gregorian calendar. It has the month of year and the day of month segment, both of which are eight bits in size.

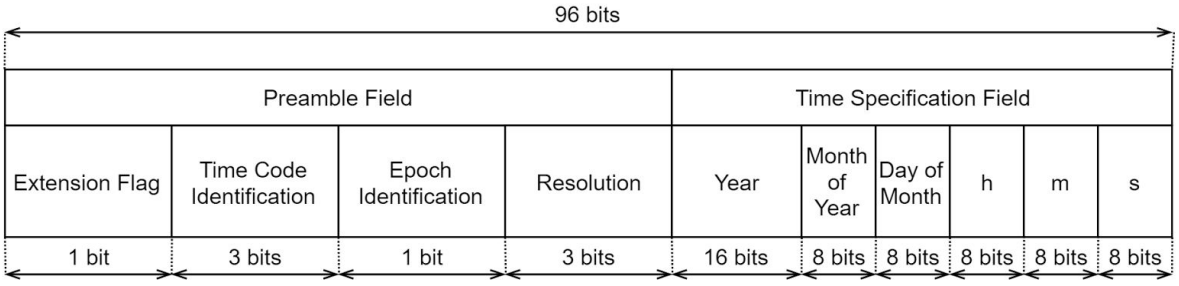

Figure 5: Format of the Timecode Field using a CCSDS Calendar Day of Month Segmented Time Code.

The day of year variation discards the segments introduced in the day of month/month of year variation and instead adapts to a simpler time representation, see figure 6. It has a day of year segment that represents a decimal between 0-365. In order to achieve this the segment is set to 16 bits.

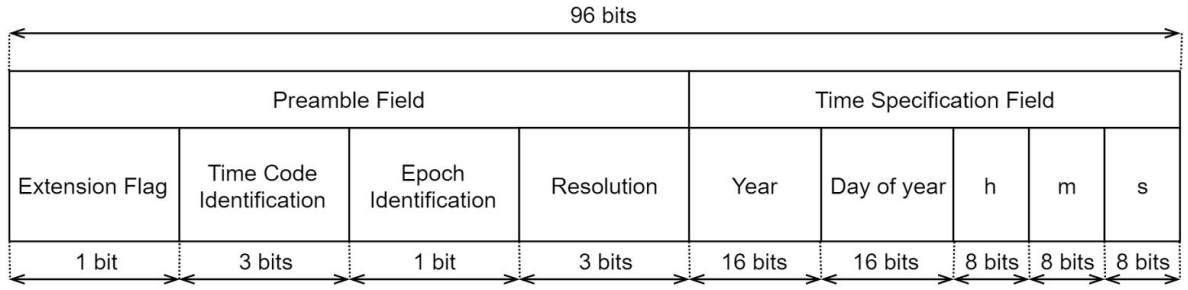

Figure 6: Format of the Timecode field using a CCSDS Day of Year Segmented Time Code.

2.3 IRIG-106 Chapter 10

All Chapter 10 packets are divided into three different mandatory parts: The packet header, packet body and packet trailer [1]. The packet header contains metadata describing the data in the rest of the packet. This header has a constant length of 192 bits (24 bytes) and is always formatted the same way for every type of data. The packet body however has a varied length and includes metadata specific to the type of data stored in the body.

The packet trailer contains filler bits and occasionally a data checksum. The filler bits are added in order to ensure that the entire packet is always 16 or 32 bit aligned. The data checksum is the sum of all bits in the datafield. This is used to ensure that the bits have not been modified during transmission due to bit errors. Before the packet body there may also be an optional secondary header. This header contains timestamps as well as a checksum for the secondary header. In figure 7 the packet header format is described bit by bit.

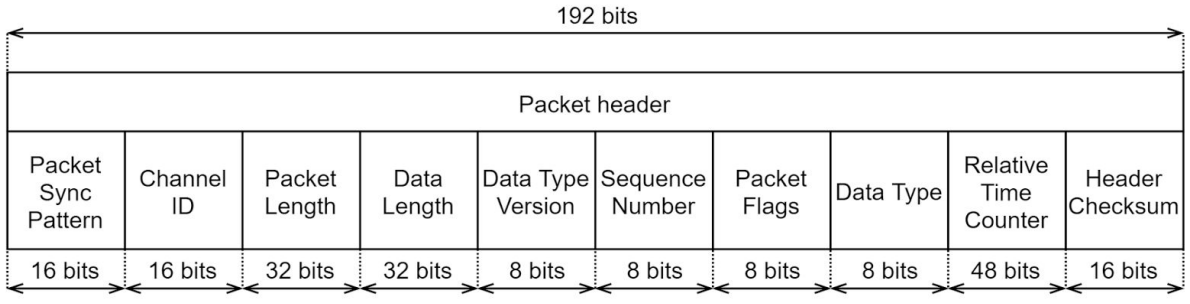

Figure 7: Format of the IRIG-106 Chapter 10 Packet Header.

2.3.1 Packet Header

1. Packet Sync Pattern. This is a constant value placed in the first bytes of the packet header. It is used to identify new packets and shall always be set to 0xEB25.

2. Channel ID. This value is used to identify which channel the packet originally came from. This information can be used to derive which data type is stored in the packet body.

3. Packet Length. This value represents the length of the packet in bytes, including the packet header and trailer. Since the packet is always 16 bit aligned, this value shall always be a multiple of four.

4. Data Length. This value represents the length of the packet body. This includes the channel specific word, eventual intra-packets, as well as the data itself.

5. Data Type Version. This value contains a bit pattern representing the release version of IRIG-106 which introduced the data type. These are the bit patterns currently supported:
    a. 0x00 = Reserved
    b. 0x01 = RCC 106-04 (Initial Release)
    c. 0x02 = RCC 106-05
    d. 0x03 = RCC 106-07
    e. 0x04 = RCC 106-09
    f. 0x05 = RCC 106-11

6. Sequence Number. This number identifies the order in which packets have been sent through a specific channel. The sequence number does not necessarily start at zero and is incremented with each packet sent through the respective channel.

7. Packet Flags. This byte contains different flags describing the format and content of the packet, as well as error identification.
    a. Bit 7. Indicates whether or not a secondary header is present in the packet
    b. Bit 6. Indicates the Intra-Packet Time source (IPTS)
    c. Bit 5. This bit is set if a RTC sync error is found.
    d. Bit 4. This bit is set if a data overflow error is found.
    e. Bit 3-2. Indicates the format of the secondary header, if present.
    f. Bit 1-0. Indicates the existence and eventual format of the data checksum.

8. Data Type. This value contains a bit pattern representing the type of the data stored in the packet body. A detailed list of data types can be found in table 10-10 page 24 of the *Digital Recording Standard* [1].

9. Relative Time Counter (RTC). This counter represents the time that has passed since the start of the transmission in ticks.

10. Header Checksum. This value contains the sum of all bytes in the packet header, except the checksum itself.

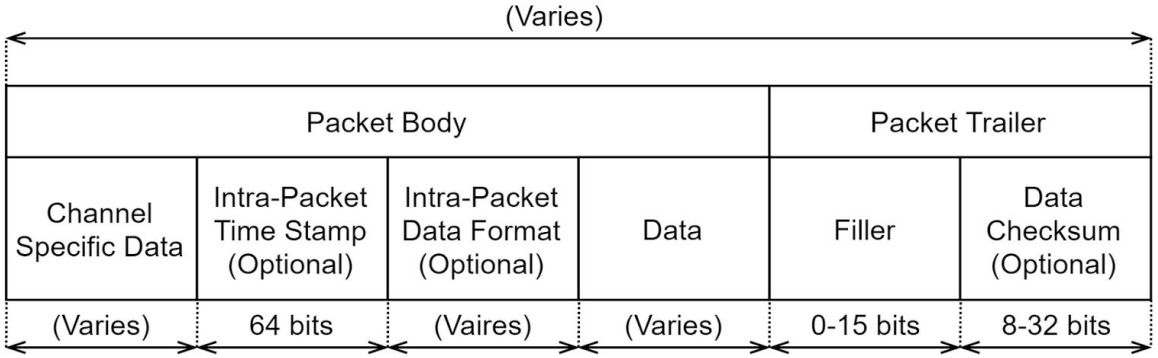

Figure 8: Format of the IRIG-106 Chapter 10 Packet Body and Trailer.

2.3.2 Packet Body

1. Channel Specific Data. This field contains information specific to the data type specified in the packet header. Its content and length varies depending on the given type but all types must have a respective channel specific data.

2. Intra-Packet Time Stamp. This is an optional field containing a timestamp for the packet. The format of the time stamp is given by the *Packet Flags* in the packet header.

3. Intra-Packet Data Format. This is an optional field containing metadata concerning the format of the following databits. Size varies depending on the type of data.

4. Data. The actual data of the packet. Format and size of the data varies depending on the type defined in the header.

2.3.3 Packet Trailer

1. Filler. This value is simply filler bits set to either all 0x00 or all 0xFF. These bits are used to make sure that the packet length is 16 bit aligned i.e divisible with 4.

2. Data Checksum. An optional checksum used for detecting bit errors in the data field. The existence and size of this checksum is defined by the *Packet Flags* in the packet header.

2.3.4 Pulse Code Modulation Standards

Pulse code Modulation (PCM) is an encoding method used to digitally represent analog signals. The amplitude of the signal is sampled at a high rate and stored as a binary number. The size of the numbers is determined by the word length used in the encoding. If, for example, a 16 bit word length is used, the binary number can vary between 0-15. The amount of samples read per second is determined by the bitrate of the encoding. These are the most important variables to consider when PCM is used. Bit rates higher than 10 megabits per second and word lengths exceeding 32 bits are considered advanced types of PCM [6].

PCM can also be used to represent serial binary signals, in which case several different methods may be used for the representation.

- Non Return to Zero Level (NRZ-L). This is the standard method for representing binary serial signals. If the sampled amplitude is 1 then the respective stored value is 1. Otherwise it is 0.
- Non Return to Zero Mark (NRZ-M). This method measures change in the signal. If the sampled amplitude has changed from 0 to 1, then the respective stored value will be 1. If no change has occurred the value will be set to 0.
- Non Return to Zero Space (NRZ-S). This method is the opposite of NRZ-M. If there is no change in level the respective stored value will be 1. otherwise the value will be set to 0.

2.3.5 PCM Packet Format

The header and trailer of a PCM packet follow the same format as any other packet. The difference lies in the body of the packet. The first 32 bits of a PCM packet body consists of a PCM Channel Specific Word [6]. This field contains information regarding the content and format of the data stored in the packet body. It is a mandatory field for all PCM packets. What follows the channel specific word is the data itself.

2.3.5.1 Major and Minor Frames

All data in a PCM packet is divided into major and minor frames, as illustrated in figure 9. These frames used to structure the data and make it more organized. The first data word of each minor frame contains the synchronization information for the minor frame. Whenever this synchronization word is found during data extraction, it signals the start of a new minor frame. In other words, a minor frame consists of all data between two different synchronization words. The total size of each minor frame shall not exceed 16384 bits or 2048 words.

A major frame consists of multiple minor frames. The length of a major frame is derived from the sum of the lengths of all minor frames contained within. The data stored within the frames can also be transmitted and stored in several different modes.

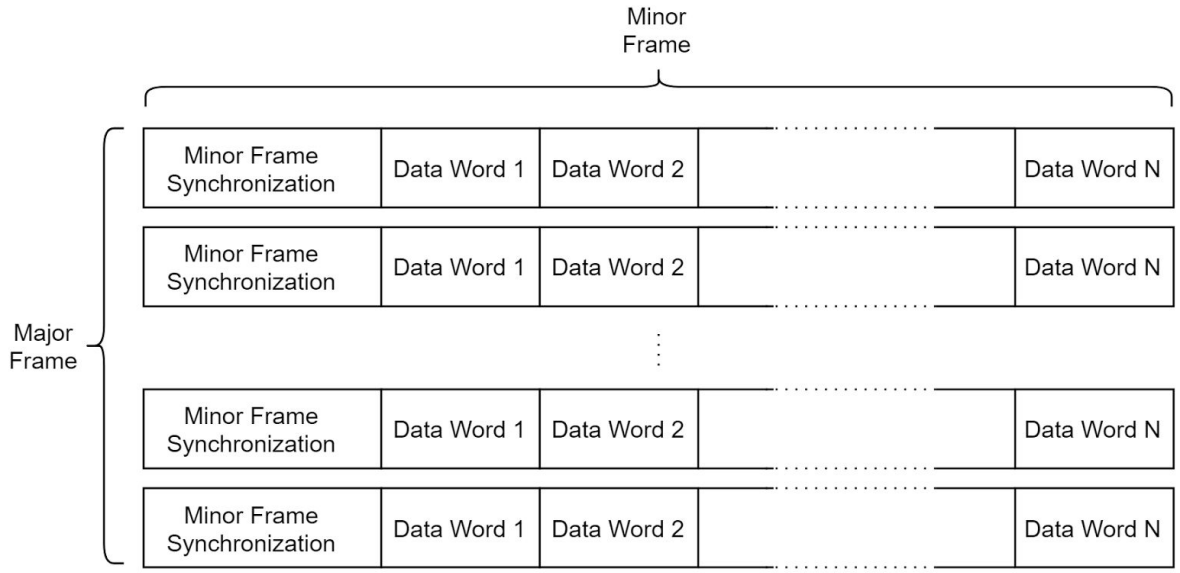

Figure 9: Minor and Major Frame format in a IRIG-106 Chapter 10 PCM Packet.

- 16 / 32 Bit Alignment. In addition to one of the modes described below, the data can be either 16 bit aligned or 32 bit aligned. When the data is in 16 or 32 bit alignment it is divided in 16 or 32 bit words respectively. This means that in the end the entire data field must be evenly divisible by either 16 or 32. This makes it easier to differ the frames from one another.

- Unpacked Mode. When the data is in unpacked mode, each word in the datafield must be exactly 16 or 32 bits long. If the data stored in each word is too short, filler bits must be added in order to pad the word.

- Packed Mode. When the data is in packed mode it is not necessary to pad each word in the data field, but some filler bits may have to be added at the end of the field to ensure that the field is either 16 or 32 bit aligned.

- Throughput Mode. When the data is in throughput mode it is not synchronized with the frames at all. The data is simply stored in the same format in which it was read.

2.3.5.2 PCM Channel Specific Data

The PCM Channel Specific Data is a field containing various metadata relating to the format of the PCM data. It has a total size of 32 bits, as illustrated in figure 10.

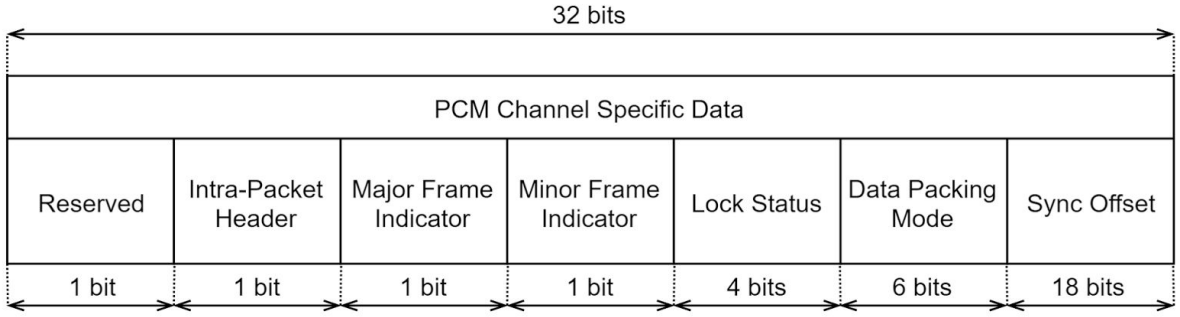

Figure 10: Format of the Channel Specific Data for a IRIG-106 Chapter 10 PCM Packet.

1. Reserved. The first bit of the Channel Specific Data is reserved for future use.

2. Intra-Packet Header. This value indicates whether the packet uses Intra-Packet Timestamps. If this value is set, timestamps must be inserted between minor frames in the data field. This option is mandatory when using packed or unpacked mode.

3. Major Frame Indicator. This value indicates whether the first word in the following data field is a major frame.

4. Minor Frame Indicator. This value indicates whether the first word in the following data field is a minor frame.

5. Lock Status. This field indicates if the frame is currently "locked". If the frame is locked it means that all data stored in the frames is completely intact and without disruptions. Lock status is not applicable when using throughput mode.

6. Data Packing Mode. This value indicates which data packing mode is used. The different options for data packing are described in section 2.3.5.1

7. Sync Offset. This value represents the offset of the first data word in the major frame. This value is only applicable for unpacked mode.

2.3.6 Time Packet Format

In the IRIG-106 Digital Recording standard, time packets must be transmitted in conjunction with the data packets. Time packets must be sent at least once every second and the first packet after the initial setup record must be a time packet [1]. Time packets follow the same general format as other data type packets. The only difference is that the data stored in the body, which in this case is timestamps. The main purpose of these time packets is to provide additional structure to the transmission, since the data packets themselves do not necessarily contain timestamps.

Just like the various data packets, the first part of the time packet body contains time channel specific data. The time channel specific data is a 32 bit field with information pertaining the format and source of the time data stored in the packet body, see figure 11.

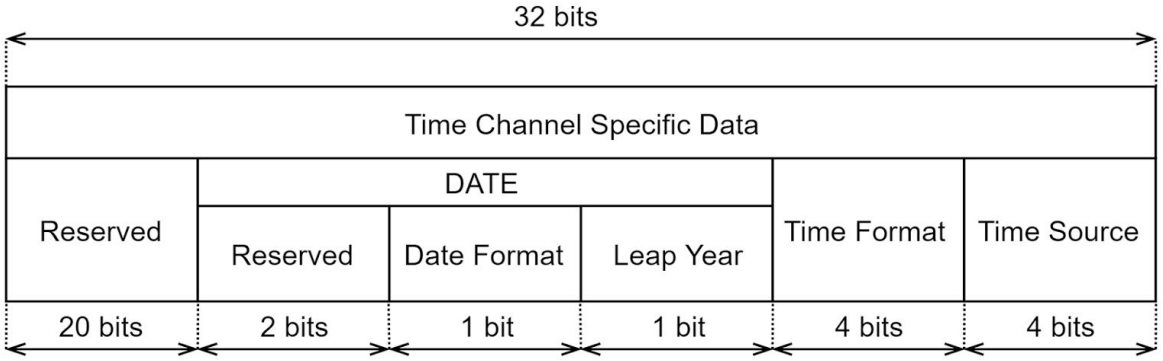

Figure 11: Format of the Channel Specific Data for a IRIG-106 Chapter 10 Time Packet.

1. Reserved. The first 20 bits of this field are reserved for future use.

2. Date Format (DATE). This 4 bit value represents the format of the timestamp
    a. Bit 3-2. These bits are reserved for future use.
    b. Bit 1. This bit defines the date format of the timestamp. If set to 0 it will use IRIG Day format. If set to 1 it will use Month and Year format.
    c. Bit 0. This bit indicates whether or not it is a leap year.

3. Time Format. This value indicates the format of the time data packets. It can be set to one of the following values:
    a. 0x0 = IRIG-B
    b. 0x1 = IRIG-A
    c. 0x2 = IRIG-G
    d. 0x3 = Real-Time Clock
    e. 0x4 = UTC Time from GPS
    f. 0x5 = Native GPS Time
    g. 0x6 - 0xE = Reserved
    h. 0xF = None / Invalid

4. Time Source. This value indicates the Source of the timestamp. It can be set to one of the following values:
    a. 0x0 = Internal clock from the recorder.
    b. 0x1 = External clock not from the recorder.
    c. 0x2 = Internal clock from a Removable Memory Module (RMM).
    d. 0x3 - 0xE = Reserved.
    e. 0xF = None

2.3.7 Setup Record Format

The setup record is always the very first packet in each transmission [1]. It is a computer generated packet containing all information regarding the transmission. It can for example define all the different channels used in the transmission. This information is useful at the receiving end when developing software to interpret and display the data. The setup record is very different compared to the usual packets. The information is written in ASCII or XML instead of binary encoded data. This text follows a standard called the Telemetry Attributes Transfer Standard (TMATS) [7]. The TMATS record can contain a wide variety of options that define the transmission. Just like all other packets the setup record body contains channel specific data, see figure 12.

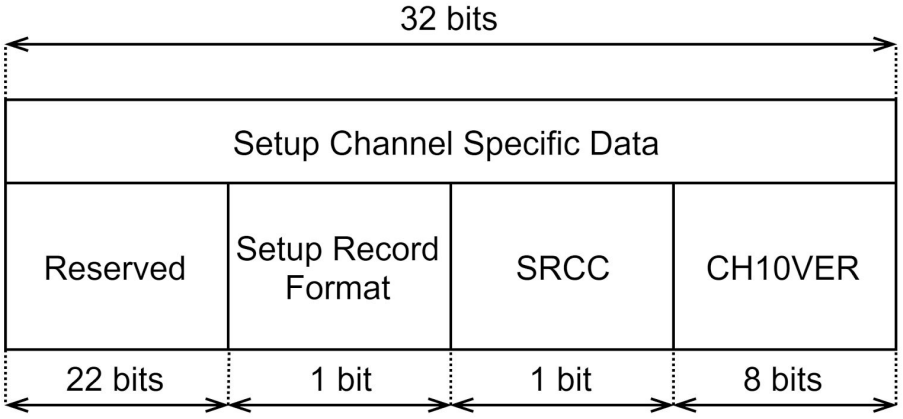

Figure 12: Format of the Channel Specific Data for a IRIG-106 Chapter 10 Setup Record.

1. Reserved. The first 22 bits of the channel specific data are reserved for future use.

2. Setup Record Format. Specifies the format of the setup record. If format is set to 1 the record will be XML encoded and if set to 0 the record will be ASCII encoded.

3. Setup Record Configuration Change (SRCC). This value will be set if the configuration has changed since the previous setup record.

4. RCC 106 Chapter 10 Version (CH10VER). This value contains a bit pattern representing the Chapter 10 version that this setup record is applicable for. These are the values currently supported:
    a. 0x00-0x06 = Reserved
    b. 0x07 = RCC-106-07
    c. 0x08 = RCC-106-09
    d. 0x09 = RCC-106-11
    e. 0x0A = RCC-106-13
    f. 0x0B = RCC-106-15
    g. 0x0C-0xFF = Reserved

3. Method

This chapter explains the method used to answer the research questions. It explains the process of converting a CCSDS packet to a Chapter 10 file. Lastly it describes how the application was evaluated with the help of the ANTS Performance Profiler.

The main purpose of this thesis is to develop an application which converts telemetric data from the CCSDS standard to the IRIG-106 Chapter 10 Digital Recording Standard. Since CCSDS packets are extremely rare to get a hold of we have to follow the CCSDS packet protocol to create our own CCSDS packets. The conversion application will only handle telemetry packets and will assume that there exists no ancillary data. It will however be able to process a secondary header, which will contain either the day segmented time code or the calendar segmented time code. These were chosen since they are similar to the timestamp in the IRIG-106 time packets. This leaves the application with a few variables that may differ depending on the packet. The application will be able to handle two different time code formats: day segmented and calendar segmented. Both of these have two variants of each other creating a total of four different structures the time codes can take. These will be extracted and put into IRIG-106 time packets. This application is going to extract the data from the CCSDS packet and wrap it into a Chapter 10 file. In order to do this an understanding of how both standards packet data is necessary. The applications were developed in C#.

3.1 Creating CCSDS packets

The application needs CCSDS packets to do the conversion. However it seems hard to come by such packets. Sample data was not provided by ICS so the project began by constructing these packets according to the documentation available to the public [2]. These were telemetry packets. The IRIG-106 standard requires time packets in between the data packets. The time packets will contain the time code provided by the CCSDS packet. A secondary header with the time code becomes mandatory for the CCSDS packets. The different time codes tested were day segmented and calendar segmented time codes.

The actual data itself is not of value to the study, it is however necessary that the data is consistent throughout the conversion. It is simply the extraction of the data that is of importance. The PCM data that were put into CCSDS packets was approximately 400 megabytes. The data was stored as a Comma Separated Value (CSV) file which amounted to over 3 million lines of data. An application was created that read each line from the CSV and wrapped it into a CCSDS packet. This means that each CCSDS packet contained one line of data. When all the data had been read, these packets were written sequentially to a binary file. The binary file became smaller in comparison to the CSV file, with a size of 265 megabytes. This due to the fact that it discards the ascii encoding and the commas of the CSV file. There is not anything that separates the packets in the binary file which makes it vital to read the correct amount of bytes from it in order for the conversion to work. There is however no real way of knowing if these packets are valid since only assumptions can be drawn.

3.2 Extracting the CCSDS Packets

The first step is to extract the CCSDS packets from the binary file. The first part of the CCSDS packet is the packet primary header. As mentioned it contains necessary metadata for the rest of the packet. The entire binary file is read into a byte array. This byte array now contains every CCSDS packet created before. The packet primary header is of fixed length which simplifies the extraction. The four major fields described in 2.1 are extracted bit by bit.

The next step is to handle the time code and the data. This is where the information recently extracted comes into play. It is necessary to check if a secondary header exists, and if so, check what time code format it uses. If no secondary header is present, the application simply assumes that the data is next. Otherwise it will extract a certain number of bytes and define this as a time code. The size of the time code depends on the time code format. The day segmented ones are ten bytes and the calendar one are twelve bytes long. The application also assumes that no extension follows the preamble. The final step is to obtain the data inside the CCSDS packet.

3.3 Creating a Chapter 10 file

Every Chapter 10 file begins with a setup record. This record is used by the receiving end to help parse the data sent during the transmission. The setup record will not contain any information extracted from the CCSDS packet and is most often generated automatically. Generating this record manually is outside the scope of this study. In order to get the setup record for the application, ICS:s own software Netview is used. With Netview we can create and configure virtual PCM and Time channels within the software. These channels were then exported as a TMATS file, describing those channels. The TMATS file is then read into our conversion application and used as data for the setup record.

3.3.1 Time data packets

A time packet must follow the setup record. The time packet much like everything else, has a standard packet header, a channel specific word, time data and filler. The packet header is created as described in figure 7. The packet specific variables are of course set to match a time packet. Following the packet header is the channel specific word. It is a total of four bytes. This can be seen as a packet type specific header. It contains information about the packet type, in this case the time packet. The date format is set to a false leap year and IRIG day format. The time format is set to IRIG-B which is the standard IRIG time format. The time source is external.

The data inside the time packet is a timestamp and it is acquired from the time code in the CCSDS packet. After the time data, the filler is inserted into the packet making it complete. The time packet should arrive with a maximum of 1 Hz, i.e. every second. It is after the first time packet that the data packets can be sent. They are PCM packets.

3.3.2 Pulse code modulation data packets

The majority of the PCM channel specific data concerns frames. It has the frame lock status and the frame indicators. The packets created will be in throughput mode which removes the need to configure and work with frames. The packets contain pure data in no specific order and do not need to be synced with the help of frames. This makes it faster to transfer but gives the receiving end the job of "deciphering" it. The sync offset is set to 0 and the intra-packet header indicator is set to *false*. Lastly the right amount of filler bytes need to be inserted to match the alignment mode. We are creating packets with 16 bit alignment. The amount of filler depends on the size of the data. Put all of these pieces together and it will make up a Chapter 10 file.

3.4 Evaluating performance

It is stated in the Digital Recording Standard that the maximum packet size is 2^{19} bytes [1]. Two options become available in order to answer the extreme cases. The option to take the one CCSDS packet and simply convert it into a single PCM packet and the option to take as many CCSDS packets as possible and cram the data into a single PCM packet. These cases will be referred to as OTOC (One To One Conversion) and FSC (Full Sized Conversion). They have the packet sizes 2^6 bytes and 2^{19} bytes respectively. The former should create more overhead since it should contain more headers.

There exist 64 (2^6) bytes of data inside each of the CCSDS packets. Fourteen different cases are going to be studied, starting with the OTOC. The PCM packet size is then going to be doubled and the application runtime evaluated again. When the packet size reaches full sized PCM packets there should exists fourteen measurements of the application's runtime. The runtime will be measured using the *ANTS Performance Profiler 9*.

3.5 ANTS Performance profiler

The ANTS Performance profiler accurately measures the performance of .NET applications. It can display detailed statistics down to every single line of code. However, choosing a higher detail level increases the overhead generated by the profiler. In other words, the runtime of the application will increase because the profiler itself takes a large part of the CPU capacity. This causes the overhead to be very inconsistent between each run. Statistics of single lines of code is not of importance in this study and thus limiting the profiler to analyze methods is sufficient. Internal .NET methods are also excluded in the profiling since the point of the profiling is to analyze our own methods; whereas the internal methods muddy the results with a lot of inconsistent overhead.

There are two functions of interest in the application. The function that reads the CCSDS data and the function that creates the Chapter 10 file. These functions are evaluated with the performance profiler. The I/O usage was also recorded since it is expected to have a vastly different utilization between the different packet sizes.

4. Results

This chapter presents the results of the study. The applications that were developed are concretely described and the performance statistics generated by the ANTS Profiler are presented.

4.1 The Applications

Two different applications were created. The first created a file containing CCSDS packets and the second converted those CCSDS packets to a Chapter 10 file. They were developed using Visual Studio 2017 and written in C#.

The first application has three important tasks. It needs to read the data from the CSV, create a CCSDS packet and lastly write the packets to a binary file. The data is read one line at a time. It discards the commas and proceeds to create a CCSDS packet which it appends to a list. The application has the ability to create CCSDS packets with two different time codes: day segmented and calendar segmented, which depends on a variable. The next step is to check if it should dump all the CCSDS packets to a binary file. This depends on an arbitrary variable and was used to ensure that the list of CCSDS packets did not grow too big. It was set to 200 000. This meant the application dumps all CCSDS packets to the binary file after 200 000 lines of data has been read. The list is then cleared and the application continues to read data. This process is explained in figure 13.

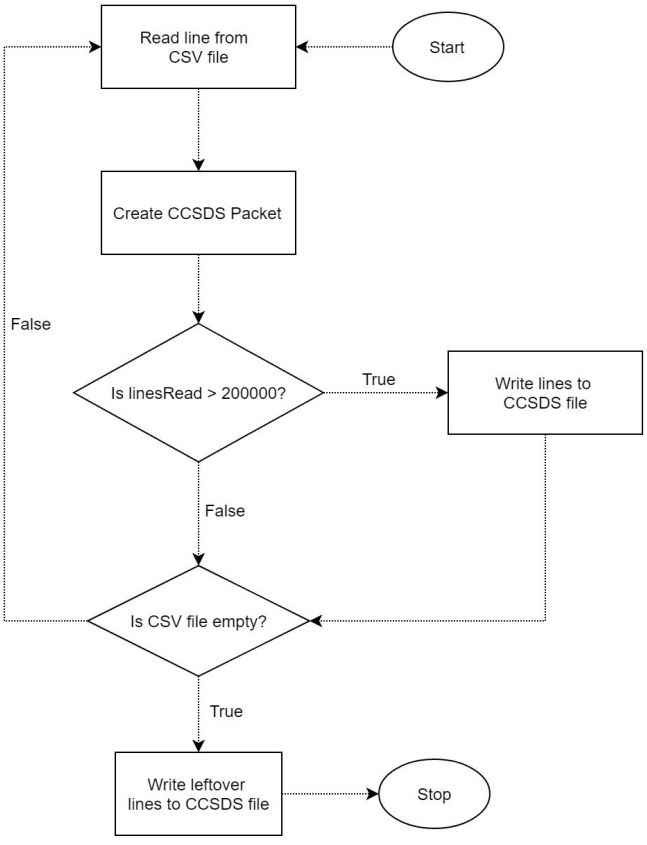

Figure 13: Flowchart of the application that creates CCSDS files

The second application does the actual conversion and is therefore more complex than the first one. It begins by reading the binary file created by the first application and inserts this into a buffer. The buffer acts as the byte array in which all the CCSDS packets are placed. These are not separated by anything which makes it vital to read the right amount of bits from the buffer. The extraction is put into a loop. Every iteration begins by extracting the primary header (48 bits) and the time code (10 or 12 bits). The next step concerns the actual data inside the CCSDS packet. With the help of the metadata found in the primary header, the application now knows how much data is inside the CCSDS packet.

```
for (int i = 0; i < data.Length; i++)
        data[i] = (byte)bitBuffer.GetBits(8);
```

Since the data length gives the length in bytes, eight bits(one byte) need to be extracted in each iteration. The function GetBits(value) obtains the number of bits specified by value from the buffer and moves forward in the buffer the same amount of steps. After the application has both the primary header and the data, it creates a class out of both of these and appends the class instance into a list. The instance then acts as one CCSDS packet. This extraction process is put into a loop which reads until the end of the byte array (end of file). The list of packets is then returned to the main program after all the packets have been obtained.

After the CCSDS data has been read the first step is to create the setup record and initial time packet. This is how every Chapter 10 file has to start according to the standard. Then the application iterates through the list of CCSDS packets in order to transfer each packet one by one into a PCM packet. Since PCM packets usually have a larger size than CCSDS packets, the application has to check whether it can fit another CCSDS packet after each iteration. In an attempt to save I/O usage, no data is written to the file until the entire PCM packet is complete.

According to the Digital Recording Standard, time packets must be written at least once each second of the transmission. This issue was solved by checking how much time has passed since the last time packet. If the elapsed time has passed one second, a time packet is created. The timestamp of said time packet is set to the most recent PCM packets time, rounded to whole seconds. When all CCSDS packets have been converted to PCM packets, and all PCM packets have been written to file, the application will terminate. The conversion process is explained in figure 14.

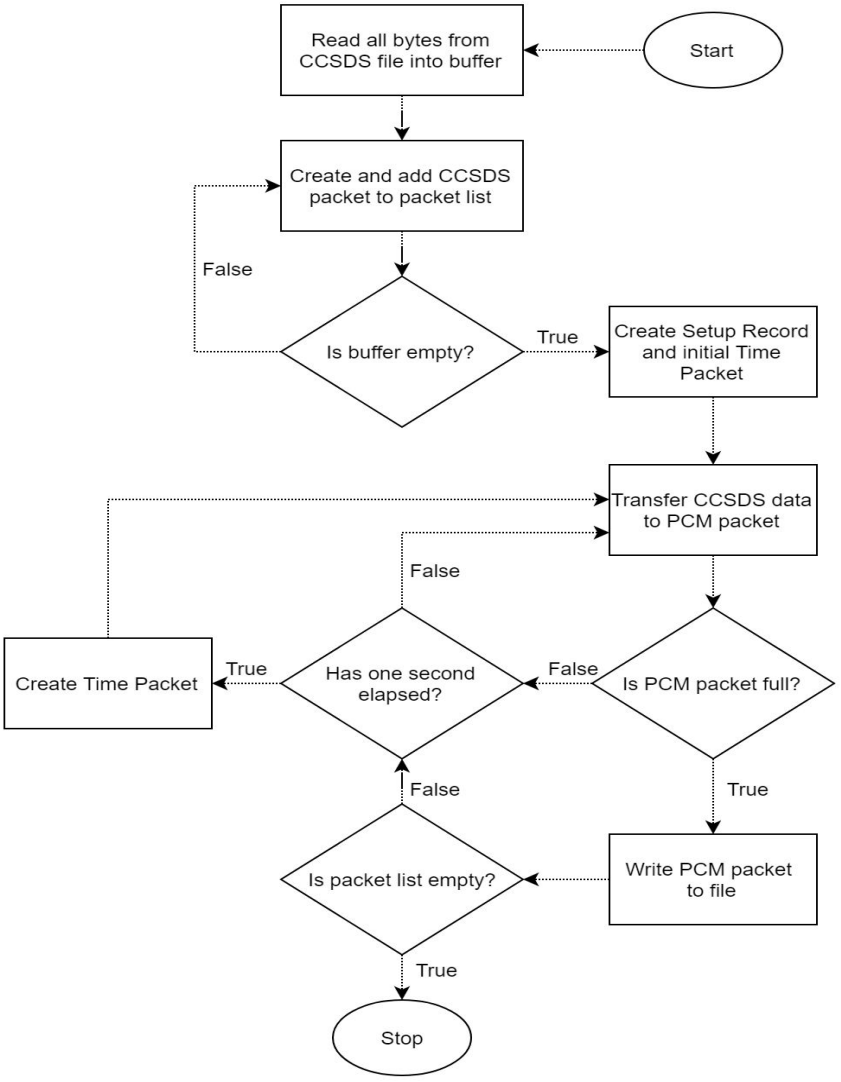

Figure 14: Flowchart of the application that creates Chapter 10 files

4.2 Performance results

Simply converting the data from a CCSDS packet to a IRIG-106 Chapter 10 file proved to be possible. A total of fourteen different cases were studied with the differing factor being packet size. This created not only a difference in file size but also impacted the application's runtime.

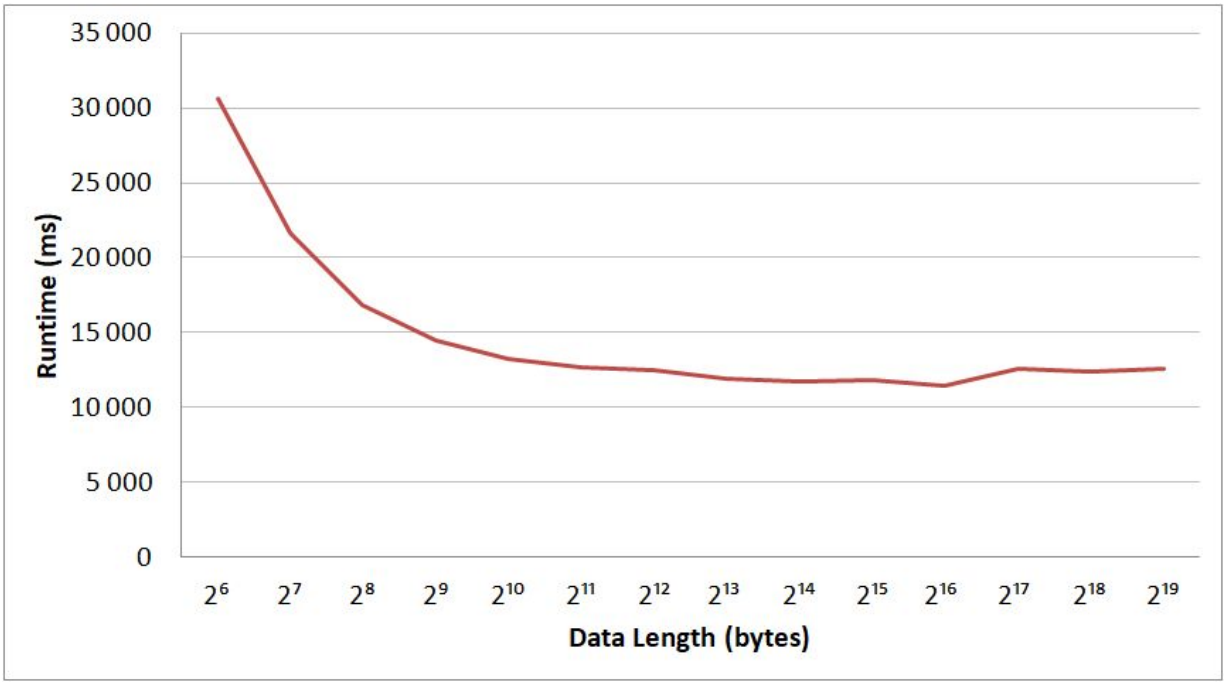

Figure 15: Total time it took for the application to complete for the different packet sizes.

The OTOC took the largest amount of time, taking more than 30 seconds to complete. It then began to gradually decrease in runtime until the packet size reached 2^{16}. At this point the application only took around 11.5 seconds to complete which is less than half the time of OTOC. The increase in packet size beyond this threshold did not seem to have a significant impact on the speed of the application, although it did seem to rise again by a few hundred milliseconds when nearing FSC.

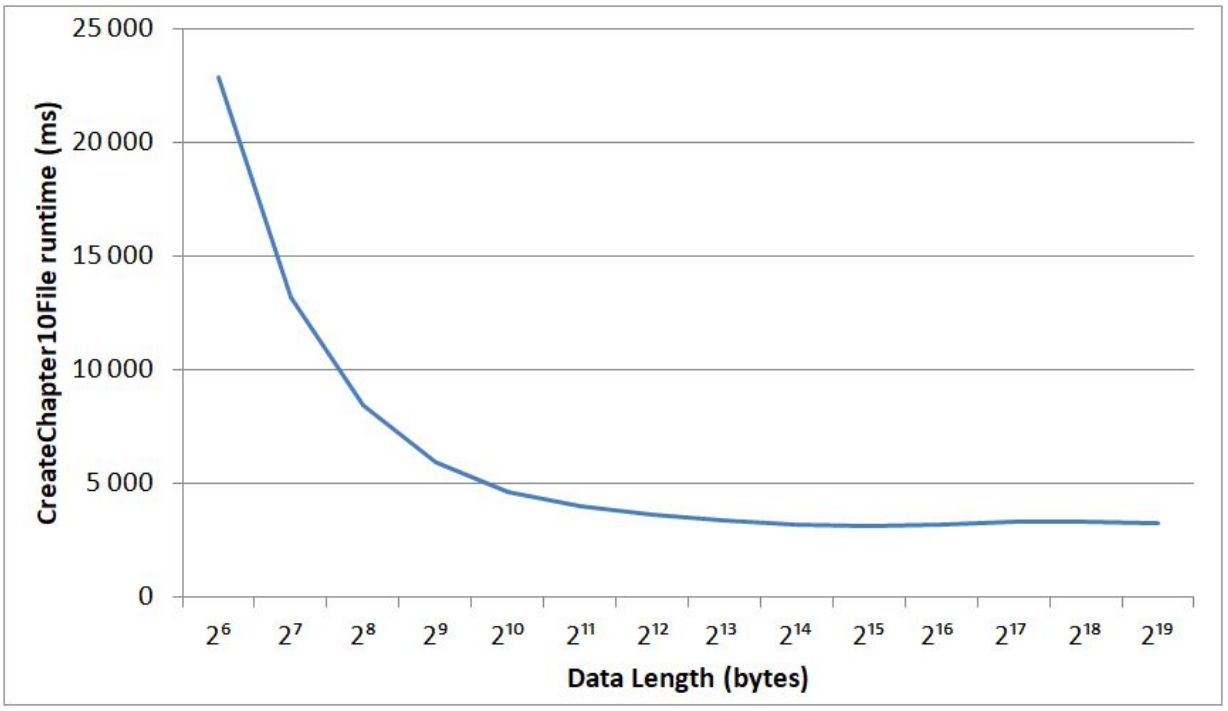

Figure 16: Total time it took for the function creating the Chapter 10 files to complete for the different packet sizes.

When the packet size increased the runtime of the function decreased. The major difference here is that the runtime did not rise noticeably when nearing full packet capacity.

It took approximately 8.7 seconds for the application to read the data and create the CCSDS packets. This value did not deviate throughout the different cases since the data was always the same. It was excluded in the table 1 because of this. The I/O write operation and the number of packets were also observed and presented in table 1.

Data Length (bytes)	Main (ms)	CreateChapter10File (ms)	I/O Write (ms)	Amount of Packets	File Size (kb)
2^6	30 704	22 894	706	3 401 016	318 851
2^7	21 778	13 161	602	1 700 508	265 710
2^8	16 932	8 419	518	850 254	239 140
2^9	14 608	5 939	458	425 127	225 855
2^{10}	13 300	4 629	511	212 564	219 212
2^{11}	12 646	4 014	461	106 282	215 891
2^{12}	12 425	3 642	566	53 141	214 230
2^{13}	11 949	3 356	326	26 571	213 400
2^{14}	11 709	3 213	229	13 286	212 985
2^{15}	11 650	3 137	150	6 643	212 777
2^{16}	11 519	3 172	114	3 322	212 673
2^{17}	12 623	3 300	82	1 661	212 621
2^{18}	12 479	3 291	73	831	212 596
2^{19}	12 497	3 279	66	416	212 583

Table 1: Results from the ANTS Performance Profiler.

Four different metrics varied for each case. The runtime of Main and CreateChapter10File functions, the runtime of the I/O write operation and the amount of packets. All of these are presented in table 1. The *Main* function sums up the entire application and the CreateChapter10File is the function that does the actual conversion. The runtimes were all measured in milliseconds.

5. Discussion

Every metric in table 1 decreases as the packet size increases, at least until it reaches a packet size of 2^{16}. After that point the curve stabilizes and the data length does not seem to have a lasting impact on the runtime. Interestingly enough when the data length approaches FSC (around 2^{17}), the runtime seems to slightly rise again. This concludes that neither of the extreme cases (OTOC and FSC) is the optimal choice when opting for the fastest solution. The optimal data length when it comes to runtime is 2^{16} bytes, but runtime is not the only factor. One could for example use a data length of 2^{13} bytes which is 16 times smaller at the cost of merely 3.7% increase in runtime.

Having a considerably smaller data length can lead to a few advantages. Most notably is an increase in consistency during the transmission. Occasionally packets will be lost during transmission in which case it is far less disruptive to use small packets. Since the Digital Recording Standard is used within aviation, consistent and reliable communication is of utmost importance. Further work could involve analyzing how frequent packet loss is within aviation and compare how much data is lost when using the different packet sizes.

Paolini et al. [8] explains in their research that it is extremely time consuming to retransmit lost packets when it comes to long distance communications. Having bigger CCSDS packets puts you at risk of losing more data if a packet loss occurs. It then becomes a question of how much time you can afford to lose, waiting for the lost data to be retransmitted. Large packets can be used if the data is not critical. The CCSDS max packet capacity is 2^{16} which coincidentally is where our application performs the best. The optimal case would be if the CCSDS packet were at max capacity. This would make the OTOC packet size 2^{16}. The concept and implementation of OTOC is simple which makes it desirable.

The file size of the chapter 10 file generated from the application decreases in size when the data length increases. This is easily explained by the fact that a bigger data length also means a decrease in amounts of packets. If there exists less packets, there exists less overhead. By doubling the data length, the amount of packets generated gets cut in half, as shown in table 1. There is a total difference of 106 286 kilobytes between OTOC and FSC when comparing file sizes. That points to FSC being 33 % smaller than OTOC while also surpassing in in terms of performance.

The results clearly show that packet size matters. A packet size of 2^6 is by far the least attractive solution since the overhead generated by each packet is overwhelming compared to the actual data. The runtime of the application is the lowest in the case of 2^{16} and the total runtime decreases by approximately 62,5 % compared to the runtime of case 2^6. This is a huge difference and further proves the point that the amount of data inside the packets matter. However the difference in runtime between cases 2^{14} and 2^{19} is not nearly as severe. This does not mean that the FSC is useless as it could still be used if memory is a limited resource.

6. Conclusion

What packet size proves to be optimal in terms of runtime and memory usage when converting CCSDS telemetry data to IRIG-106 Chapter 10 Digital Recording Standard?

By developing a conversion application in C# we were able to efficiently take telemetric data from the CCSDS standard and convert it to PCM data packet according to the IRIG-106 Digital Recording Standard. The packets could then be displayed in ICS own software, Netview. The case 2^{16} bytes comes off as the optimal solution when looking at the runtime of the application. However the packet size may be increased up to 2^{19} bytes if memory is a limited resource.

7. References

[1] Telemetry Group, Range Commanders Council (2015): *Digital Recording Standard*

[2] National Aeronautics and Space Administration, NASA (2003): *Space Packet Protocol, CCSDS 133.0-B-1*

[3] Li Guojun, Zhang Running, Shi Jian. "*Lossless data compression algorithm for satellite packet telemetry data,*" .Mechatronic Sciences, Electric Engineering and Computer (MEC), Proceedings 2013 International Conference on. 20-22 Dec. 2013.

[4] National Aeronautics and Space Administration, NASA (2014): *Overview of Space Communications Protocols, CCSDS 130.0-G-3*

[5] Space Communications and Navigation Office, 7L70 (2010): *Time Code Formats, CCSDS 301.0-B-4*

[6] Telemetry Group, Range Commanders Council (2015): *Pulse Code Modulation Standards*

[7] Telemetry Group, Range Commanders Council (2015): *Telemetry Attributes Transfer Standard*

[8] E. Paolini, M. Varrella, M. Chiani, and G. P. Calzolari, "*Recovering from packet losses in CCSDS links*," in Proc. 4th Adv. Satellite Mobile Syst. Conf., Bologna, Italy, Aug. 2008, pp. 283–288.