Software Decommutation Implementations for End Users

K. Wade Nye, Ulyssix Technologies, Inc. Wade@Ulyssix.com
Glenn Rosenthal, Ulyssix Technologies, Inc. Glenn@Ulyssix.com

Abstract

Advances in computer processing power enabled telemetry decommutation to transition from hardware to software implementations even with increasing data rates and complexity of telemetry commutation. End users desire to integrate decommutation with analysis scripts to create quick look results of their experiments. This paper discusses a methodology of writing software telemetry decoms in pseudocode based on the Ulyssix Tarsus Archive Data (TAD) data format and Ulyssix PCM hardware but all the ideas presented are usable across many different vendor PCM hardware.

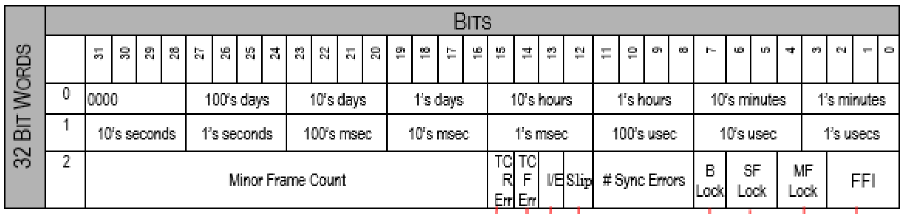
Introduction

Software telemetry decoms have fast development cycles and infinite flexibly in implementing new features. The concatenation of these new customer driven features result in a complex vendor software package. For some applications, like telemetry check out of a test article, a custom software decom solution is desirable. The benefits include simpler operation, simpler documentation, and reduction of human error during expensive text events.

Publishing live telemetry data over computer networks coupled with the desire for quick look analysis of telemetry data led to the need for customized integrated software decoms and analysis. Some applications include feeding the telemetry data into decommutation and analysis scripts written in languages such as Python, MatLab, or C#.

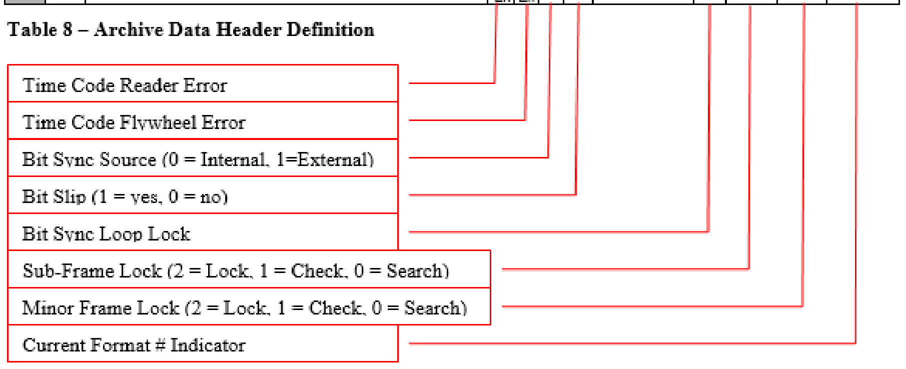
Tarsus Archive Data Format

The Tarsus Archive Data format (TAD) format begins with a file header followed by packed telemetry minor frames appended with a data header (TarsusHS User's Manual Appendix B). When working from a TAD file the initial 328 byte file header should be ignored (this is used exclusively with the Ulyssix TarsusPCM and Altair software suites). The minor frame data header contains three 32-bit integers. The first 64-bits are a Binary Coded Decimal (BCD) time stamp in Julian Day/Hour/Min/Sec with 1 microsecond resolution format. The last 32-bit integer includes a 16-bit minor frame counter and lock indicators. All Ulyssix PCM hardware Direct Memory Access (DMA) transfers data to the computer in the TAD format. TAD data packets can be broadcast over computer networks via User Datagram Protocol (UDP).



32 Bit Words	Bits																															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0000				100's days				10's days				1's days				10's hours				1's hours				10's minutes				1's minutes			
1	10's seconds				1's seconds				100's msec				10's msec				1's msec				100's usec				10's usec				1's usecs			
2	Minor Frame Count																TC R Err	TC F Err	I/E	Slip	# Sync Errors				B Lock	SF Lock		MF Lock		FFI		

Table 8 – Archive Data Header Definition


Time Code Reader Error
Time Code Flywheel Error
Bit Sync Source (0 = Internal, 1=External)
Bit Slip (1 = yes, 0 = no)
Bit Sync Loop Lock
Sub-Frame Lock (2 = Lock, 1 = Check, 0 = Search)
Minor Frame Lock (2 = Lock, 1 = Check, 0 = Search)
Current Format # Indicator

Due to operating system preference, the computer receives the TAD data from the Ulyssix PCM hardware in Little Endian byte order. In Little Endian byte order, the least significant byte occurs first in a 32-bit integer. Telemetry data is commutated in Big Endian byte order, where the most significant byte occurs first. This Endianess mismatch results in a difference in the telemetry byte number and the computer data byte number.

The two examples below show Decom Parameters affected by Little Endian Byte Order. In the first example, a 16-bit Decom Parameter is broken from sixteen contiguous bits in the telemetry data stream (yellow) into two non-adjacent byte locations (gray) by Little Endian byte order. In the second example, a 48-bit Decom Parameter is broken from forty-eight contiguous bits in the telemetry data stream (yellow) into three non-adjacent sets of bytes (gray) by Little Endian byte order. Breaking a Decom Parameter into non-adjacent bytes complicates the software decommutation algorithm.

16-Bit Decom Parameter and Little Endian Byte Order

	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	B16
Telemetry Byte Order	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Little Endian Byte Order	4	3	2	1	8	7	6	5	12	11	10	9	16	15	14	13

48-Bit Decom Parameter and Little Endian Byte Order

	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	B16
Telemetry Byte Order	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Little Endian Byte Order	4	3	2	1	8	7	6	5	12	11	10	9	16	15	14	13

For a given telemeter, the frame size, measured in bits per minor frame, and bit rate are used to calculate the Packed Words and Block Size. Packed Words is the number of 32-bit integers required to contain a minor frame plus the TAD Header (three 32-bit integers). The number of 32-bit integers is important because data is moved from the Ulyssix PCM hardware to the computer in increments of 32-bits. Any unused bits in a 32-bit integer are set to 0. Packed Bytes equals Packed Words multiplied by 4. This converts from 32-bit integers to 8-bit bytes.

$$Packed\ Words = Ceiling(\frac{bits\ per\ minor}{32}) + 3$$

The Block Size is the integer number of minor frames that occur in 10mS. The 10mS time interval is based on Microsoft Windows servicing DMA interrupt requests from Ulyssix PCM hardware. Ulyssix PCM hardware delivers Block Size number of Packed Words each 10mS.

$$Block\ Size = Floor(\frac{bitrate * 0.01}{(bits\ per\ minor\ frame)})$$

Setting Up the Frame

The first step in setting up a software decom is to define the parameters that describe the frame. A telemetry frame includes a Frame Sync Pattern and at least one minor frame. Often there are multiple minor frames per major frame. The Minor Frame Counter runs from zero to number of minor frames minus one, however this paper begins counting minor frames at Minor Frame 1. The Frame Sync Pattern is defined as the first word in the minor frame (IRIG 106-15 Chapter 4). This paper refers to the first word after the Frame Sync Pattern as Word 1. Frame length can be defined in many ways, but this paper uses *Bits per Minor Frame* as the total number of bits in the minor frame including the Frame Sync Pattern.

In many older telemeters, each Decom Parameter had the same number of bits. This is referred to as Fixed Bits per Word. The need to pack more data into a fixed frame size and the desire to include more complicated data encoding, like floating point numbers, led to more complicated commutation with variable words sizes. This is referred to Variable Bits per Word. For a decom with Variable Bits per Word, the starting point is to build a table with the word number and word length. The Variable Bits per Word table must be the same for each minor frame in a major frame.

The examples below show two different frames that both have 832 bits per minor frame including the 32-bit Frame Sync Pattern. The first frame has one hundred 8-bit words and a 32-bit frame sync pattern (only the first eighteen words are shown). This is a Fixed Bits per Word Frame.

Fixed Bits per Word Frame

FS	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15	W16	W17	W18
32	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8

The second example is a Variable Bits per Word frame. There is a mix of 8-bit and 10 bit words. In this example, there is a repeating pattern of word lengths every nine words (shown in gray) this kind of repetition is common, but not required with Variable Bits per Word. The repeating block of data has a total of 80 bits. Ten of these blocks occur in the frame resulting in 32 bits of Frame Sync Patter and 800 bits of data.

Variable Bits per Word Frame

FS	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15	W16	W17	W18
32	8	10	8	10	8	10	8	10	8	8	10	8	10	8	10	8	10	8

Define Decom Parameters

Each Decom Parameter has attributes required to extract the binary value from the telemetry stream. The commutation type of the Decom Parameter determines some of the required attributes. Commutation types simplify the definition of where the word occurs in the frame. IRIG 106-15 Chapter 4 defines Normal, Sub, and Super Commutation. In Normal Commutation, the parameter occurs once per minor frame and occurs in the same word number in every minor frame. In Super Commutation, the parameter occurs multiple times in each minor frame. The occurrences of the parameter must be evenly spaced by a fixed number of words called the Interval. Please note with Variable Bits per Word that Super Commutated parameters are expected to be equally spaced in bits not words. In Sub Commutation, the parameter occurs in the same word number but does not in every minor frame. The attribute, Frames, is the number of minor frames between occurrences of the parameter.

For added flexibility in decommutation, non-IRIG Random and Random Normal Commutation should also be considered. In Random Commutation, words can occur in any word in any minor frame. For a Random Commutated word, each instance in the frame must be defined by its Word Number and Minor Frame Number. For a Random Commutated Decom Parameter, the Interval should be the number of samples per major frame. In Random Normal Commutation, the parameter can occur in any word in a minor frame, but must occur in the same words in every minor frame. Each occurrence must be defined by its Word Number. For a Random Normal Commutated Decom Parameter, the Interval should be the number of samples per minor frame.

Random Commutation

	Sync	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7
Frame 1		🟨						
Frame 2			🟨					
Frame 3					🟨			
Frame 4			🟨					

Random Normal Commutation

	Sync	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7
Frame 1		▨		▨				▨
Frame 2		▨		▨				▨
Frame 3		▨		▨				▨
Frame 4		▨		▨				▨

Each Decom Parameter needs to have the following attributes defined, depending on its commutation type. See the chart below for the required value for each commutation type.

Word Number – The number of word in the Variable Bits per Word table.
Minor Frame Number – The number of the minor frame.
Frames – The number of minor frames between Sub Commutated word occurrences.
Interval – For Super Commuated words, the number of words between parameter occurrences. For Random, the number of parameter occurrences in the major frame. For Random Normal, the number of parameter occurrences in the minor frame.

	Word Number	Minor Frame Number	Frames	Interval	Sample Number
Normal	🟩	1	0	1	1
Sub Comm	🟩	🟩	🟩	1	1
Super Comm	🟩	1	0	🟩	1
Random	🟩	🟩	0	# Samp	🟩
Random Normal	🟩	1	0	# Samp	🟩

🟩 Requires user entry for parameter

After each Decom Parameter is extracted from the telemetry stream, the series of bit needs to converted to a data type. The series of bits must be manipulated to conform to Most Significant Bit (MSB) first or Least Significant Bit (LSB) first. Typical data types include unsigned binary, two's compliment signed binary, one's compliment signed binary, binary coded decimal, and floating point numbers.

Software Decom Methodology

Reorganizing each Block Size of data to Big Endian byte order would be algorithmically easy, but the resulting code must run on each data acquisition, is computationally inefficient, and has performance issues at higher bit rates. Since the Decom Parameter's position in the telemetry frame is constant, the corresponding byte location in the Little Endian data can be calculated before data acquisition begins. This calculation is done for every word in the Variable Bits per Word table. Then during data acquisition, a Decom Parameter can reference the pre-data acquisition calculation on the Variable Bits per Word table to determine to byte location in the Little Endian data. This method requires more complex code but is computationally efficient.

Pre-Data Acquisition Calculations

For simplicity, the pseudocode calculations in this paper are for Decom Parameters with a maximum length of 32-bits. The code is easily expandable to Decom Parameter lengths with a maximum of 64-bits, but, as previously mentioned, these calculations require involve Little Endian byte order breaking a Decom Parameter into three or more non-adjacent byte locations and unduly complicate the example code. All calculations assume that the Decom Parameter is MSB. A conversion to LSB occurs during data acquisition, if applicable. Here are the needed data structures in pseudocode:

```
class(Word)
{
    int StartByte;      //first byte location
    int Offset;         //bit offset in Start Byte
    int EndByte;        //Second byte location for word split by Little Endian
    int LengthEnd;      //bit length of End Byte
    int TotalBits;      //bit count from the start of the minor frame to first bit in the word
    ulong Mask;         //bit mask for words not split by Little Endian
    ulong StartMask; //bit mask for first byte location in Little Endian
    ulong EndMask;   //bit mask for last byte location in Little Endian
}
```

```
List<Word> Words;   //list of words where first entry is the FS Pattern

class(VarBitsPerWord)
{
      int bitLength;      //number of bits from start of frame
}

//variable bits per word table
List<VarBitsPerWord> VariableBitsPerWordTable;   //starts with FS Pattern
```

The Pre-Data Calculations step through the bit lengths in the Variable Bits per Word table and calculate the byte location and bit masks for every word location in the frame:

```
int bitLocation = 0;     //number of bits from start of minor frame to start of word
int wc = 0;              //word count in the var bits per word table

//step through each bit length in the Variable bits per Word Table
foreach (bitLength in VaraibleBitsPerWordTale)
{
      Words[wc].StartByte = Floor((bitLocation-1)/8);
      Words[wc].Offset = MOD(bitLocation-1, 8);
      Words[wc].EndByte = Floor((bitLocation-bitLength)/8);
      Words[wc].LengthEnd = 0;
      Words[wc].TotalBits = bitLocation;

      //correct offset for Little Endian
      if(Words[wc].Offset > 0)
            Words[wc].Offset = 8 - Words[wc].Offset;

      //correct start and end bytes for Little Endian
      int LEoffsetStart = MOD(Words[wc].StartByte, 4)
      int LEoffsetEnd = MOD(Words[wc].EndByte, 4)
      Words[wc].StartByte = Floor(Words[wc].StartByte/4)*4 + (3-LEoffsetStart);
      Words[wc].EndByte = Floor(Words[wc].EndByte/4)*4 + (3-LEoffsetStart);

      //check if little endian correction splits the word - aka start > end
      if (Words[wc].StartByte > Words[wc].EndByte)
            Words[wc].LengthEnd = bitLength-(LEoffsetEnd*8)- (8-Words[wc].Offset);

      //move End Byte back to the beginning of the 32-bit block
      Words[wc].EndByte = Floor(Words[wc].EndByte/4)*4;

      //create bit masks for isolating desired bits from bit stream
      Words[wc].Mask = Math.Pow(2, bitLength) - 1;
      Words[wc].StartMask = (Math.Pow(2, bitLength - Words[wc].LengthEnd) - 1) << Words[wc].Offset;
      Words[wc].EndMask = Math.Pow(2, Words[wc].LengthEnd) - 1;

      //increment bit location and word count
      bitLocation += bitLength;
      wc++;
};
```

Data Acquisition Calculations

A block of data is processed immediately when it arrives from the Ulyssix PCM hardware. Each block of data contains Packed Words * Block Size * 4 bytes. The most computational efficient way to extract Decom Parameters from the block of data is to use direct memory addressing of the data via pointers.

The data structure for a Decom Parameter is composed of the Commutation Type, Interval, Frames, and a List of Samples. Each Sample has a Word Number, Minor Frame Number, and Sample Number. The Decom Parameter should be defined by the user and saved to a file before data acquisition. The data structured below are in pseudocode:

```
class Sample
{
    int WordNumber;       //number in Variable Bits per Word table
    int MFrameNumber;     //minor frame number
    int SampleNumber;     //number of sample starting at 1
}

//create an object for a parameter
class Param
{
    Comm;                 //commutation type
    int Interval;
    int Frames;
    int Length            //word length in bits
    ulong mask;           //bit mask based on word length
    List<Sample> Samples;
}

//a list of parameters
List<Param> ParamList;
```

The following pseudocode steps through each parameter and moves the pointer to the location of each occurrence of the parameter in the data block. Then the methods GetParamValue and GetParamTime extract the binary value and time stamp, in microseconds, for the Decom Parameter from the data block. These two methods are discussed later in the paper.

```
int dataSize = BlockSize*PackedWords*4;  //bytes in a data block
byte* pStartBlock = dataBlock;           //pointer to start of data block
ulong Time;                              //time stamp in BCD from TAD header
int fCount;                              //value of frame counter from TAD header

//extract the time stamp and frame counter from first header in block
void ConvertHeader(pStartBlock, out Time, out fCount);

foreach(p in ParamList)
{
    byte* pBlock = pStartBlock;      //create pointer that can move from original position
    int sampleInit = 0;              //Initial sample number - 0 for all but Random Commutation
    long mfIncr = PackedBytes;       //Minor frame increment in number of byes
    uint tsBitCount=0;               //bits from first bit in minor frame to calculate time
```

```
//if commutation does not occur in every minor frame, move pointer to minor frame with data
if (p.Comm == Sub or Random)
{
    mfIncr = PackedBytes * p.Frames

    //move to sub frame that has an instance of parameter p
    if (fcount < p.Samples[0].MFrameNumber)
        pBlock += (p.Samples[0].MFrameNumber - fCount) * PackedBytes;
    else
        pBlock += (p.Interval - MOD(fCount - p.Samples[0].MFrameNumber, p.Interval)) *
            PackedBytes;

    //if pBlock is greater than the DataSize skip out of the loop
    if ((pBlock < pStartBlock) or (pBlock - pStartBlock > DataSize))
        continue;
}

//step through each minor frame in the data block
for (i = p.Samples[0].MFrameNumber; i < BlockSize; i =+ p.Interval)
{
    //ensure that the pointer location is inside the data block
    if ((pBlock < pStartBlock) or (pBlock - pStartBlock > DataSize))
        pBlock = pStartBlock;

    //get time stamp and frame count for new minor frame
    ConvertHeader(pBlock, out Time, out fCount);

    //Random Commutation word position changes every minor frame - do calculation every minor
    if (p.Comm == Random)
    {
        //step through each sample and set sampleInit to lowest Sample Number
        sampleInit = int.Max;       //set samplerInit to max and then loop finds lower values
        foreach(sample in p.Samples)
        {
            //if sample is not in this minor frame, skip rest of loop
            if (sample.MFrameNumber != fCount)
                continue;
            //find lowest sample number in this minor frame
            if(sample.SampleNumber < sampleInit)
                sampleInit = sample.SampleNumber - 1;//SampleNumber start at 1 - frame count at 0
        }
        //if sampleInit is not changed, set to 0
        if (sampleInit == int.Max)
            sampleInit = 0;
    }

    //step through the samples and extract value and time for the parameter
    for (s = sampleInt; s < p.Interval; s++)
    {
        //get the parameter value and time stamp
        RawValue = GetParamValue(pBlock, sample, fCount, p, out tsBitCount);
        TimeStamp = GetParamTime(Time, tsBitCount);
    }

    //move the pointer to the next minor frame
    pBlock += mfIncr;
    }
}
```

The method GetParamValue takes the pointer and sample number and extracts the raw binary value from the data block using the Pre-Acquisition Calculations from the Variable Bits per Word table. A 64-bits unsigned integer is extracted from the data block via a pointer. To convert the 64-bits to the Decom Parameter's raw binary value, the 64-bit unsigned integer is bit shifted and then bitwise Boolean AND with a bit mask. At the end of this method, the raw binary value can be converted to LSB or to other data types. These binary manipulations are widely available and not included in this document.

```
ulong GetParamValue(byte* point, int sample, int frameCount, Param pc, uint* bitcount)
{
    ulong frameWord = 0;        //used to hold raw binary value for output
    ulong* startP;              //pointer to data location for param or 1st part of LE param
    ulong* endP;                //pointer to data location for 2nd part of Little Endian param
    int wnum;                   //word number

    foreach (s in pc.Samples)
    {
        //set word num for diff comm types
        if (pc.Comm == RandomNormal)
        {
            //continue if requested sample does not match the sample number in the current param sample
            if (sample != s.SampleNumber - 1) //sample counts from 0 and SampleCount from 1
                continue;
            wnum = s.WordNumber;
        }
        else if (pc.Comm == Random)
        {
            //continue if minor frame count does not match minor frame number in current param sample
            if (framecount != s.FrameNumber)
                continue;
            //continue if requested sample does not match the sample number in the current param sample
            if (sample != s.SampleNumber - 1) //frame counts from 0 and SampleCount from 1
                continue;
            wnum = s.WordNumber;
        }
        else
            wnum = s.WordNumber + pc.Interval*sample;

        //ensure that word num is a valid word number
        if (wnum < Words.Length)
        {
            //move pointer for 64-bit unsigned word location and set frameWord value
            startPtr = point + 12 + Words[s.WordNumber].StartByte;
            frameWord = *startPtr;
        }

        //if Start <= End then the word is not affected by Little Endian byte reorg
        if (Words[s.WordNumber].StartByte <= Words[s.WordNumber].EndByte)
        {
            //This is our param value! – shift bits and use mask to keep only desired bits
            frameWord = (frameword >> Words[s.WordNumber].Offset) &
                Words[s.WordNumber].Mask;
        }
        //else word is affected by Little Endian byte reorg
        else
        {
            //create second pointer for the Little Endian split word
            ulong* endPtr = point +12 + Words[s.WordNumber].StartByte;
            ulong frameWordEnd = *endPtr;

            //use mask to only keep the bits of interest and shift bits to desired location
            frameWordStart = (frameWord & Words[s.WordNumber].MaskStart) >>
                Words[s.WordNumber].Offset;
            frameWordEnd = (frameWordEnd & Words[s.WordNumber].MaskEnd) << (pc.Length –
                Words[s.WordNumber].LengthEnd);

            //combine start and end - this is our param value!
            frameWord = frameWordStart | frameWordEnd;
        }

        //get the number of bits from Frame Sync Pattern to calc time stamp
        *tsBitCount = Words[s.WordNumber].TotalBits;

        //at this point LSB and Data Conversion math should occur if needed!

        return frameWord;
    }
}
```

The method GetParamTime computes the time stamp for the Decom Parameter. It begins by converting the Time Header from BCD to microseconds. The Time Header is the time at the last bit of the minor frame. To calculate the time stamp for the Decom Parameter, the time delta from the Decom Parameter to the end of the minor frame is needed. The time delta is calculated by taking the number of bits from the Decom Parameter to the end of the minor frame, dividing by the bit rate, and then converting from seconds to microseconds. The Decom Parameter time stamp is the Time Header, in micro seconds, minus the time delta.

```
ulong GetParamTime(ulong Time, ulong tsBitCount)
{
    ulonog usTime = 0;    //holds the time stamp for the param

    //convert time header from BCD to Binary
    ulong usecs, msecs, secs, mins, hours, days;
    usecs = (pMFHeader->TimeWords >> 32 & 0xf) + (10 * ((pMFHeader->TimeWords >> 36) & 0xf))
        + (100 * ((pMFHeader->TimeWords >> 40) & 0xf));
    msecs = ((pMFHeader->TimeWords >> 44) & 0xf) + (10 * ((pMFHeader->TimeWords >> 48) &
        0xf)) + (100 * ((pMFHeader->TimeWords >> 52) & 0xf));
    secs = ((pMFHeader->TimeWords >> 56) & 0xf) + (10 * ((pMFHeader->TimeWords >> 60) & 0xf));
    mins = ((pMFHeader->TimeWords) & 0xf) + (10 * ((pMFHeader->TimeWords >> 4) & 0xf));
    hours = ((pMFHeader->TimeWords >> 8) & 0xf) + (10 * ((pMFHeader->TimeWords >> 12) & 0xf));
    days = ((pMFHeader->TimeWords >> 16) & 0xf) + (10 * ((pMFHeader->TimeWords >> 20) & 0xf))
        +  (100 * ((pMFHeader->TimeWords >> 24) & 0xf));
    //combine the time to get total microseconds
    usTime = (UInt64)(days * 86400000000) + (hours * 3600000000) + (mins * 60000000) +
        (secs * 1000000 ) + (msecs * 1000) + usecs;

    //calculate the time to the first bit in the Decom Param
    if (usTime > 0)
        //decrement usTime by bits from end of the minor frame div bit rate converted to uS
        usTime - = (ulong)(1/BitRate * 1.0e6 * (BitsPerMinorFrame - tsBitCount));

    return usTime;
}
```

Conclusion

This paper discussed an implementation for software decommutation of telemetry data based on the TAD format. Example pseudocode demonstrates extracting IRIG 106-15 commutation types as well as non-IRIG commutation types. Special consideration is given to a computationally efficient solution to Little Endian byte order data transfer. The example pseudocode lays out a starting point for a custom software decom that is approachable and flexible. Each specific telemeter commutation has its own unique challenges and requires customization to provided pseudocode.

References

IRIG 106-15, Chapter 4 Pulse Code Modulation Standards
http://www.irig106.org/docs/106-15/chapter4.pdf
TarsusHS User’s Manual, Appendix B Archive Data Files Explained
http://www.ulyssix.com/#!brochures-and-manuals/cwr9