ABSTRACT

A new decoder is being developed by the Jet Propulsion Laboratory for NASA's Deep Space Network. This unit will decode the new turbo codes, which have recently been approved by the Consultative Committee for Space Data Systems (CCSDS). Turbo codes provide up to 0.8 dB improvement in $E_b/N_0$ over the current best codes used by deep space missions.

The new decoder has been implemented in software running on commercial Digital Signal Processor (DSP) chips, removing the need to design complicated and expensive hardware as was the case with the previous generation of codes. The decoder time-tags the data frames, performs a buffered frame synchronization in the symbol domain (as opposed to the current bit domain synchronization), distributes the turbo coded frames among several DSPs working in parallel for decoding, and releases the decoded bits in the CCSDS Standard Formatted Data Units format. The decoder uses a stopping rule to detect convergence, decreasing the average number of decoding iterations required. The decoder currently decodes at rates up to 700 kbps (depending on frame length and SNR), and will increase in rate as DSP clock rates increase. The implementation will go operational in October, 2003.

1.0 INTRODUCTION

In 1993, a new class of error correcting codes was developed [1]. These codes provide up to a 0.8 dB improvement over the current best codes used by deep space missions. The Consultative Committee for Space Data Systems (CCSDS) has recommended a set of these codes for use in future missions. These codes are different from the turbo codes used for third-generation wireless applications.

The Jet Propulsion Laboratory (JPL) is developing a turbo decoder for use in the Deep Space Network (DSN) that JPL manages for NASA. This decoder not only decodes the turbo encoded data, but also performs time-tagging and frame synchronization as part of the decoding process. As opposed to previous decoder implementations, this decoder is implemented in software that runs on commercial Digital Signal Processor (DSP) chips. This provides a quicker development time, a cheaper production implementation, and an easy upgrade path for future DSPs.

This paper is divided into three parts. First, we describe turbo codes and how they are used. Next, we show the advantages of turbo codes versus the currently used deep space error correcting codes. Finally, we describe the design and implementation of the turbo decoder that is being developed by JPL.

2.0 DESCRIPTION OF TURBO CODES

Turbo codes are block codes. That is, the encoding is done on one block of data at a time. A transfer frame (as defined by [2]) is the basic block. As part of the transfer frame, the 16-bit Frame Error Control Field (FECF) at the end of the frame is required; the FECF is a Cyclic Redundancy Code (CRC). After the encoding is done, the frame synchronization marker is attached to the beginning of the block. This is different than the sequence for the concatenated convolutional / Reed-Solomon encoding that is currently done for deep space systems. In that case, the transfer frame is Reed-Solomon encoded (which is a block code), has the frame synchronization marker attached, and then is convolutionally encoded. The stages in generating the encoded output data stream are described below.
2.1 Frame Error Control Field

One feature of the Reed-Solomon coding is its use as a final "good frame/bad frame" indicator. The decoder either corrects the frame or indicates that it cannot decode the frame. The probability of its making an error in that process is very small (approximately the Bit Error Rate (BER) of the code divided by 16!, or $4.8 \times 10^{-16}$ BER), so it can be used as a reliable frame "goodness" indicator.

Unfortunately, by itself, a turbo code does not have that property. Its error floor on the decoded frame is higher than what the Reed-Solomon code provides. However, when using the FECF, as defined in the Transfer Frame specification [2] (where it is specified as optional), this problem is alleviated. The FECF is a 16-bit CRC that is used after the turbo decoding to detect remaining bit errors. When used as a frame "goodness" indicator, the probability of an erroneous frame being accepted as a good frame is very small (approximately the BER divided by $2^{16}$, or $1.5 \times 10^{-10}$ BER). The next revision of the telemetry channel coding recommendation [3] will specify that the FECF always be used with turbo encoding.

2.2 Encoding Algorithm

Turbo encoding is very straightforward. It uses two constraint length 4 convolutional codes to generate the encoded symbols. The process is as follows:

The transfer frame is input to the encoder. There are four frame sizes defined: 1784, 3568, 7136, and 8920 bits (a fifth frame size, 16384 bits, is defined, but the encoding parameters have yet to be specified). Note that the frame sizes correspond to the Reed-Solomon frame's data allocation (the same amount of data is sent per frame). The frame of data (of length $k$ bits) is input to one of the convolutional encoders. An interleaved (permuted) version of the data frame is input to the second encoder. For a code rate of $k/n$, the original bit and $n-1$ coder tap outputs are output as the encoded symbols (this means that the unencoded bits are available, if necessary). The CCSDS recommendation defines turbo codes for $n$ equal to 2, 3, 4, and 6. After the $k$ bits are clocked into the encoders, four flush bits are input; these flush bits clear out the coder memory (also known as trellis termination). Thus, for a $k$ bit frame, we get $n(k+4)$ symbols output. Fig. 1 shows the turbo encoder structure. For every input (either an input bit or a flush bit) to the shift registers, $n$ symbols are output. The output sequence is from top to bottom in the figure (e.g., for rate $1/6$, the output sequence is $0a, 1a, 2a, 3a, 1b, 3b$).

The interleaving is a fixed sequence, which is on a bit-by-bit level (as opposed to the Reed-Solomon interleaving on a byte-by-byte level). The interleaving is described algorithmically in [3].

2.3 Pseudo Randomization

If sufficient data transitions (which are required to lock up the receiving system) are not guaranteed, either by the modulation scheme or the data stream, then the CCSDS recommends that a Pseudo Randomizer be used on the encoded data. This means that a pseudo noise (PN) sequence is exclusive-ORed bit by bit with the encoded data. A PN sequence 255 bits long is repeated until it fills the encoded block. The sequence is defined by the following generator polynomial:

$$h(x) = x^8 + x^7 + x^5 + x^3 + 1$$

The sequence generator is always initialized to the all-ones state for the beginning of each encoded block. Although it is currently an option, it is expected that, in the next revision of the standard, pseudo randomization will be required.

2.4 Frame Synchronization Marker

Once the frame is encoded, a frame synchronization marker must be applied. This is similar in concept to what is currently done with the Reed-Solomon encoding. The only difference here is that the encoded block is already in the symbol domain, so the frame marker must be applied in the symbol domain. Thus, the 32-bit marker is appended as a 32n-symbol marker (e.g., 96 symbols for rate 1/3, 192 symbols for rate 1/6). The markers for the different code rates are defined in [3]. If pseudo randomization is used, the synchronization marker is not exclusive-ORed with the PN sequence.
3.0 PERFORMANCE GAIN

The performance of a code can be judged by the ratio of the energy-per-bit to the noise spectral density \( E_b/N_0 \) needed to achieve a desired probability of error \( P_e \); \( P_e \) is also known as the Bit Error Rate (BER). The lower the \( E_b/N_0 \) required for a given \( P_e \), the better the code's performance.

Fig. 2 provides a comparison of the two standard codes currently used for deep space and the new turbo codes. The current codes are the constraint length 7, rate 1/2 convolutional code (denoted as the \((7, 1/2)\) code), concatenated with the (255, 223) Reed-Solomon code, and the constraint length 15, rate 1/6 \((15, 1/6)\) convolutional code, also concatenated with the (255, 223) Reed-Solomon code. The comparison is for a frame size of 8920 bits, which corresponds to a Reed-Solomon interleave factor of 5. As can be seen, for a BER of \(10^{-6}\), the rate 1/6 turbo code provides approximately 0.8 dB improvement over the \((15, 1/6)\) concatenated code, the rate 1/4 code provides 0.6 dB improvement, the rate 1/3 code provides 0.4 dB improvement and the rate 1/2 is 0.3 dB worse. Comparing the \((7, 1/2)\) code with the turbo codes, we see improvements of 2.4 dB, 2.2 dB, 2.0 dB, and 1.3 dB for the rate 1/6, 1/4, 1/3, and 1/2, respectively.

4.0 TURBO DECODING

On the receiving side, the turbo decoder takes 8-bit quantized symbols from the receiver and produces the decoded bits, along with time-tag information. The process is described below.

4.1 Description

The turbo decoder actually encompasses several functions: time tagging, frame synchronization, pseudo randomization, turbo decoding, and CRC checking. More detail is available in [4].

Time tagging is accomplished by counting the cycles of the 10 MHz reference signal, using a 1 pulse per second (pps) timing reference to zero out the count (the 1 pps occurs on the second boundary). Each symbol clock, which clocks the
symbol into the decoder, latches this count. This process gives a 24-bit count of the 0.1 μsec of the current second. The I/O processing of the decoder adds the current second to the count. Thus, each symbol has a time tag associated with it.

Next, frame synchronization is performed. The frame synchronizer searches for the frame synchronization marker that was appended to the coded block. The frame synchronizer checks for both normal and inverted polarity in the marker; if it detects the inverted polarity, the encoded block is marked for inversion before being sent to the decoding task. The synchronizer can buffer a minimum of four frames. This allows the system to acquire synchronization and then apply it backwards to the previous frames, reducing loss of data during the lock up period. Also, due to the fact that the synchronization search is done in the symbol domain (as opposed to the bit domain as has always been done for convolutional/Reed-Solomon coding), the SNR that the synchronizer operates at is lower; this requires that multiple frames be summed to achieve high enough SNR for determining synchronization.

Each synchronized block is passed to a decoder element, along with a flag indicating whether or not the block is inverted. Unlike the convolutional/Reed-Solomon codes, turbo codes are not transparent; the inversion of a codeword is not a codeword. So, if indicated, the decoder must invert the block prior to decoding. Also, if pseudo randomization was applied to the codeblock, it must be removed.

The turbo decoder itself is an iterative decoder. Each of the two component codes (non-interleaved and interleaved) are alternately decoded, with the results being passed back and forth between the two. Eventually, the decoder produces an output. The decision on when to stop iterating can be achieved either by performing a fixed number of iterations or by using a metric to determine that the decoder has converged to a result. These methods are described in section 4.4. The data blocks are output to the project as Standard Formatted Data Units (SFDUs), a standard format, based on a CCSDS recommendation [5]. The format is defined in [6].

Since the decoder operates on blocks (frames), higher speed can be achieved by having multiple decoder elements. The total speed of the decoder is the product of the number of decoder elements and the average speed of an individual element. The current requirement for the first turbo decoder implementation is a 365 kbps rate.

The turbo decoder is implemented on commercial Digital Signal Processor (DSP) boards. There is a total of eight Texas Instruments (TI) TMS320C6000 family DSPs available for use on a board. The design of the DSP usage is provided in the next section.
4.2 Software System Architecture

The primary computational tasks of the decoder are frame synchronization, turbo decoding, and control and coordination. These tasks are divided among the eight DSPs as shown in Fig. 3. The Control DSP handles data transfer and coordination among the DSPs, and performs various other minor tasks. The Frame Sync DSP identifies the embedded synchronization markers to determine the location and polarity of each turbo coded block. These blocks are distributed to six Decoder DSPs, each of which runs identical software and performs the iterative turbo decoding. At the 200 MHz clock rate, each Decoder DSP can maintain a data rate of 50 kbits/second while performing ten iterations on each block. This gives a system throughput of 300 kbits/second at ten iterations, and this increases proportionally as the clock rate is increased, and as the average number of iterations is decreased with a stopping rule.

The basic skeleton of the turbo decoder algorithm is written in C to preserve readability of the code and to permit minor modifications; all the computation routines are written in optimized assembly. All the software for the Frame Synchronizer DSP is in assembly; the skeleton is not optimized but all the computation is. Software for the Control DSP is written in assembly language because it deals extensively with interrupts and hardware resources which higher level languages do not handle as readily; the assembly is not optimized because speed is not critical.

4.2.1 The Turbo Decoder DSPs

Turbo decoding is performed iteratively. A simple description will be provided here; details of the algorithm are available in [4]. In the first half iteration, an attempt is made to decode the first constituent convolutional code using its received data independent of information from the second code, using the BCJR soft-decision algorithm (named for Bahl, Cocke, Jelinek, and Raviv [7], and also known as the "forward-backward" algorithm). This algorithm works both forwards and backwards, through the block, performing iterative calculations. These are combined to form "reliabilities" representing the estimated probability that the jth bit is a binary 1. These probabilities are modified by subtracting a term common to both constituent decoders, and the resulting "extrinsic" information is "interleaved" (permuted according to the permutation algorithm) and used as an aid in decoding the second convolutional code during the second half iteration. A new set of extrinsic information is computed, de-interleaved, and passed back to the first decoder for use in the second iteration. This process is repeated for some number of iterations, and usually the message...
estimates converge to the correct decoded sequence. The number of iterations can be fixed, or determined by a "stopping rule" based on the reliabilities at the end of each iteration. Upon completion, a final estimate is computed.

While implementation of the algorithm is essentially straightforward, efficient memory use and numerical precision require consideration. Each DSP contains 64 kbytes of internal data RAM which it can access in a single clock cycle (technically, internal data RAM requires no wait states (assuming bank hits are avoided), so while load instructions involve four delay slots, the carefully written program executes at full speed). Megabytes of off-chip memory are available, but they impose 16, 18, or more wait states, entirely wasting that many clock cycles. Thus it is very desirable to perform all computation using internal memory. Simultaneously, the data for future computations can be transferred from external to internal memory by DMA in the background, without imposing any significant penalty on the CPU speed. Because the CCSDS codes use 16-state (constraint length 4) convolutional encoders, decoding a codeword of $N$ message symbols requires regular access to well over $16N$ intermediate values. The internal memory is too small to hold this many 16-bit quantities, even for the smallest turbo codes. The solution to this memory problem is to break each block of data into a sequence of "windows".

All computation is done using fixed point arithmetic for speed. The received symbols are scaled as required by the first step of the decoding algorithm and quantized to preserve about three bits of "soft" information by setting the nominal BPSK values to $\pm 10/\sigma$, where $\sigma$ is the standard deviation of the symbol noise. The Block V receiver must do at least part of this scaling because $\sigma$ is not known to the decoder; scaling by constant factors can be done by the receiver, the decoder, or both. To avoid quantization losses at both the receiver/decoder interface and after scaling by the decoder, it can be shown that the decoder's scale factor should be close to the reciprocal of an odd integer.

As decoding proceeds, the extrinsic information exchanged between decoders usually grows in magnitude with each iteration, and this must be restricted to prevent numeric overflow. There are anecdotal research results that show that "clipping" the extrinsic values to some limit also improves decoder performance, and increases immunity to outlying received noisy symbols (due, perhaps, to non-Gaussian data errors). For these reasons, clipping of the extrinsics is implemented, with a programmable level to permit making trades among performance, required renormalization rate, and immunity to outlying symbol values.

4.3 Decoder Speed

Decoder speed has several potential bottlenecks: the symbol input, the frame synchronizer, and the decoder elements. All of the processing are functions of the DSP clock speed. Current development is on 200 MHz boards; all of the numbers quoted in this section are for these boards. (The implementation in the DSN will use 300 MHz boards, so a 1.5 times speed improvement is expected).

Input rates of 16 MHz have been successfully demonstrated. This would give a symbol input rate of 16 Msps, which translates to 2.67 Mbps for a rate 1/6 code. It is expected that higher speed processors will be able to handle higher rates, up to a maximum input rate of 26.4 Msps (the maximum output rate of the receiver). However, even at the current speed, the symbol input will not be a bottleneck.

The frame synchronizer must be run in serial with the data stream (as opposed to the parallelism that can be achieved with the decoder elements). Current projections from the prototype indicate that it runs at a rate of 4.5 Msps for the worst case (192 symbol frame marker and an 8920 bit frame). This corresponds to a bit rate of 750 kbps for a rate 1/6 code. This will increase with faster speed processors. In addition, if a higher throughput is needed, there are some additional tricks that can be done for higher symbol rates (e.g., checking every other frame).

The decoder elements are the main bottlenecks in the speed equation. There are three ways to increase the speed of the decoding. First, the processor speed can be increased; the decoder speed is basically linear with the processor speed (doubling the speed doubles the decoding rate). This is definitely a viable option; as mentioned earlier, while the current development has been done on 200 MHz DSPs, 300 MHz DSPs will be fielded, and TI has announced part numbers for 600 MHz parts and has promised a 1 GHz DSP in the future. Secondly, more DSPs can be added to the system.

The final way to increase the decoding rate is to use stopping rules in the decoding process. As described earlier, the decoder is an iterative process that can run for a fixed number of iterations, or can be stopped when convergence is detected. The method for determining the convergence is called the stopping rule. Stopping rules are described below.
4.4 Stopping Rules

When decoding a block of data, the reliabilities of the message bits generally improve with each iteration. Depending on the particular values of the received noise, the estimated message is often entirely correct after a few iterations, but may not be until ten or more, or never, in which case a decoding error is unavoidable. One could perform a fixed number of iterations on each block (typically ten), knowing that most decodable blocks will be correctly decoded by then. A superior method is to iterate only until the decoder is sufficiently "confident" in its estimates, or until some maximum number of iterations is reached. In this way, most blocks are decoded in a few iterations, and the time saved can be used to perform extra iterations on the difficult blocks, resulting in a better decoder, or a faster one, or both.

This technique requires a stopping rule to determine when a sufficient confidence has been reached. Balancing implementation issues against performance, we consider Rule $S_2$ from [8] which stops decoding when all the extrinsic values have a magnitude exceeding some fixed threshold $\theta$. This test is performed after odd half iterations, and when satisfied, decoding is stopped at the end of the iteration (i.e., half an iteration later).

With a stopping rule, any two of the four parameters of frame error rate (FER), decoder speed, SNR, and threshold $\theta$ can be determined from the remaining two. In Fig. 4, frame error rate is plotted parametrically against decoder speed (measured as the reciprocal of the average number of iterations performed times the speed per iteration), as the SNR and $\theta$ are varied. For a fixed SNR, we see that as $\theta$ is reduced from infinity, a marked increase in speed is realized with virtually no penalty in FER, then there is a "knee" in each curve, after which the FER increases rapidly with modest increases in speed. By choosing a $\theta$ of 100 (in its essentially arbitrary units), the decoder operates near these "knees", achieving the best speed consistent with a minimal FER penalty.

In Fig. 5, speed is plotted parametrically against SNR for four of the CCSDS codes as $\theta$ is varied, with the FER fixed at $10^{-4}$. This shows the SNR required to achieve a particular data rate. The expected operating point is the knee of the curve. However, the figure shows that an increase in the data rate can be achieved at the cost of a higher threshold SNR.

5.0 IMPLEMENTATION STATUS

A prototype has been developed and the implementation work for installing into the DSN has started. The prototype uses two boards with four DSPs on each to implement the turbo decoder. This implementation was developed under the Telecommunications and Mission Operations Directorate (TMOD) Technology (TMOT) program. This decoder has successfully interfaced with the DSN's Block V Receiver (BVR) in the Telecommunications Development Lab (TDL). The processor speed is 200 MHz. It implements the symbol input, the fractional time tagging, the frame synchronization and six decoder elements. The decoder element speed is about 54 kbps (without stopping rules - 10 iterations are done), for an aggregate rate of 324 kbps, far exceeding its initial goal of demonstrating a 250 kbps decoding rate. Stopping rules have been implemented and are showing the expected speed improvement. Extensive BER testing shows excellent agreement with theory.

The turbo decoder will be installed into the new Downlink Tracking and Telemetry (DTT) subsystem that is being delivered as part of the DSN's Network Simplification Project (NSP). Specifically, the decoder boards will be installed in the new Telemetry Processor (TLP) and controlled by the Downlink Channel Controller (DCC). This equipment will be installed in the DSN in the 2002-2003 time frame. The decoding capability will be operational at all sites by October 2003. Implementation across the DSN will be staggered over the year 2003, so capability to support missions will be in place at some antennas before October. Full compatibility testing capability will be available in January 2003 and limited compatibility testing will be available prior to that date, using prototype and first production units. Two missions, MESSENGER and STEREO, have already committed to using turbo coding.

6.0 CONCLUSION

The turbo decoder being developed at JPL for deep space missions has been described. The decoder allows missions to use the new CCSDS turbo codes, which provide up to 0.8 dB improvement over the best coding that is currently used. The initial implementation will support at least 706 kbps and will be available for use at all DSN antennas by October 2003.
REFERENCES


ACKNOWLEDGEMENTS

The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

We would like to acknowledge all of those who have contributed to the design and implementation of the turbo decoder: Samuel Dolinar, Daruish Divsalar, Fabrizio Pollara, Valerie Stanton, David Wotola, Victor Chen, Jim Weese, Adina Matache, and Todd Chauvin.