

Implementation of a 3GPP Turbo Decoder on a Programmable DSP Core

James G. Harrison
3DSP Corporation

*As presented at the
Communications Design Conference,
San Jose, California, October 2, 2001.*

INTRODUCTION

Third generation (3G) portable wireless applications will require greater data rates at lower channel SNR than ever before. To enable reliable data transmission for these applications more advanced error correcting techniques are required. An error correction technique known as Turbo Coding has greater error correction capability than any other known code [1], and it is specified as one of the coding options in the 3GPP standard for the European 3G Universal Mobile Telecommunications System (UMTS) [2].

This paper describes the techniques required to implement a turbo decoder on a fixed-point programmable DSP. Various mathematical and approximation techniques are described that dramatically reduce the computations required. Specifically, the implementation of a turbo decoder for the 3GPP standard for UMTS is described, but these techniques can be applied to any turbo decoder. This turbo decoder has been implemented on the 3DSP Corp. SP-5 SuperSIMD™ DSP core, enabling real-time UMTS data rates. Results of simulations are presented, showing the number of cycles and amount of memory required, and the data rates supported for various processor clock rates.

TURBO CODES

Prior to the first publication describing turbo codes in 1993 [3], the most powerful error correcting codes were convolutional encoders paired with Viterbi decoders. Convolutional codes are theoretically capable of allowing a communication system to reach the Shannon limit, which is the theoretical limit of SNR for error free communication over a given noisy channel. However, Viterbi decoders grow exponentially in complexity as their error correction capability is increased, making their practical limit 3 to 6 dB away from the Shannon limit for practical hardware/software implementations. The first paper describing turbo codes showed a practical decoder that achieved a performance of 0.7 dB from the Shannon limit, far surpassing the performance of a convolutional-encoder/Viterbi-decoder of similar complexity.

Turbo codes have therefore generated great interest in the communications field. Intense academic research has uncovered the mathematical reasons for turbo codes' phenomenal performance, and hardware and software are just starting to appear in the marketplace. However, decoding algorithms have not yet reached the maturity and open availability that Viterbi algorithms enjoy, so implementing a turbo code is not a trivial exercise.

3GPP TURBO CODE

The International Telecommunications Union (ITU) has published a vision for 3G mobile systems called IMT-2000. A European 3G system called UMTS (Universal Mobile Telecommunications System) has been proposed that follows the IMT-2000 recommendations. The 3GPP (Third Generation Partnership Project) is a European industry group that is writing the standards for UMTS and getting them adopted by the European Test Standards Institute (ETSI). These will be hereafter referred to as the “3GPP standard.” The 3GPP standard specifies the turbo encoder in great detail but does not specify the decoder, leaving that choice up to the designer.

3GPP Turbo Encoder

The encoder, shown in Figure 1, consists of two simple 8-state convolutional encoders (the “constituent” encoders). The data bit stream goes into the first constituent encoder that produces a parity bit for each input bit. The data bit stream also goes through an interleaver which scrambles the bit ordering and then to the second constituent encoder which produces a parity bit for each input bit. The interleaver ordering remains the same from frame to frame. For 3GPP, the interleaver can be anywhere from 40 bits to 5114 bits long. The data sent across the channel is the original bit stream, the parity bits of the first constituent encoder, and the parity bits of the second constituent encoder. So the entire turbo encoder is a rate 1/3 encoder. The encoder is very simple so its implementation will not be discussed in this paper.

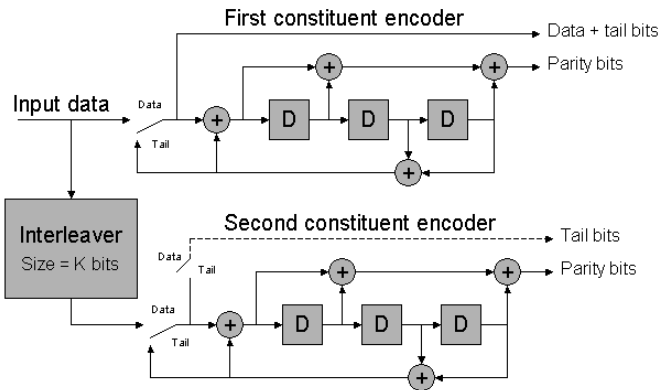


Figure 1. 3GPP Turbo Encoder

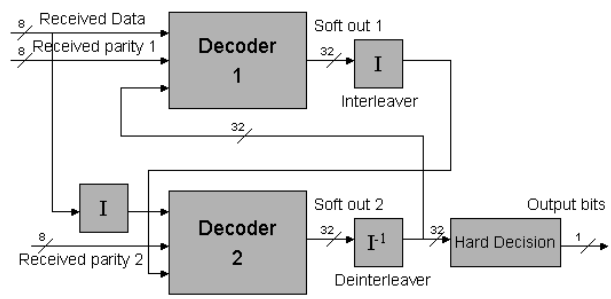


Figure 2. Generic Turbo Decoder

3GPP Turbo Decoder

Turbo decoders operate in an iterative fashion as shown in Figure 2, with two decoder blocks corresponding to the two constituent encoders. The first decoder block makes an estimate of the probability for each data bit as to whether it is a 1 or a 0 by operating on the received data and parity bits (soft values from demodulator) produced by the first constituent encoder. This estimate is then sent to the second decoder block along with the interleaved received data and the parity bits (soft values from demodulator) produced by the second constituent encoder. This process of two passes thru the decoding algorithm is considered to be one iteration and is repeated for a fixed number of iterations, or until some external mechanism determines that no further iterations will improve the BER for that frame. After all iterations are complete, the original data bits are recovered by making a hard decision on the last soft output.

The algorithm within the two decoder blocks must operate on soft inputs (the demodulator outputs and the probability estimates) and produce soft outputs. The MAP (Maximum A-posteriori Probability) algorithm produces the best BER performance for these requirements.

MAP Algorithm

The MAP is a trellis decoding algorithm, like the Viterbi algorithm. Each constituent encoder in the 3GPP turbo encoder can be defined by a trellis diagram with 8 states in the vertical axis, and $K+3$ time intervals along the horizontal axis as shown in Figure 3. K is the length of the interleaver and the three extra time intervals are needed for the trellis termination (getting the encoder back to state 0). The trellis diagram is simply a state diagram with a time axis. Note that after three time intervals the branches in the trellis repeat themselves, so only one time slice of the trellis is needed to define the entire trellis.

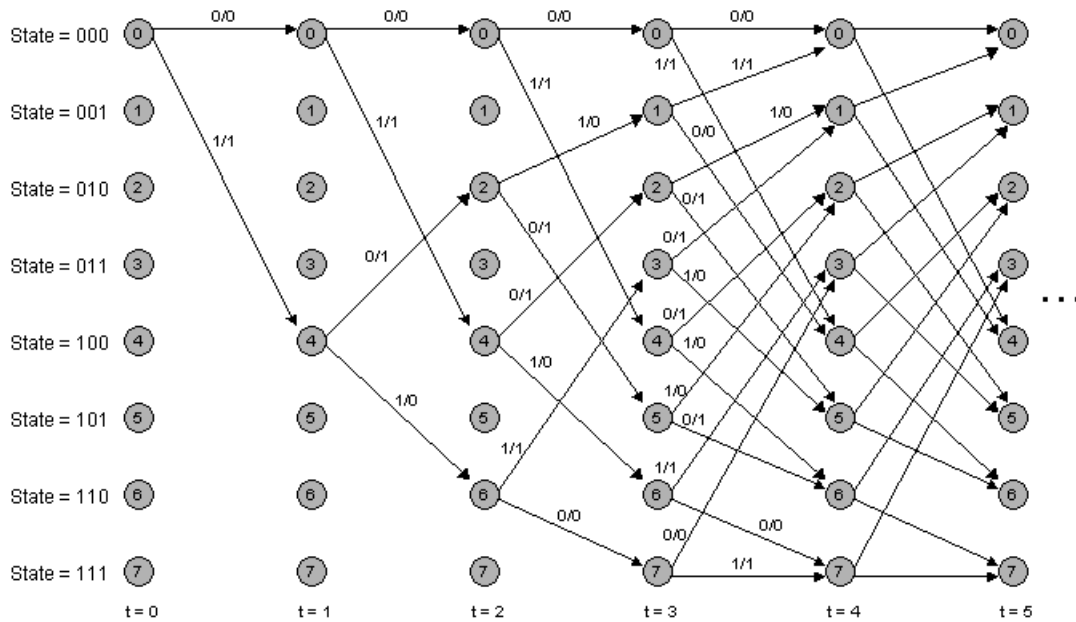


Figure 3. Trellis Diagram of One Constituent Encoder

The computations required for the MAP algorithm are much more complicated than the Viterbi algorithm, and require more memory storage. The computations for MAP are summarized as follows (remember that these are the computations for only one decoder for one iteration):

- 1) Compute and store the branch metrics (called gamma) for all branches in the trellis. The branch metrics are the exponential of the negative of the distance between the hard encoder values and the soft received values from the demodulator, divided by the channel noise variance, times the probability estimate from the previous decoder.
- 2) Perform a forward recursion on the trellis by computing alpha for each node in the trellis. Alpha is the sum of the previous alpha times the branch metric along each branch from the two previous nodes to the current node.

- 3) Perform a backward recursion on the trellis by computing beta for each node in the trellis. The computation is the same as the alphas, but starting at the end of the trellis and going in the reverse direction.
- 4) Compute the log likelihood ratio (LLR) (λ) for each t . This is the sum of the products of the alphas, betas, and gammas for each branch at time t that is associated with a 1 in the encoder, divided by the sum of the products of the alphas, betas, and gammas for each branch at time t that is associated with a 0 in the encoder.
- 5) Compute the “extrinsic information” that is to be fed to the next decoder in the iteration sequence. This is the LLR minus the input probability estimate.

This sequence of computations is repeated for each iteration by each of the two decoders. After all iterations are complete, the decoded information bits can be retrieved by simply looking at the sign bit of the LLR: if it is positive the bit is a one, if it is negative the bit is a zero. This is because the LLR is defined to be the logarithm of the ratio of the probability that the bit is a one to the probability that the bit is a zero.

Due to the complexity of MAP, if the computations were performed as-is, we would not be able to achieve real-time operation. Therefore, some simplifications, approximations, and tricks can be used to dramatically reduce the computations required.

DECODER SIMPLIFICATIONS

Log-MAP Algorithm

The first simplification to the computations of the MAP decoder algorithm is to operate in the log domain. This converts all multiplications to additions, divisions to subtractions, and eliminates exponentials entirely, without affecting BER performance.

Since our goal is to perform the decoder on a DSP, one might question why this is necessary, since a DSP can perform a multiplication in the same time as an addition. There are other reasons to go to the log domain: logs and exponentials can be eliminated, and the numbers do not grow as rapidly. Because the forward and backward recursions require successive multiplications by numbers less than one, even 32-bit floating-point numbers will underflow unless they are scaled. Scaling requires additional operations that will slow down the turbo decoder. By going to the log domain no scaling is needed for 32-bit fixed-point numbers, and minimal scaling can be employed to utilize 16-bit or even 8-bit numbers. Also, avoiding the use of multiplications will reduce the power consumption of the processor.

Another advantage of going to the log domain is that the desired output of the algorithm, the log likelihood ratio or LLR, is in the log domain so it is automatically produced without having to actually take a logarithm. The LLR is the ratio of the logs of the probabilities that the particular data bit is a 1 or a 0.

One awkward computation to result from this, however, is the log of the sums of exponentials:

$$\ln(e^a + e^b + e^c + \dots)$$

This problem can be simplified by using the Jacobian formula:

$$\ln(e^a + e^b) = \max(a,b) + \ln(1 + e^{-|a-b|})$$

The max function is a single instruction in DSPs, but this still leaves the second term to be computed. However, it has been shown [4] that this term can be implemented with a lookup table of only eight entries with no degradation of BER performance, and the input to the lookup table is one dimensional, $|a - b|$. This entire operation, max plus table lookup, is often called the max* function to distinguish it from a simple max.

Max-Log-MAP Algorithm

A further simplification can be made by simply ignoring the second term of the max* altogether and just using the max instruction. This reduces the amount of instructions required but also reduces the BER performance of the decoder since the computations are not as accurate. One must be careful when looking at the performance results of turbo decoders since sometimes it is claimed that a MAP algorithm is being utilized when in fact they are doing the max-log-MAP to get lower cycle or MIPS numbers. However, to achieve the performance advantage that MAP offers, one must use max*. But, so that our performance numbers can be compared with other processors on an equal basis, we will give the cycle numbers for the max-log-MAP along with log-MAP.

Results for Max-Log-MAP

A complete turbo decoder, including the two max-log-MAP decoders for each iteration, was written in assembly language for the 32-bit fixed-point 3DSP Corp. SP-5 DSP and run on a cycle accurate simulator. The SP-5 is a super-scalar DSP that allows two instructions to be processed in each clock cycle. This can potentially double the processing power over a conventional DSP. The soft outputs were compared with a C simulation to verify mathematically correct operation. The results are shown in Table 1. Performance is given in total cycles and cycles/bit/iteration (c/b/i), where one iteration includes two max-log-MAP decoders. Cycles/bit/iteration equals the number of cycles from the simulator divided by the number of information bits, divided by the number of iterations. Also the data rate for a 100 MHz clock is given in kilo-bits per second (kbps). K is the frame and interleaver size in bits.

Note that these are not theoretical counts of operations, but are the actual outputs of the simulator running a complete, functioning turbo decoder with the entire loop overhead and register set-up required in a real program. As the frame size is increased, the overhead becomes a smaller percentage of the operations so the c/b/i approaches that required for just the computations themselves. So the processing required for an individual max-log-MAP is just over 50 cycles. The assembly program size is 415 words. The data memory required is a direct function of K and is $17*(K+3)$ words.

K	1 Iteration			4 Iterations			8 Iterations		
	Cycles	C/b/i	Kbps*	Cycles	C/b/i	Kbps*	Cycles	C/b/i	Kbps*
50	5836	116.7	857	23035	115.2	217	45967	114.9	109
640	69556	108.7	920	276145	107.9	232	551597	107.7	116
1024	111028	108.4	923	440881	107.6	232	880685	107.5	116
1620	175396	108.3	923	696565	107.5	233	1391457	107.4	116

*Data through-put for a 100 MHz processor clock.

Table 1. Performance of 32-bit Fixed-Point Max-Log-MAP Turbo Decoder on SP-5

Results for Log-MAP

To go to the log-MAP algorithm, which will give better BER performance, it is necessary to replace all of the max operations in the max-log-MAP with max* operations. This will add six instructions (worst case) to every max as shown in Figure 4.

- 1) compute $c = a - b$
- 2) compute absolute value of c
- 3) is c greater than the number of table entries?
 - If yes
 - Use $\max(a, b)$
 - If no
 - 4) add c to table base address
 - 5) perform table lookup
 - 6) add table value to $\max(a, b)$

Figure 4. MAX* Operations

Since there are 30 max operations in the max-log-MAP algorithm, this will add a total of 180 instructions to each decoder, or 360 instructions to each iteration. The SP-5 is a super-scalar processor and can perform 2 instructions per cycle, so this would bring the cycles/bit/iteration number up to 287.4 in the best case in Table 1.

To improve this performance, the configurable architecture of the SP-5 can be modified using 3DSP's HiFI tool to create new instructions for the processor [5]. The max* operation can be reduced to three new instructions which will reduce the log-MAP algorithm to 197.4 cycles/bit/iteration.

TURBO DECODER PARALLEL PROCESSING

In order to achieve further improvements in speed, it is necessary to go to parallel processing. Two methods of DSP parallel processing are discussed here: super-scalar operation and SIMD (Single Instruction Multiple Data) operation.

Super-Scalar

Super-scalar operation is a form of parallel processing in that it allows two instructions to be processed in each clock cycle. The SP-5 has two independent pipelines with automatic data and resource hazard checking that allows a potential doubling of processing power. The effects of the super-scalar operation in the SP-5 are already taken into account in the cycle counts in Table 1.

SIMD (Single Instruction Multiple Data)

The SP-5 is also capable of SIMD operation. This allows each 32-bit operational unit (adders, multipliers, logical units, shifters) within the DSP to perform two 16-bit computations simultaneously, or four 8-bit computations simultaneously. The SIMD capability could be taken advantage of in two ways:

- 1) Compute two or four frames at the same time. This would double or quadruple the overall throughput, but would also double or quadruple the delay because two or four frames would have to be buffered up before they are processed.
- 2) Parallel the computations within a frame to perform a single frame twice or four times as fast. This is the approach that will be considered in this paper because it has the advantage of not increasing the delay of the algorithm, although it requires more extensive modification of the code.

Word Sizes

In order to utilize SIMD, the computations must be done in only 16-bits or 8-bits. Therefore it is necessary to see if the performance of the algorithm can be maintained with these word sizes. The difficult values are the forward (alpha) and backward (beta) recursions

The forward and backward recursions are performed by successively adding the branch metrics as the trellis is traversed in time. Therefore the alphas and betas will grow as we move along the trellis, and the longer the trellis the larger they will grow. The author has also observed empirically that the alphas and betas grow larger with high SNR's than with low SNR's. Experimentally the author has determined that 32 bits are sufficient to handle alphas and betas for a frame and interleaver size up to the maximum of 5114 bits for 3GPP with an SNR up to 10dB.

Normalizing

So, if we want to go to 16-bit or 8-bit SIMD to double or quadruple our turbo decoder data rate, we must do something about the alphas and betas that require 32 bits. Looking at the algorithm it can be seen that the exact values of these quantities are not needed -- it is their relative values that are important. In computing the alphas and betas many max operations are performed, which only depend on relative relationships. And the final LLR computation is dependent only on the difference of numerator and denominator. Also these relative relationships remain close together, especially for low SNR's where no trellis path has a great deal more weight over other paths.

Therefore, as long as the alphas or the betas for all 8 states at a given time are scaled the same amount, the relative weight relationships between the paths through the trellis will be maintained. A simple way to accomplish this is at each point in the trellis, compute the 8 alphas (or betas), determine the maximum among these 8, then subtract the maximum from the 8 alphas (or betas). This prevents the maximum alpha and beta from ever growing but maintains the relative distances from the other alphas and betas. There is absolutely no effect on the BER performance at 16-bits with normalization. But, there is some degradation of performance at 8-bits because the differences between values sometimes exceeds 256.

The problem with normalization is that finding the maximum of 8 alphas or betas and then subtracting this value from 8 alphas or betas each time through the forward and backward recursion loops doubles the number of operations in the forward and backward recursions, eliminating the advantage of going to 16-bit SIMD.

An alternative to normalizing is desired that requires fewer operations. By creating custom instructions for the SP-5, a method of performing the log-MAP and max-log-MAP algorithms has been found that requires no normalization for 16-bit operation, thus allowing the processing speed of the algorithm to be doubled. For 8-bit operation, however, standard normalization is still required because of the small dynamic range.

Results for SIMD

The results for SIMD operation on the SP-5 are shown in Table 2. As can be seen, the cycles/bit/iteration for 16-bit SIMD is half what it is for 32-bit operation. The cycles/bit/iteration for 8-bit SIMD is approximately the same as for 16-bit SIMD because of the extra normalization operations.

Cycles/bit/iteration	Max-Log-MAP	Log-MAP
32-bit	107.4	197.4
16-bit SIMD (w. custom instr.)	53.7	113.7
8-bit SIMD (w. normalization)	64	110

Table 2. Performance of 16-bit and 8-bit Turbo Decoder using SIMD on SP-5

CONCLUSION

Utilizing all of the various techniques described in this paper to speed up the operation of a turbo decoder allows a single core programmable DSP to operate at real-time data rates in a real system, such as that specified by the 3GPP standard. The data rates handled by the various versions of MAP decoder algorithms with 8 iterations are shown in Table 3 for various clock rates achievable by current semiconductor processes on the SP-5. This will easily handle the currently planned UMTS (3GPP) data rates that range from 4 kbps to 384 kbps. The future UMTS goal of 2 Mbps could be handled by a multi-core design.

Max-Log-MAP Decoder	100 MHz	150 MHz	200 MHz	250 MHz
32-bit SP-5	116 kbps	175 kbps	233 kbps	291 kbps
16-bit SIMD SP-5	233	349	466	582
8-bit SIMD SP-5	195	293	391	488
Log-MAP Decoder				
32-bit SP-5	63	95	127	158
16-bit SIMD SP-5	110	165	220	275
8-bit SIMD SP-5	114	170	227	284

Table 3. Data Rate of Turbo Decoder (8 iterations) for Various Clock Rates on SP-5

REFERENCES

- [1] B. Vucetic, J. Yuan, “*Turbo Codes*”, Kluwer Academic Publishers, 2000
- [2] 3rd Generation Partnership Project (3GPP) Technical Specification Group: Universal Mobile Telecommunications System (UMTS); Multiplexing and Channel Coding (FDD), TS 25.212 v3.4.0
Also referred to as:
European Telecommunications Standards Institute (ETSI) TS 125 212
- [3] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo Codes”, in Proc. 1993 Inter. Conf. Commun., 1993, pp. 1064-1070
- [4] P. Robertson, E. Villebrun and P. Hoeher, “A Comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain”, Proc. ICC’95, Seattle, June 1995
- [5] “*HiFI User’s Guide*”, 3DSP Corporation, 16271 Laguna Canyon Rd., Irvine CA, 92618 USA, 2000

3DSP Corporation 16271 Laguna Canyon Road Irvine CA 92618 USA
(949) 435-0600 www.3dsp.com