Project	**IEEE 802.16 Broadband Wireless Access Working Group <http://ieee802.org/16>**
Title	**Turbo Product Code FEC Contribution**
Date Submitted	**2000-06-14**
Source(s)	David Williams Advanced Hardware Architectures 2365 NE Hopkins CT. Pullman, WA 99163 USA Voice: 509 334 1000 Fax: 509 334 9000 mailto:davew@aha.com
Re:	This is a response to the BWA FEC call for contributions IEEE 802.16.1p-00/06
Abstract	This document expands and clarifies the IEEE 802.16.1pc-00/32/rl submission, comments are provided with respect to the IEEE 802.16.1pc-00/31, IEEE 802.16.1pc-00/33 submissions and finally provides additional coding options based on Block Turbo Codes (aka Turbo Product Codes).
Purpose	This submission is offered to the IEEE 802.16 group as a means of more accurately understanding FEC alternatives for BWA Burst Communications including complexity and code rate tradeoffs. Large block coding options for the Mode A downlink continuous stream option are presented.
Notice	
Release	
Patent Policy and Procedures	

Turbo Product Code FEC Contribution

David Williams

Advanced Hardware Architectures

Overview

The following is offered to provide additional insights into Block Turbo Codes as well as to provide a summary of subsequent sections of this document.

- Generic Block Turbo Code (aka TPC) architectures for encoder and decoders are non-proprietary and can be supported by various suppliers who may or may not choose to use proprietary decoder algorithms. Equivalent coding performance from two potential suppliers (IEEE 802.16.1pc-00/32/rl) has been shown. The use of product codes were described in published literature in 1954 [1] and the use of iterative decoding techniques for these codes were described in published papers and at least one textbook [2] in the early 1980's.
- Block Turbo Codes are a proven technology with hardware that has been in the field for over a year. A single chip decoder ASIC has been commercially available since November 1998 [3]. Two hardware evaluation platforms and a commercially available modem (Comtech Communications Corp.) which utilize this technology are on the market.
- Block Turbo Codes have been available as licensed cores since December 1999 [4].
- Block Turbo Codes are capable of significantly outperforming the FEC coding that has been proposed in other 802.16 submittals.
- Block Turbo Code simulations have been verified with actual hardware that has been verified by independent third parties. It has been shown that simulations match hardware performance within normal measurement accuracy.
- In addition to both software simulations and hardware verification, a union-bound based analysis has been generated which supports the simulation and hardware results [5]. A brief description of this analysis is included later in this paper. This analysis can be used to predict code performance to arbitrarily low BERs. The accuracy of these predictions has been verified with actual hardware measurements for selected codes down to approximately 10^{-11}.
- Block Turbo Code encoders/decoders are very flexible. A single unified encoder/decoder design can support data block sizes from a few bytes up the maximum size chosen in increments of 1 bit. These same encoders/decoders can also support a wide range of code rates typically from about rate 1/5 to as high as rate 0.98 (large block codes).
- Encoder complexities for Block Turbo Codes are low (in the range of 10K gates), are non-proprietary and are constructed from Hamming and/or parity codes. Memory requirements are less than 500 bits for the codes proposed in IEEE 802.16.1pc-00/32/rl and approximately 1Kb for the higher complexity/performance codes that are introduced later in this submittal. Latency through such an encoder is less than a few bit periods at the highest data rates.

(Overview continued)

- Decoder complexities for Block Turbo Codes are higher than for Reed-Solomon based concatenated codes. This increase is offset by the increased performance available or can be traded off against reduced complexity in other system level components such as lower power amplifier requirements, smaller antennas, higher receiver noise figures, etc. The increase in complexity is estimated to be less than 5% of the total system complexity. Decoder complexities of less than 150 Kgates can be achieved that support both the worse case downlink and uplink decoder requirements (240 Mbits/sec coded data rate) for the codes proposed in IEEE 802.16.1pc-00/32/rl.
- If deemed necessary, reduced decoder complexity can be achieved with less decoder iterations. Reducing decoder iterations from 5 iterations to 4 iterations will typically degrade coding gain by only 0.1 dB. Reducing decoder iterations from 5 iterations to 3 iterations will typically degrade coding gain by 0.3 dB. In both cases, coding gain can still significantly outperform the FEC recommendations of IEEE 802.16.1pc-00/31 and IEEE 802.16.1pc-00/33. Reductions in decoder complexity can be as high as 30% with such tradeoffs.
- Decoder complexity will scale with improvements in process technology. The next mainstream CMOS process technology will enable a significant reduction in gate count complexity since higher clock speeds allow simplification of the circuitry.
- In the FEC submittal, IEEE 802.16.1pc-00/31, in the last paragraph of the introductory section, the author incorrectly compares Turbo Convolutional Codes with Block Turbo Codes. Turbo Convolutional Codes have a very low minimum distance, which typically results in few bit errors per block. Block Turbo Codes with high minimum distances do not exhibit this trait at the bit error rate operating points of interest. This can be supported with simulation and hardware results.
- In this joint submittal, IEEE 802.16.1pc-00/32/rl, the co-author suggests the use of a diagonal method of code shorting. While this method may be valid, this submittal does not support this method of code shortening for the following reasons:
  - Traditional row and column shortening is non-proprietary. There may be unknown/specified IP issues with a diagonal shortening approach.
  - Traditional row and column shortening is very simple to understand, document and implement.
  - Existing TPC solutions available on the market, including an available core generator do not support diagonal shortening.
  - From the simulation results presented in IEEE 802.16.1pc-00/32/rl, there is no evidence of any performance advantage.
- This proposal recommends the inclusion of programmable CRC based block error detection. The implementation cost (gate count) is low and the impact on coding gain is typically less than 0.1 dB. Additional information and analysis on this topic is contained later in this document.

(Overview continued)

- This proposal adds additional Block Turbo Code options for use in the Mode A continuous downstream case. These codes are 16 Kbits in size (coded bits) and will provide Eb/No capability as low as 1.8 dB for a rate 0.45 code and an Eb/No of less than 4 dB with a code rate of 0.88. This performance is significantly higher than the codes proposed in IEEE 802.16.1pc-00/33 and for the high rate code will provide close to a 10% payload increase to the end user which will provide a continual payback in revenue for the service provider utilizing this code. A rate 0.984 code is also proposed which can provide over 4 dB of coding gain for situations where link margins can support this very high rate code. Implementation complexity and other issues are described later in this document.

Turbo Code Description

The Block Turbo Code is a Turbo decoded Product Code (TPC). The idea of this coding scheme is to use well-known product codes in a matrix form for two-dimensional coding, or in a cubical form for three dimensions.

The matrix form of the two-dimensional code is depicted in figure 1. The k_x information bits in the rows are encoded into n_x bits, by using a binary block (n_x,k_x) code. The binary block codes employed are BCH-codes (Bose-Chaudhuri-Hocquenghem), of which the Hamming codes are a particular case (one-error-correcting BCH codes).

The redundancy of the code is $r_x = n_x - k_x$ and d_x the Hamming distance. After encoding the rows, the columns are encoded using another block code (n_y,k_y), where the check bits of the first code are also encoded. The overall block size of such a product code is $n = n_x \times n_y$, the total number of information bits $k_x \times k_y$, the code rate is $R = R_x \times R_y$, where $R_i = k_i/n_i$, i=x,y. The Hamming distance of the code is $d = d_x \times d_y$.

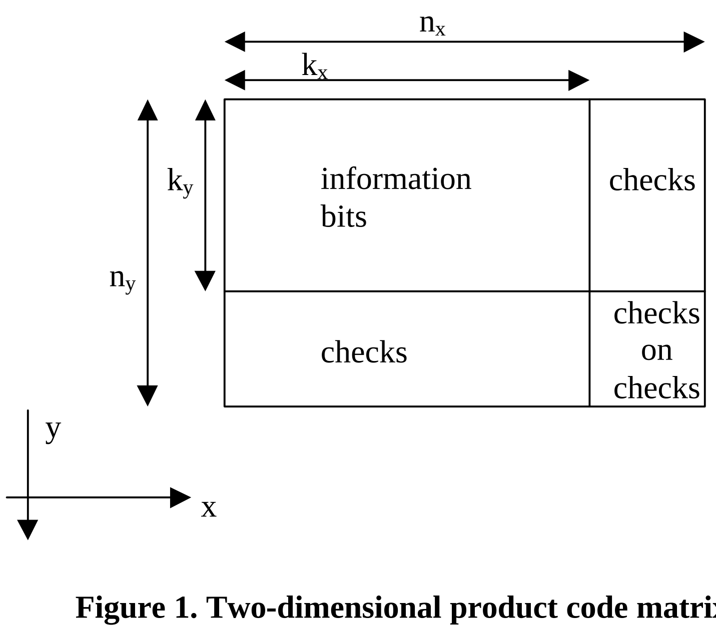


Figure 1. Two-dimensional product code matrix.

Encoding

The encoder for a Block Turbo Code (BTCs) has near zero latency, and is constructed of linear feedback shift registers (LFSRs), storage elements, and control logic. Encoding of a product code requires that each bit be encoded by 2 or 3 codes.

The constituent codes of BTCs are extended Hamming or parity only codes. Table 2 gives the generator polynomials of the Hamming codes used in BTCs. For extended Hamming codes an overall parity check bit is added at the end of each codeword.

Table 1 Generators Polynomials of Hamming Codes

N	K	Generator
7	4	$x^3 + x + 1$
15	11	$x^4 + x + 1$
31	26	$x^5 + x^2 + 1$
63	57	$x^6 + x + 1$
127	120	$x^7 + x^3 + 1$

Fig. 2 shows an example LFSR to encode the (16,11) extended Hamming code. Note that the overall parity bit is encoded in a separate register.

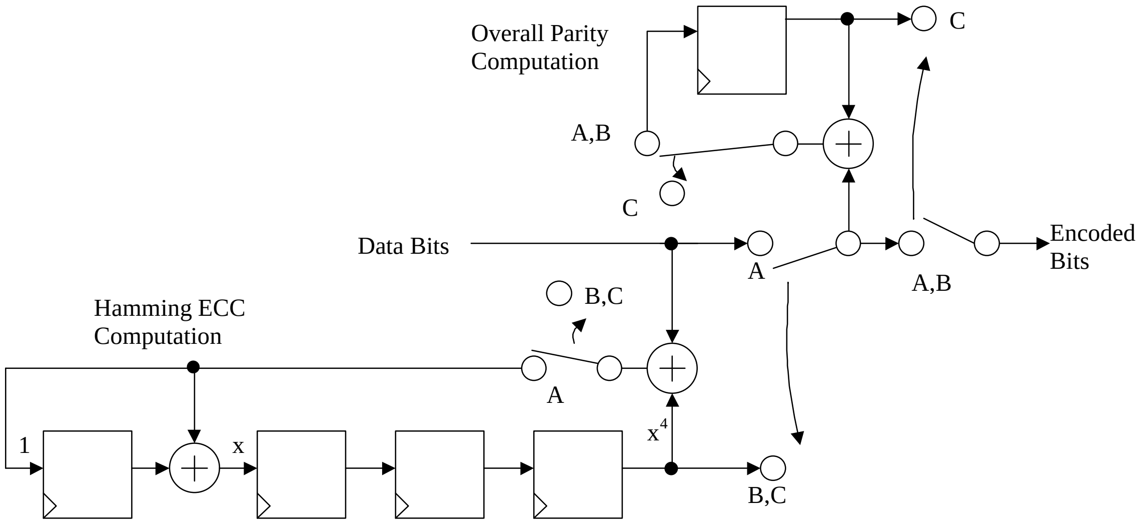


Figure 2 Encoder for (16,11) Extended Hamming Code

The circuit begins with all toggle switches in position A. Data to be encoded is input one bit per clock to both the Hamming error correction code (ECC) computation logic and the overall parity computation logic. Extended Hamming codes are systematic codes, so this data is also output on the encoded bit output. After all k bits are input, the toggle switches are moved to position B. At this point, data from the Hamming ECC logic is shifted out on the encoded bits bus. Finally, the overall parity bit is shifted out when the output select switch is moved to position C.

In order to encode the product code, each data bit is input both into a row LFSR and a column LFSR. Note that only one row LFSR is necessary for the entire block, since data is input in row order. However, each column of the array must be encoded with a separate LFSR. Each column LFSR is clocked for only one bit of the row, so a more efficient method of column encoding is to store the column LFSR states in a k_x x $(n_y\text{-}k_y)$ storage memory. A single LFSR can then be used for all columns of the array. With each bit input, the appropriate column LFSR state is read from the memory, clocked, and written back to the memory.

The encoding process will be demonstrated with an example. Assume a two-dimensional (8,4)x(8,4) extended Hamming Product code is to be encoded. This block has 16 data bits, and 64 total encoded bits. Fig. 3 shows the original 16 data bits denoted by D_{yx}.

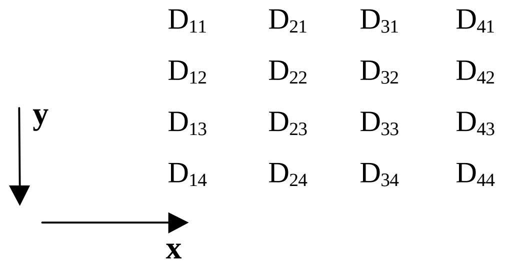


Fig. 3 Original Data for Encoding

The first four bits of the array are input to the row encoder in the order D_{11}, D_{21}, D_{31}, D_{41}. Each bit is also input to a unique column encoder. Again, a single column encoder may be used, with the state of each column stored in a memory. After the fourth bit is input, the first row encoder ECC bits are shifted out.

This process continues for all four rows of data. At this point, 32 bits have been output from the encoder, and the four column encoders are ready to shift out the column ECC bits. This data is shifted out at the end of the row. This continues from the remaining 3 rows of the array. Fig. 4 shows the final encoded block with the 48 generated ECC bits denoted by E_{yxz}.

$$\begin{array}{cccccccc}
D_{11} & D_{21} & D_{31} & D_{41} & E_{51} & E_{61} & E_{71} & E_{81} \\
D_{12} & D_{22} & D_{32} & D_{42} & E_{52} & E_{62} & E_{72} & E_{82} \\
D_{13} & D_{23} & D_{33} & D_{43} & E_{53} & E_{63} & E_{73} & E_{83} \\
D_{14} & D_{24} & D_{34} & D_{44} & E_{54} & E_{64} & E_{74} & E_{84} \\
E_{15} & E_{25} & E_{35} & E_{45} & E_{55} & E_{65} & E_{75} & E_{85} \\
E_{16} & E_{26} & E_{36} & E_{46} & E_{56} & E_{66} & E_{76} & E_{86} \\
E_{17} & E_{27} & E_{37} & E_{47} & E_{57} & E_{67} & E_{77} & E_{87} \\
E_{18} & E_{28} & E_{38} & E_{48} & E_{58} & E_{68} & E_{78} & E_{88}
\end{array}$$

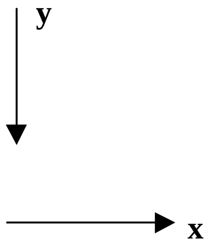


Fig. 4 Encoded Block

Transmission of the block over the channel occurs in a linear fashion, with all bits of the first row transmitted left to right, followed by the second row, etc. This allows for the construction of a near zero latency encoder, since the data bits can be sent immediately over the channel, with the ECC bits inserted as necessary. For the (8,4)x(8,4) example, the output order for the 64 encoded bits would be D_{11}, D_{21}, D_{31}, D_{41}, E_{51}, E_{61}, E_{71}, E_{81}, D_{12}, D_{22}, … E_{88}.

Notation:

the codes defined for the rows (x-axis) are binary (n_x,k_x) block codes

the codes defined for the columns (y-axis) are binary (n_y,k_y) block codes

the codes defined for the z-dimension (z-axis) are binary (n_z,k_z) block codes

data bits are noted $D_{y,x,z}$ and parity bits are noted $E_{y,x,z}$

Shortened BTCs

To match packet sizes, a product code is shortened by removing symbols from the array. In the two-dimensional case, either rows or columns can be removed until the appropriate size is reached. Unlike one-dimensional codes (such as Reed-Solomon codes), parity bits are removed as part of shortening process, helping to keep the code rate high.

There are two steps in the process of shortening of product codes. The first step is to remove an entire row or column from a 2-dimensional code, or an entire X, Y, or Z plane from a 3-dimensional code (such a code is defined as $((n_i,k_i)$-N), where N is a number of deleted rows or columns). This is equivalent to shortening the constituent codes that make up the product code. This method enables a coarse granularity on shortening, and at the same time maintaining the highest code rate possible by removing both data and parity symbols. Further shortening is obtained by removing individual bits from the first row of a 2-dimensional code, or from the top plane of a 3-dimensional code.

Example of a Shortened Two-Dimensional BTC

For example, assume a 456-bit block size is required, with code rate of approximately 0.6. The base code chosen before shortening is the (32,26)x(32,26) code which has a data size of 676 bits. Shortening all rows by 5 and all columns by 4 results in a (27,21)x(28,22) code, with a data size of 462 bits. To get the exact block size, the first row of the product is shortened by an additional 6 bits. The final code is a (750,456) code, with a code rate of 0.608. Fig. 5 shows the structure of the resultant block.

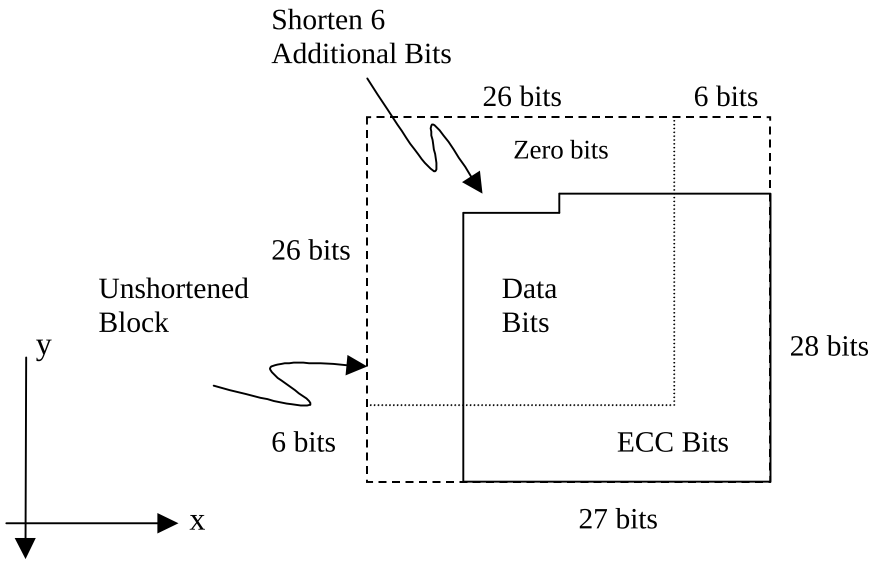


Fig. 5 Structure of Shortened 2 D Block

Modifications to the encoder to support shortening are minimal. Since shortened bits are always zero, and zeros input to the encoder LFSR result in a zero state, the shortened bits can simply be ignored for the purposes of encoding. The encoder simply needs to know how many bits per row to input to the row LFSR before shifting out the result. Similarly, it must know the number of columns to input to the column encoders. Beyond this, no modifications are necessary.

Three Dimensional TPC Encoding Example

For a three-dimensional TPC block, the element ordering for input/output for both encoding and decoding is as follows:

Rows;

Columns;

Z-axes.

For a three-dimensional data block of size (i,j,k) and total size (data +ecc bits) of (p,q,r) the bit order for input and output is:

$D_{111}, D_{211}, D_{311}, \ldots D_{j11}, \ldots E_{p11}, D_{121}, D_{221}, \ldots E_{p21}, \ldots E_{pq1}, D_{112}, D_{212}, \ldots E_{p12}, \ldots E_{pq2}, \ldots E_{pqr}$.

This is shown in Figure 6.

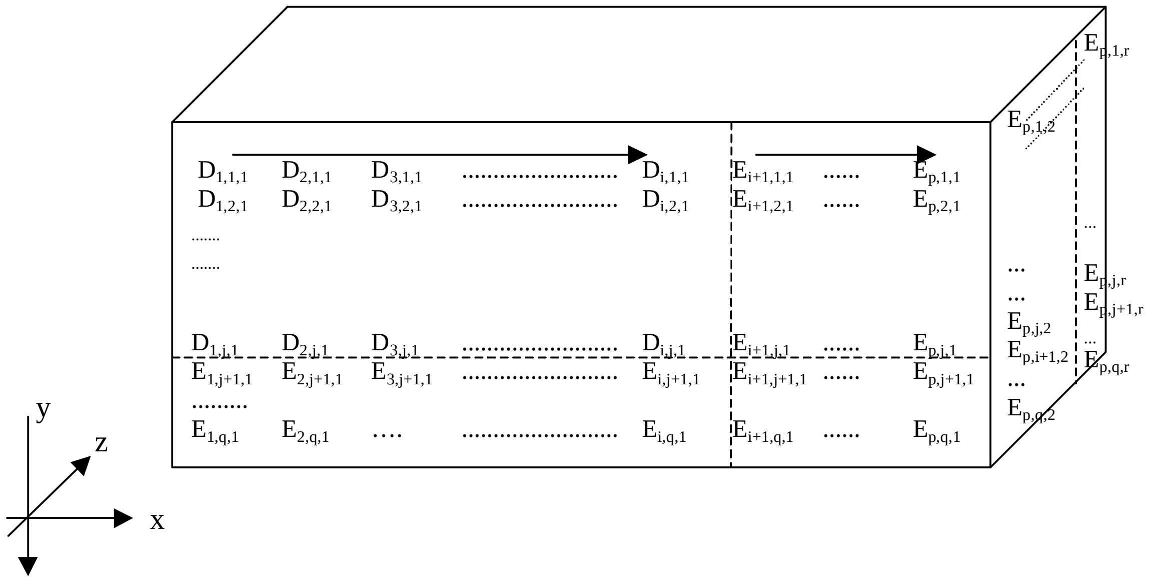

$D_{1,1,1}$ $D_{2,1,1}$ $D_{3,1,1}$ .... $D_{i,1,1}$ $E_{i+1,1,1}$ ...... $E_{p,1,1}$
$D_{1,2,1}$ $D_{2,2,1}$ $D_{3,2,1}$ .... $D_{i,2,1}$ $E_{i+1,2,1}$ ...... $E_{p,2,1}$
$D_{1,j,1}$ $D_{2,j,1}$ $D_{3,j,1}$ .... $D_{i,j,1}$ $E_{i+1,j,1}$ ...... $E_{p,j,1}$
$E_{1,j+1,1}$ $E_{2,j+1,1}$ $E_{3,j+1,1}$ .... $E_{i,j+1,1}$ $E_{i+1,j+1,1}$ ...... $E_{p,j+1,1}$
$E_{1,q,1}$ $E_{2,q,1}$ …. .... $E_{i,q,1}$ $E_{i+1,q,1}$ ...... $E_{p,q,1}$
$E_{p,1,r}$
$E_{p,1,2}$
$E_{p,j,r}$
$E_{p,j+1,r}$
$E_{p,j,2}$
$E_{p,i+1,2}$
$E_{p,q,r}$
$E_{p,q,2}$
y
z
x

Figure 6 Structure of 3-Dimensional TPC

Suppose a 0.4-0.45 rate code is required with a data block size of 1096 bits. The following shows one possible method to create this code.

Start with a (32,26)x(32,26)x(4,3) code. The optimum shortening for this code is to remove rows and columns, while leaving the already very short z axis alone. Therefore, since we desire a 1096 bit 3-D code, we can find the desired vector data size by taking the square root of 1096/3, and rounding up. This yields a row/column size of about 20. In fact, having a row size of 20, a column size of 19, and a z column size of 3 gives us the closest block size to 1096 bits.

The code size is now a (26,20)x(25,19)x(4,3) = (2600,1140). To get the exact data size, we further shorten the first plane of the code by 44 bits. This is accomplished by shortening 2 full rows from the first plane, with each row removing 20 bits from the data block, and shortening another 4 bits from the next row. This results in a (2544,1096) code, with rate = 0.43. The following diagram shows the original code, along with the physical location of the shortened bits.

The following diagram shows the original code, along with the physical location of the shortened bits.

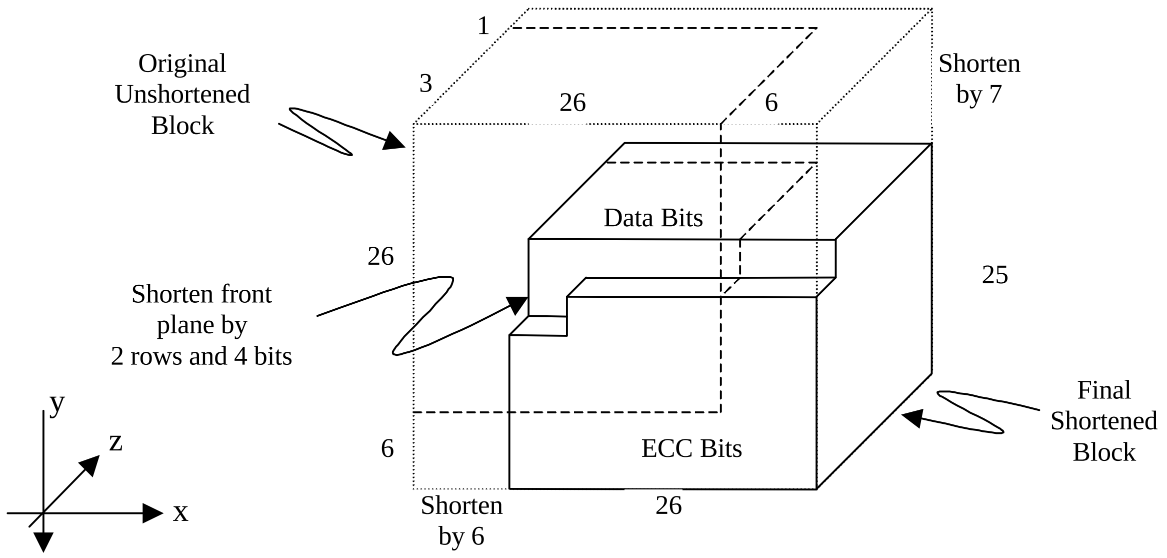


Figure 7 Structure of Shortened 3 D Block

Fine adjustment of code shortening with 3-D codes is accomplished by shortening a given number of full rows from the top plane of the array.

Iterative Decoding

Each codeblock in a product code is decoded independently. First, all the horizontal blocks are decoded then all the vertical received blocks are decoded (or vice versa). The decoding procedure is generally iterated several times to maximize the decoder performance. To achieve optimal performance, the block by block decoding must be done utilizing soft information. This soft decision decoder must also output a soft decision metric corresponding to the likelihood that the decoder output bit is correct. This is required so that the next decoding will have soft input information as well. In this way, each decoding iteration builds on the previous decoding performance.

The core of the decoding process is the soft in soft out (SISO) constituent code decoder. High performance iterative decoding requires the constituent code decoders to not only determine a transmitted sequence, but to also yield a soft decision metric which is a measure of the likelihood or confidence of each bit in that sequence. Since most algebraic block decoders don't operate with soft inputs or generate soft outputs, such block decoders have been primarily realised using the Soft-Output Viterbi Algorithm (SOVA) [6] or a soft-output variant of the modified Chase algorithm(s). However, this does not limit the choice of decoding algorithms as other SISO block decoding algorithms can be used.

The following describes the Modified Chase Algorithm method of decoding block codes. We may assume that we start by decoding the matrix rows. Soft input is calculated by squared Euclidean distance calculation on the I and Q components of the QPSK signal. A certain radius p around each received word is chosen as a parameter. The p least reliable positions in the received word are determined, and by a set of test-sequences formed by the p least reliable positions, the codewords lying closest to the received word are determined by hard decision decoding. A weighted reliability on each of the bits in the received word is calculated by simplified derivatives of the LLR algorithm. This weighted reliability on each decision is used as the soft input for the decoding of the columns. The same algorithms are used for the columns, and we may return to the rows for a second decoding, etc.

In addition to the SISO decoding of the constituent codes, information must be shared between decoding iteratives. It turns out that the soft input to a given iteration, which is the array of soft information (Soft_Input$_x$) should be the sum of the soft output (array) from the previous iteration (Soft_Output$_{x-1}$) and soft input to the first iteration (Original_Soft_Channel_Data). This can be represented as:

Soft_Input$_x$ = Soft_Output$_{x-1}$ – Soft_Input$_{x-1}$ + Original_Soft_Channel_Data

For an n - dimensional TPC,

Soft_Input$_x$ = Soft_Output$_{x-1}$ – Soft_Input$_{x-1}$

+ Soft_Output$_{x-2}$ – Soft_Input$_{x-2}$

+ Soft_Output$_{x-2}$ – Soft_Input$_{x-2}$

+

•

•

•

+ Soft_Output$_{x-(n-1)}$ – Soft_Input$_{x-(n-1)}$ + Original_Soft_Channel_Data

A Block Diagram of the iterative decoding process is given in figure B1 and figure B2 for two and three dimensional codes, respectively. Previous Soft_Input minus Soft_Output data is stored in the difference arrays. The weighting factor is used to enhance performance. The value used depends on the constituent block codes used along with number of iterations.

Two Dimensional TPC Decoding per Axis Iteration

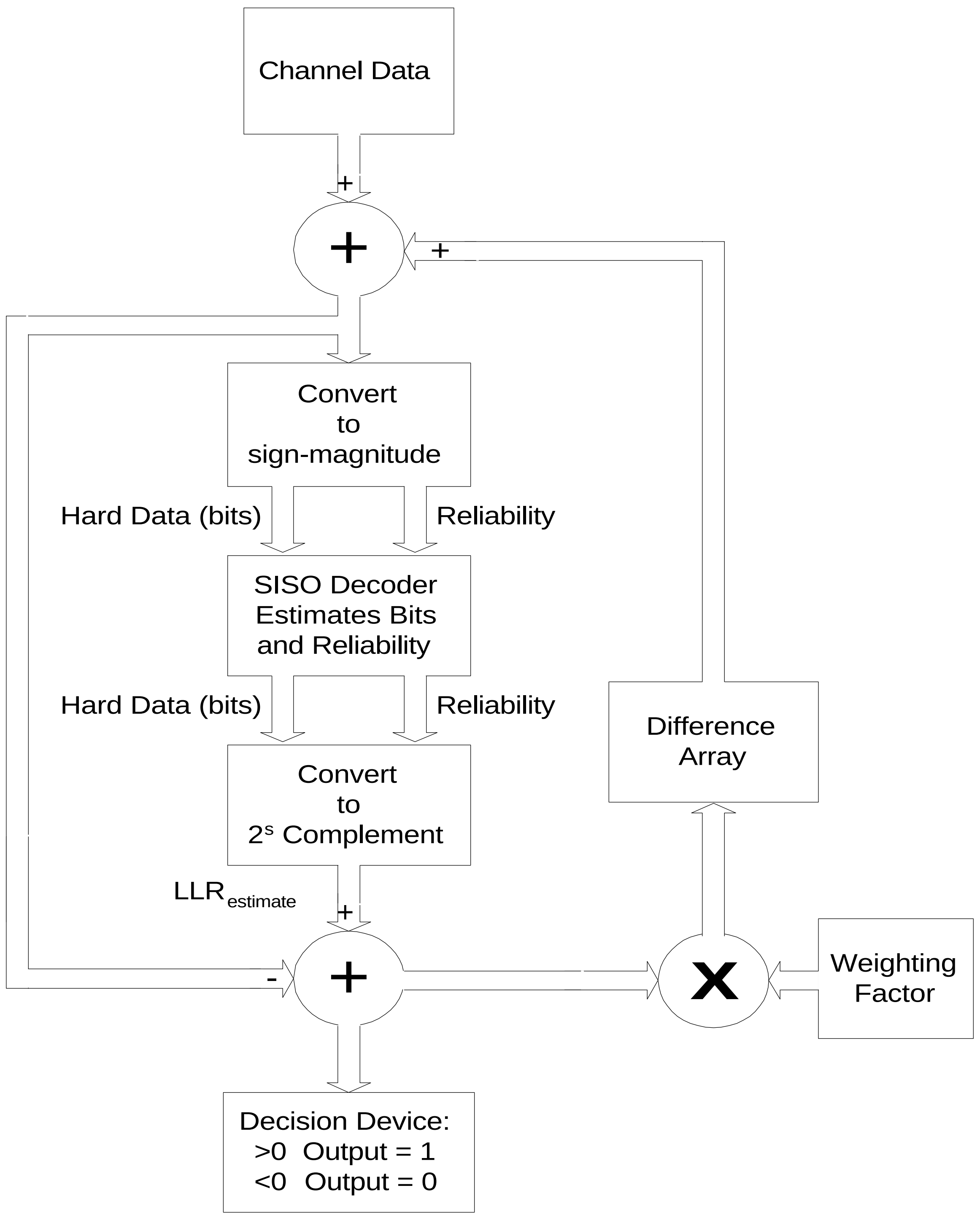


Figure 8 Two Dimensional TPC Decoder Block Diagram.

Three Dimensional TPC Decoding per Axis Iteration

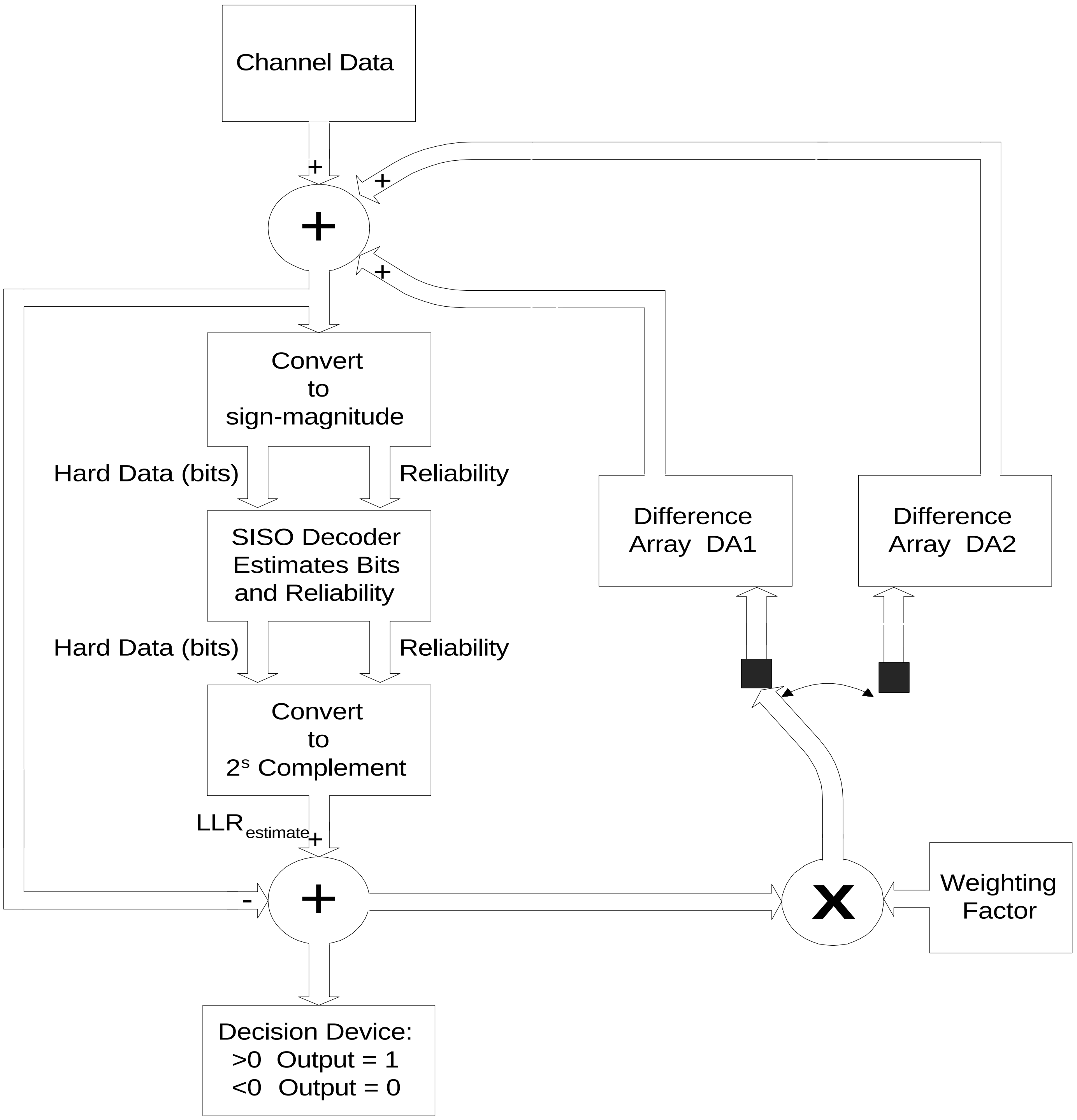


Figure 9 Three Dimensional TPC Decoder Block Diagram.

Distance Structure and Bounds

In order to determine an asymptotic coding bound that is valid for low bit error rates, the distance structure of the code must be known. The entire distance structure of a large product code is not easily found, but the minimum Hamming weight codewords can be computed by exploiting the product nature of the code. When a product code is constructed from linear block codes, the overall product code is also linear. Therefore, finding the minimum Hamming weight codewords is equivalent to finding the minimum distance between neighboring codewords of the set.

It is well known that Hamming codes have a minimum Hamming weight of three. Extending these codes with the addition of an overall parity bit increases this weight to four. The minimum weight of a product code is found by multiplying the weights of the constituent codes. Fig. 10 shows an example minimum weight codeword of the (8,4)x(8,4) product code. Note that the overall minimum weight codeword contains four rows of ones, with each row containing four ones, creating a codeword of weight 16.

```
1  1  0  0  0  1  0  1
0  0  0  0  0  0  0  0
1  1  0  0  0  1  0  1
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
1  1  0  0  0  1  0  1
1  1  0  0  0  1  0  1
0  0  0  0  0  0  0  0
```

Fig. 10 Minimum Weight Codeword

In order to find the number of codewords of this minimum weight, we again look at the weight structure of the extended Hamming code. The number of codewords of weight four in the extended Hamming code is easily found with a computer search. The number of codewords of weight 16 in the product code is equal to the product of the number of codewords of weight 4 in the constituent code. For example, the (32,26)x(32,26) product code has 1240^2 codewords of weight 16.

Under the assumption of an AWGN channel, the probability of codeword (packet) error, P_E for a two-dimensional product code is lower bounded by

$$P_E \approx a_x a_y \cdot Q\left(\sqrt{\frac{d^x_{\min} d^y_{\min} \cdot 2 \cdot R \cdot E_b}{N_0}}\right) \qquad (1)$$

where a and d_{min} are the number of codewords and minimum distance of the constituent codes, R is the overall product code rate, and E_b/N_0 is the signal to noise ratio expressed in terms of energy per data bit.

For each packet that fails, the number of errors in the packet is greater than or equal to the minimum distance of the product code. Therefore, a bound on the probability of bit error, P_B can be found from equation (1) as

$$P_B \approx \frac{d_{\min}^{x} \cdot d_{\min}^{y}}{N} \cdot P_E \qquad (2)$$

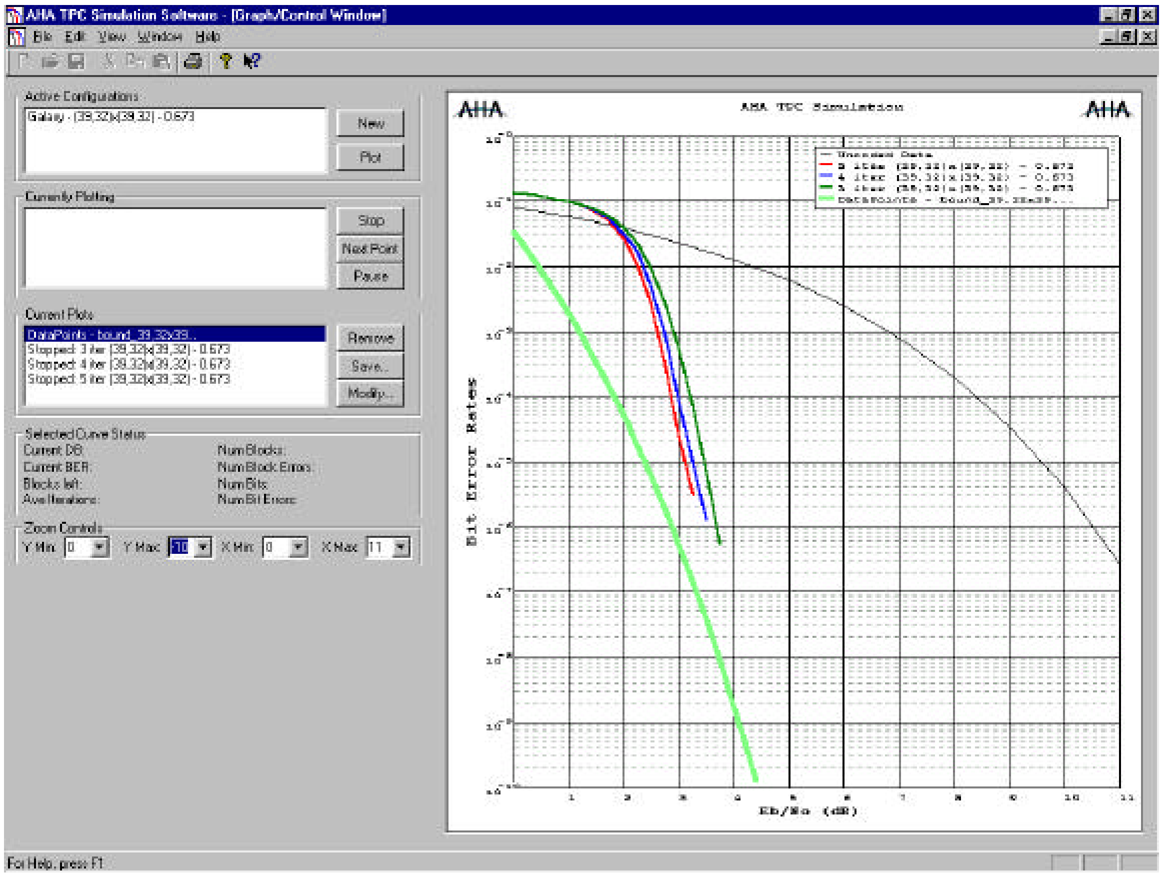
where N is the size of the overall product code. This bound becomes tight as the SNR increases. Note that the data and ECC bits have the same probability of bit error, so the computation of P_B is equivalent to computation over just the data. Further work on coding bounds for product codes was done by Tolhuizen et al.

These bounds are easily extended to three-dimensional and shortened product codes with either extended Hamming or parity only constituent codes. These bounds, along with simulation results, are available if needed for the FEC selection process.

Decoder Complexity vs. Decoder Iterations

As mentioned in the introductory summary, decoder complexity is a function of several variables including the maximum size of the codes, semiconductor process technology, the internal math precision of the decoder engine, the specific decoder algorithm used, and the number of decoder iterations performed.

A reduced iteration decoder provides a means of reducing decoder complexity without any changes to the physical layer and associated air interface. A reduction in decoder iterations from five to three for example would enable the construction of a lower cost decoder, if needed, for a complexity sensitive receive terminal. In this case, the performance is degraded by approximately 0.3 dB at a BER of 10^{-6} but with a decoder complexity reduction of up to 30%. As the operating point approaches the asymptotic bound, the difference between the three iteration plot and the five iteration plot will asymptotically approach zero dB. The choice of decoder complexity versus performance can be left up to the individual vendors building systems to this standard. The following graph illustrates the decoder iteration count vs. BER performance for one of the proposed codes. The light green line to the left represents the asymptotic bound for this particular code.


AHA TPC Simulation Software - [Graph/Control Window]
File Edit View Window Help
New
Plot
Stop
Next Point
Pause
Remove
Save...
Modify...
AHA
AHA TPC Simulation
AHA
Uncoded Data
3 iter (39,32)x(39,32) - 0.673
4 iter (39,32)x(39,32) - 0.673
5 iter (39,32)x(39,32) - 0.673
DataPoints - bound_39,32x39...
Bit Error Rates
Eb/No (dB)
For Help, press F1

Active Configurations
Galaxy - (39,32)x(39,32) - 0.673

Currently Plotting

Current Plots
DataPoints - bound_39,32x39...
Stopped: 3 iter (39,32)x(39,32) - 0.673
Stopped: 4 iter (39,32)x(39,32) - 0.673
Stopped: 5 iter (39,32)x(39,32) - 0.673

Selected Curve Status
Current DB: NumBlocks:
Current BER: NumBlock Errors:
Blocks left: NumBits:
Ave Iterations: NumBit Errors:

Zoom Controls
Y Min: 0 Y Max: -10 X Min: 0 X Max: 11

Figure 11 Effect of Decoder Iterations on Coding Gain

Block Error Detection

This proposal recommends the inclusion of a CRC based block error detection scheme as a part of the standard. A 32-bit linear feedback shift register with programmable taps can implement any desired CRC polynomial up to 32 bits in length. For situations where a strong block error detection mechanism is not required, inclusion of a CRC word can be turned off.

In the IEEE 802.16.1pc-00/31 submittal, it was suggested that inclusion of a CRC block error detection mechanism was not needed for Reed-Solomon based solutions due to the strong inherent detection capability of these codes. Note that an unshortened Reed-Solomon decoder (t=8) has a block error detection capability equal to a CRC size of 12 bits, but only if the erasure detection capability of the Reed-Solomon code is not used. Some of the shortened codes proposed in IEEE 802.16.1pc-00/31 have in fact very poor block error detection capability based on their low t value. For example, the 53 byte, rate 0.8265 code has a t = 2 error correction capability which equates to a very low probability of a correct block error detection.

The relationship between the t of a Reed-Solomon code and its block error detection probability is as follows:

$Pd = 1 - 1 / t!$ where ! denotes a factorial

Inclusion of a CRC based block error detection scheme in this standard is both desired and highly practical from a complexity and performance standpoint. The following table lists some commonly used CRC's and their associated block error detection capability.

Table 2 CRC Polynomials

CRC	Size (bits)	Polynomial (hex)	Detection Capability
4	4	1f	0.9375
8	8	1d5	0.99609
12	12	180f	0.999756
ANSI	16	18005	0.999985
CCITT	16	11021	0.999985
SDLC	16	1a097	0.999985
24	24	1805101	0.9999999404
32A	32	1404098e2	0.99999999953
32B	32	104c11db7	0.99999999977

Notes: The code rate loss due to the addition of an outer CRC code to a TPC is minimal. The Eb/No loss due to the addition of the CRC code can be computed with the following equation:

$Eb/No \text{ loss (dB)} = 10*\log10(k/(k+c))$

where k is the data block size, and c is the CRC size. For example, the addition of a 16 bit CRC to a 188 byte (1504 bit) code results in a 0.046 dB loss. This CRC has a detection capability much higher than a t=8 Reed Solomon code.

The detection capability is the probability that an incorrect block is not marked in error. The probability of an undetected block is computed by multiplying the block error rate by (1 - Detection Capability).

Recommended Codes

This section has added three codes for consideration to those submitted in IEEE 802.16.1pc-00/32/rl. It is not intended to replace those codes but is offered for consideration in the Mode A downlink continuous transmission case.

An encoder and/or decoder based on these codes will also support all of the codes contained in the IEEE 802.16.1pc-00/32/rl submittal.

The key aspects of these codes are:

1. Provides higher BER performance
2. Provides higher code rates for increased data throughput
3. A single decoder and/or encoder design will support all of this codes plus those proposed

   in IEEE 802.16.1pc-00/32/rl with no additional complexity.

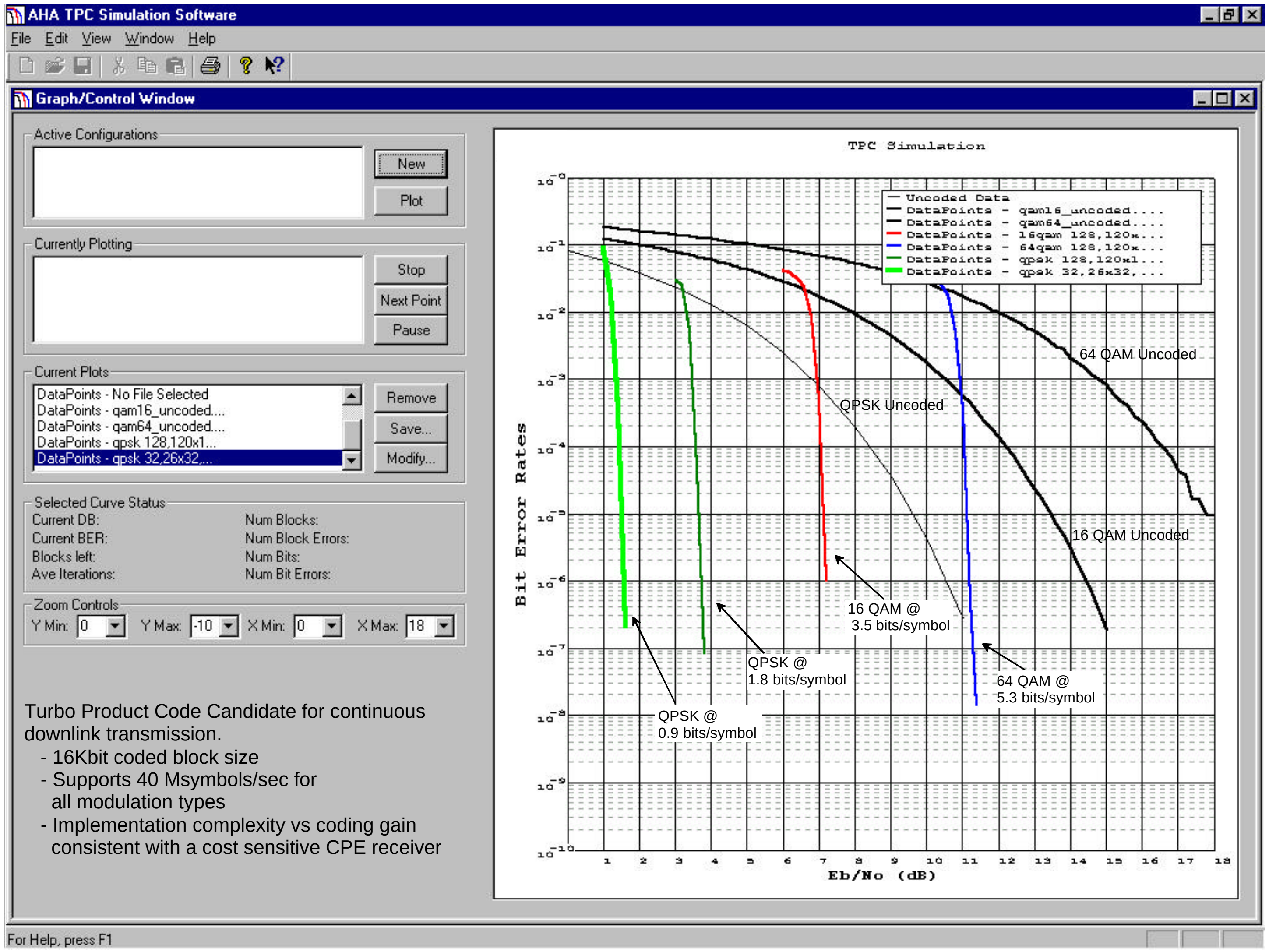
These codes are based on a maximum encoded (data +FEC) block size of 16Kbits. The following details the codes and their performance attributes.

Table 3 Large Block Codes

Code	**32,26 x 32,26 x 16,11 (sb 4)**	**128,120 x 128,120 (sb 0)**	**128,127 x 128,127 (sb 1)**
Aggregate Code Rate	0.454	0.880	0.980
Uplink/Downlink/Both	Downlink	Downlink	Downlink
Eb/No Required @ 10^{-6}	1.5 dB	3.8 dB	7.5 dB
Eb/No Required @ 10^{-9}	1.8 dB	4.0 dB	8.5 dB
Encoder Complexity QPSK/16QAM/64QAM	15 Kgates	15 Kgates	15 Kgates
Decoder Complexity QPSK/16QAM/64QAM	< 250 Kgates @ 5 iter. and 240 Mbits/sec	< 250 Kgates @ 5 iter. and 240 Mbits/sec	< 250 Kgates @ 5 iter. and 240 Mbits/sec
Block size (payload bits)	7,432	14,400	16,128
Latency	32 Kbits	32 Kbits	32 Kbits

Notes: The complexity estimates are for a single encoder or decoder that supports all three large block codes shown here plus the small block codes of IEEE 802.16.1pc-00/32/rl. A small block only implementation will require substantially fewer gates. Estimates take into account all memory requirements.

The following figure plots the BER performance for the rate 0.88 code for all three modulation types and for the rate 0.45 code for QPSK modulation. This plot provides a good overview the breadth of payload throughput vs Eb/No that can be achieved with just two of the codes. For high Eb/No scenarios, the rate 0.98 code can provide even higher throughput thus providing increased value to service providers whose basic measure is bits/sec delivered to the end user.



— Uncoded Data
— DataPoints - qam16_uncoded....
— DataPoints - qam64_uncoded....
🟥 DataPoints - 16qam 128,120x...
🟦 DataPoints - 64qam 128,120x...
🟩 DataPoints - qpsk 128,120x1...
🟩 DataPoints - qpsk 32,26x32,...

DataPoints - No File Selected
DataPoints - qam16_uncoded....
DataPoints - qam64_uncoded....
DataPoints - qpsk 128,120x1...
DataPoints - qpsk 32,26x32,...

Current DB:
Current BER:
Blocks left:
Ave Iterations:
Num Blocks:
Num Block Errors:
Num Bits:
Num Bit Errors:

Y Min: 0 Y Max: -10 X Min: 0 X Max: 18

Turbo Product Code Candidate for continuous downlink transmission.
- 16Kbit coded block size
- Supports 40 Msymbols/sec for all modulation types
- Implementation complexity vs coding gain consistent with a cost sensitive CPE receiver

Figure 12   Large Block Code Performance

Small Block Codes

For the convenience of the reader, the codes proposed in the IEEE 802.16.1pc-00/32/rl submission are repeated here. The performance figures have been updated to reflect additional simulation results that utilized better feedback coefficients for the iterative decoding process. All results are based on five iterations and 4 soft metric bits into the decoder. For reference the results listed as “S/W sim 1” reflect the same submitters results shown here. Some of these codes have been validated with HW to as low as 10^{-10} BER. These HW results are available upon request. Asymptotic bound curves are also available upon request.

Table 4 Small Block Codes

Code	**39,32 x 39,32 sb 0**	**46,39 x 46,39 sb 17**	**63,56 x 63,56 sb 0**
Aggregate Code Rate	0.673	0.717	0.790
Uplink/Downlink/Both	Downlink	Downlink	Downlink
Eb/No Required @ 10^{-6} QPSK/16QAM/64QAM	3.5/6.5/10.7 dB	3.6/6.6/10.5 dB	3.5/6.9/11.0 dB
Eb/No Required @ 10^{-9} QPSK/16QAM/64QAM	4.3/7.5/11.7 dB	4.3/7.8/11.5 dB	4.3/7.5/12.0 dB
Encoder Complexity QPSK/16QAM/64QAM	10 Kgates	10 Kgates	10 Kgates
Decoder Complexity QPSK/16QAM/64QAM	< 150 Kgates @ 5 iter and 240 Mbits/sec	< 150 Kgates @ 5 iter and 240 Mbits/sec	< 150 Kgates @ 5 iter and 240 Mbits/sec
Block size (payload bits)	1024 (128 bytes)	1504 (188 bytes)	3136 (392 bytes)
Latency	3042 bits	4232 bits	7938 bits

Note: Gate counts are for a single encoder or decoder that supports all of the small block codes shown on this page and the next. Estimates take into account all memory requirements.

Table 4 Small Block Codes (continued)

Code	**11,10 x 5,4 sb 0**	**11,10 x 8,4 sb 0**	**29,23 x 6,5 sb 3**
Aggregate Code Rate	0.727	0.455	0.655
Uplink/Downlink/Both	Uplink	Uplink	Uplink
Eb/No Required @ 10^{-6} QPSK/16QAM/64QAM	7.2/11.2/16.8 dB	6.5/9.0/13.0 dB	5.4/8.8/12.8 dB
Eb/No Required @ 10^{-9} QPSK/16QAM/64QAM	8.9/15.5/22 dB	8.7/11.8/17.0 dB	7.3/10.8/15.5 dB
Encoder Complexity QPSK/16QAM/64QAM	10 Kgates	10 Kgates	10 Kgates
Decoder Complexity QPSK/16QAM/64QAM	< 150 Kgates @ 5 iter and 240 Mbits/sec	< 150 Kgates @ 5 iter and 240 Mbits/sec	< 150 Kgates @ 5 iter and 240 Mbits/sec
Block size (payload bits)	40 (5 bytes)	40 (5 bytes)	112 (14 bytes)
Latency	110 bits	176 bits	348 bits

Code	**16,11 x 16,11 sb 9**	**30,24 x 25,19**
Aggregate Code Rate	0.453	0.608
Uplink/Downlink/Both	Uplink	Uplink
Eb/No Required @ 10^{-6} QPSK/16QAM/64QAM	4.0/6.8/9.8 dB	3.4/6.3/10.0 dB
Eb/No Required @ 10^{-9} QPSK/16QAM/64QAM	5.8/8.8/11.8 dB	4.7/7.5/11.5 dB
Encoder Complexity QPSK/16QAM/64QAM	10 Kgates	10 Kgates
Decoder Complexity QPSK/16QAM/64QAM	< 150 Kgates @ 5 iter and 240 Mbits/sec	< 150 Kgates @ 5 iter and 240 Mbits/sec
Block size (payload bits)	112 bits (14 bytes)	456 bits (57 bytes)
Latency	512 bits	1500 bits

System Level Performance Validation

Block Turbo Code (Turbo Product Code) performance has been validated by simulation, theoretical analysis and by hardware implementations.

The following is offered as additional evidence:

1. Comtech Communications Corp. introduced a satellite modem in 1999 incorporating the turbo code technology being advocated in this proposal. The following is an excerpt from a statement made by Richard Miller, Vice President of Modem Engineering for Comtech:

"During the initial design process, AHA provided simulation software which enabled us to evaluate various code rates, block sizes, and the effects of code shortening. This permitted us to select an optimum approach for our intended application, and gave us an extremely accurate prediction of bit error rate (BER) versus Eb/No. When the AHA4501 hardware was incorporated into the new design, the results we obtained were within a few tenths of a dB from the predicted performance. We believe that this is almost entirely accounted for by the implementation loss of the demodulator, and the uncertainty in measurement of Eb/No.

In conclusion, the AHA simulation software appears to very closely match the real-world performance of their silicon, which in turn is truly impressive. "

Modeling Support

As noted above, simulation software is available that can simulate the performance of all codes proposed and in addition can support all of the modulation formats proposed. An expanded version of this simulation software is also available to tie into system level simulations that incorporate channel models, phase jitter, and other system level impairments. This software supports both C/C++ and Matlab API's and is currently available for Windows 95, 98 and NT platforms.

References

[1] P. Elias, "Error-Free Coding, " IRE Trans. Inf. Theory, PGIT-4, pp.29-37, September 1954

[2] Error Control Coding: Fundamentals and Applications, S. Lin and D. Costello, Prentice-Hall, 1983, pp. 274-277.

[3] Press Release, "AHA announces Turbo Product Code Forward Error Correction Technology", Nov. 2, 1998

[4] Press Release, "AHA announces Turbo Product Code Core Generator", September 20 1999

[5] Hewitt, E, "Turbo Product Codes for LMDS," IEEE Radio and Wireless Conference, August 1998

[6] A Viterbi Algorithm with Soft-Decision Outputs and its Applications, J. Hagenauer, P. Hocher, *IEEE Globecom '89,* Nov. 1989, pp. 1680-1685.

Related reference materials

Error Correcting Codes, W. Peterson, E. Weldon, *The MIT Press*, Cambridge Mass., 1972.

Separable map "filters" for the decoding of product and concatenated codes, J. Lodge, R. Young, P. Hoeher, J. Hagenauer, in *Proc. ICC '93*, pp. 1740 - 1745, May 1993.

Product Specification AHA4501 : 36 Mibts/sec Turbo Product Code Encoder / Decoder, available at http://www.aha.com.

Error-Control Tech. for Digital Comm., A.Michelson, A Levesque, *John Wiley & Sons, Inc*. 1985, pp. 45.

A Recursive Approach to Low Complexity Codes, R. Tanner, IEEE Trans. Inf. Theory, Sept. 1981, vol. IT-27, No. 5

R. Pyndiah, "Near-Optimum Decoding of Product Codes: Block Turbo Codes" IEEE Trans. Comm., vol. 46, pp. 1003-1010

White paper, "Helical Interleaving for Burst Error Correction with Turbo Product Codes", Efficient Channel Coding, Inc., June 1998