# IEEE 802.16 Compatible Turbo Product Code Decoder v1.1

## Features

- Performs decoding for the turbo product codes listed in the IEEE 802.16 and 802.16a standards

- Optimized for Virtex®-II and Virtex-II Pro FPGAs using structural VHDL and relationally placed macro (RPM) technology for maximum and predictable performance. Spartan-3 and Virtex-4 device families are supported with a non-RPM version.

- Based on the Comtech AHA Corporation (AHA) TPC Galaxy Core

- Supports block sizes from 64 bits to 4 Kbits, 64 possible product codes

- Four-SISO option achieves a data rate of 155 Mbps with five iterations using $(64,57)^2$ code (in Virtex-II FPGAs)

- Single-SISO option achieves a data rate of 45 Mbps with five iterations using $(64,57)^2$ code (in Virtex-II FPGAs)

- Provides more than 7 dB of coding gain at $10^{-6}$ BER compared to uncoded BPSK in an AWGN channel

- Provides up to 10 dB of coding gain over Reed-Solomon concatenated with Viterbi in an MMDS fading channel at a BER of $10^{-6}$

- Separate input and output enables plus internal buffering allow flexible handling of data flow

- Four-SISO option requires 3313 slices, 17 block RAMs, and 16 hardware multipliers (in Virtex-II/Virtex-II Pro FPGAs)

- Single-SISO option requires 1062 slices, 13 block RAMs, and 4 hardware multipliers (in Virtex-II/Virtex-II Pro FPGAs)

- Maximum clock rate of 150 MHz in Virtex-II (-6), 170 MHz in Virtex-II Pro (-7), 111 MHz in Spartan-3 (-5), and 172 MHz in Virtex-4 (-10) FPGAs

## Applications

The TPC decoder core operates in local multi-point distribution service (LMDS) and multi-channel, multi-point distribution service (MMDS) communication systems. The core is especially useful for those communication links that require high data rates, low latency, high code rates, high spectral efficiency, and a high degree of forward error correction capability. Turbo product codes offer a higher performance alternative to Reed-Solomon or Reed-Solomon concatenated with Viterbi error correction methods.

| LogiCORE™ Facts | | | | |
|---|---|---|---|---|
| **Core Specifics** | | | | |
| Supported Device Family | Virtex-II, Virtex-II Pro, Virtex-4, and Spartan-3 | | | |
| **Resources Used (Virtex-II)** | | | | |
| **Core Option** | **I/O** | **LUTs** | **FFs** | **Block RAMs** |
| Four-SISO | 75 | 6350 | 5572 | 17 |
| Single-SISO | 75 | 1943 | 1780 | 13 |
| Special Features | | | | RPM |
| **Provided with Core** | | | | |
| Documentation | Product Specification | | | |
| Design File Formats | Encrypted EDIF | | | |
| Constraints File | .ucf (user constraints file) | | | |
| Verification | VHDL Test Bench | | | |
| Instantiation Template | VHDL Wrapper | | | |
| **Design Tool Requirements** | | | | |
| Xilinx Implementation Tools | ISE 4.2.03i or later | | | |
| Verification | Mentor Graphics® ModelSim® PE 5.4e | | | |
| Simulation | Mentor Graphics ModelSim PE 5.4e | | | |
| Synthesis | None | | | |
| **Support** | | | | |
| Provided by Xilinx, Inc. | | | | |

## Pinout

Figure 1 shows the top level interface of the TPC Decoder core. The ports are defined in Table 1.



*Figure 1:* **Core Schematic Symbol**

*Table  1:* **Core Signal Pinout**

| Signal | Direction | Description |
|---|---|---|
| data_in[15:0] | Input | Four samples of 4-bit quantized data input to decoder in parallel<br>data_in[3:0] = first sample in time<br>data_in[7:4] = second sample in time<br>data_in[11:8] = third sample in time<br>data_in[15:12] = fourth sample in time. |
| data_en | Input | Active High enable. On rising edge of sysclk, if data_en is High, then data_in[15:0] is written to decoder internal buffer. |
| start_in | Input | Active High start signal is used to indicate the start of an input code block. Only valid when data_en is High. Start_in must be coincident with the data_en pulse corresponding to the start of a new code block. |
| output_en | Input | Active High enable requesting that the decoder output data. Output_en should be High for one sysclk per output data transfer and should be asserted only when OutputRDY is High. Asserting output_en when OutputRDY is Low will result in an underflow. |
| xcode[4:0] | Input | Specifies the code type in the x dimension (see Table 3). |
| ycode[4:0] | Input | Specifies the code type in the y dimension (see Table 3). |
| xfeedback[4:0] | Input | Provides scaling of SISO output after each x-axis iteration. A value of 16 is typical ([Ref 1]) |
| yfeedback[4:0] | Input | Provides scaling of SISO output after each y-axis iteration. A value of 16 is typical ([Ref 1]). |
| NumIter[7:0] | Input | Number of full iterations. |
| sysclk | Input | Clock used for majority of design (rising edge). |
| clk2x | Input | 2X clock that is rising edge aligned with sysclk. |
| reset | Input | Active High synchronous reset. |

*Table 1:* **Core Signal Pinout** *(Cont'd)*

| Signal | Direction | Description |
|---|---|---|
| data_out[15:0] | Output | Four samples of 4-bit quantized data out of decoder<br>data_out[3:0] = first sample in time<br>data_out[7:4] = second sample in time<br>data_out[11:8] = third sample in time<br>data_out[15:12] = fourth sample in time. |
| data_valid | Output | High when data_out[15:0] is valid. Data_valid is a delayed version of output_en. The delay from output_en to data_valid is 4 sysclks. |
| start_out | Output | Active High start signal is used to indicate the start of an output code block. Start_out is High whenever data_out[15:0] corresponds to the first four samples of an output code block. |
| hdata_in[3:0] | Output | Hard input data is delayed to match corresponding decoded data out. |
| OutputRDY | Output | Active High indicates that the decoder is ready to output data. OutputRDY remains High as long as the internal output FIFO has data available to send out of the core. OutputRDY must be High before asserting output_en to avoid an underflow. |
| overflow | Output | Active High fault indicating an overflow in internal FIFO. |
| underflow | Output | Active High fault indicating an underflow in internal FIFO. |

# Functional Description

A product code works on an array of information bits by encoding all of the rows followed by encoding all of the columns. The column encoding is performed on both the original information bits as well as the parity bits generated from the row encoding. The constituent codes supported by the TPC decoder core are either extended Hamming codes or simple parity codes.

The encoding process is illustrated in Figure 2 for an (8,4)-by-(8,4) product code, where the $D_{ij}$ represent input data, the $H_{ij}$ represent parity bits from the Hamming code and the $P_{ij}$ represent the overall parity bits. In general, a $k_x \times k_y$ information array is transformed into an array of dimension $n_x \times n_y$. The input data array is filled row-by-row from left to right and the code block is read row-by-row from left to right.

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

ds212_02_050208

*Figure 2:* **Product Code Block for the (8,4)-by-(8,4) Case**

Table 2 lists the Hamming code generator polynomials supported by the core. The extended Hamming code is defined by appending an overall (even) parity bit to the Hamming code vector. The parity only code is a single parity bit computed over an information bit vector.

*Table 2:* **Hamming Code Generator Polynomials**

| n | k | Generator Polynomial |
|---|---|---|
| 7 | 4 | $X^3 + X + 1$ |
| 15 | 11 | $X^4 + X + 1$ |
| 31 | 26 | $X^5 + X^2 + 1$ |
| 63 | 57 | $X^6 + X + 1$ |

The TPC decoder core supports eight possible code types for each of the constituent codes, for a total of 64 possible product codes. The xcode[4:0] and ycode[4:0] inputs to the core are used to specify the code type as shown in Table 3. The code in the x dimension (i. e, the code for each row) is specified with the control bits xcode[4:0]. The code in the y dimension (i. e., the code for each column) is specified with the control bits ycode[4:0]. For example, setting xcode[4:0] = "00101" and ycode[4:0] = "00110" configures the decoder for the (32,26) × (64,57) code.

*Table 3:* **Constituent Codes Supported by TPC Decoder**

| X/YCode[4:0] | X/Y Code | Code Type |
|---|---|---|
| 00011 | (8,4) | Extended Hamming |
| 00100 | (16,11) | Extended Hamming |
| 00101 | (32,26) | Extended Hamming |
| 00110 | (64,57) | Extended Hamming |
| 10011 | (8,7) | Parity Only |
| 10100 | (16,15) | Parity Only |
| 10101 | (32,31) | Parity Only |
| 10110 | (64,63) | Parity Only |

The TPC decoder performs iterative decoding of the input code block with the aid of one or four soft-in/soft-out (SISO) decoders. As the 4-bit soft data arrives at the TPC decoder, it is stored in an internal input array. Once an entire code block is stored, the SISO decoders begin processing the data by reading the data from the input array, performing SISO decoding and storing the SISO outputs in an intermediate storage array. The SISO output from the final iteration is combined with the input array and stored in the output array. While the first code block is being processed, a second code block is being stored in a second page of the input array. The SISO decoding of the first code block must be complete before storing a third code block because it is written to the same page of memory as the first code block. This double buffer approach continues for each successive code block that is processed. The top-level block diagram of the TPC decoder is shown in Figure 3.
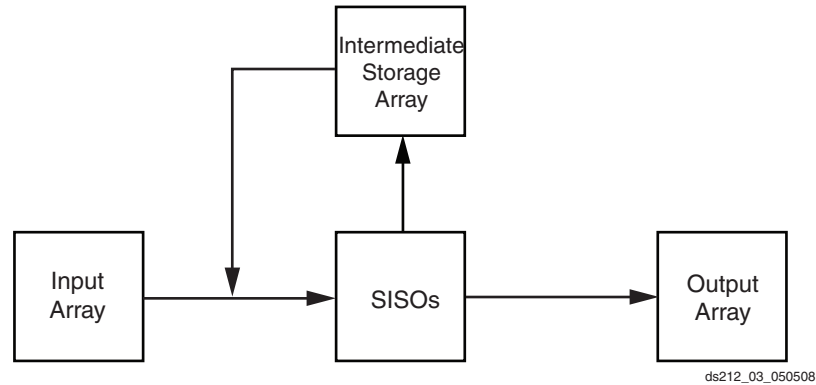
*Figure 3:* **Top-Level Block Diagram of TPC Decoder**

SISO decoding is performed on the code block one axis at a time. The SISO output from decoding along the x dimension is scaled by the xfeedback[4:0] signal, combined with the original input array and then fed back into the SISO for decoding along the y dimension. The SISO output from the y dimension decoding is then scaled by the yfeedback[4:0] signal, combined with the original input array and then fed back for another iteration along the x dimension, and so on.

The completion of one decoding pass along one dimension (x or y) of the array is referred to as an axis iteration. The completion of decoding along the x dimension followed by decoding along the y dimension is called a full iteration. The number of full iterations performed is controlled by the input signal NumIter[7:0]. See [Ref 1] for more information.

To increase data throughput, each SISO processes four samples of a code vector at a time. Data_in[3:0] contains the first input sample, data_in[7:4] contains the second input sample, data_in[11:8] contains the third input sample and data_in[15:12] contains the fourth input sample. Similarly, data_out[3:0], data_out[7:4], data_out[11:8], and data_out[15:1] contain the first, second, third, and fourth output samples, respectively. Each sample is quantized to 4 bits using an unsigned number format as shown in Table 4.

It is important to note that the core output is a soft version of the entire code block after TPC decoding, not just the information bits. This means that external control and data selection circuitry must be used to strip out the information bits from the decoded data bus. This is accomplished by converting four parallel 4-bit samples to four serial 4-bit samples. If only the hard decisions are required, the external circuitry can further select the MSB of each 4-bit sample. The data stream can then be passed through a demux circuit that selects the first $k_x$ out of $n_x$ samples from each of the first $k_y$ rows. The selected bits are the desired decoded information bits.

*Table 4:* **Number Format for TPC Decoder Input Samples**

| 4-Bit Unsigned Number | Meaning |
|:---:|:---:|
| 0000 | Strongest 0 |
| … | |
| 0111 | Weakest 0 |
| 1000 | Weakest 1 |
| … | |
| 1111 | Strongest 1 |

The underflow and overflow outputs of the decoder are used to indicate an underflow or overflow in the internal output FIFO. These are major faults from which the decoder cannot gracefully recover. If either fault occurs, the decoder must be flushed for one code block before correct operation can resume.

An underflow will occur if the OutputRDY signal is Low and the output_en signal is asserted High. It is, therefore, recommended that the OutputRDY signal be ANDed externally to the output_en signal as a safety mechanism to protect against underflow.

The output FIFO has two pages of memory. Each page is capable of storing a single code block. Therefore, an overflow may occur if the number of code blocks in is greater than the number of code blocks out + 1. It is, therefore, imperative that an output code block be read before attempting to process a third code block.

Approximately five percent of the TPC decoder design is clocked at twice the speed of the rest of the design. The two-times clock (clk2x) must be rising-edge aligned with the system clock, sysclk. Generating clk2x from sysclk can be accomplished with the use of a Virtex-II digital clock manager (DCM). Refer to the *Virtex-II Platform FPGA Handbook* [Ref 2] for further details.

## Latency

The latency of the TPC decoder, defined as the number of clocks from start_in to OutputRDY, is given by

$$\Lambda = 1 \text{ code block delay} + N\left[\frac{n_x n_y}{2N_s} + \frac{(n_x + n_y)}{2} + 50\right] - \frac{n_x n_y}{4N_s} + 6$$

where $n_x$ and $n_y$ are the dimensions of the code block, N is the number of iterations, and $N_s$ is the number of SISOs.

## Data Rates

The achievable information bit rate, $R_b$, of the TPC decoder is given by

$$R_b = \left(\frac{k_x k_y}{\Lambda'}\right) f_{clk}$$

where $f_{clk}$ is the sysclk frequency, $k_x$ and $k_y$ are the dimensions of the information block, and $\Lambda'$ is the latency of the decoder from the time the last sample of the first code block is written to the input FIFO to the time the last sample of the first code block is written to the output FIFO.

Specifically, $\Lambda'$ is given by

$$\Lambda' = N\left[\frac{n_x n_y}{2N_s} + \frac{(n_x + n_y)}{2} + 50\right] + 6$$

Table 5 lists the maximum achievable data rates in Mbps for some common code types versus the number of full iterations using the four-SISO option of the decoder. The numbers are based on the above equations with a system clock frequency ($f_{clk}$) of 150 MHz. Note that the information bit rate scales linearly with the system clock frequency. Therefore, to obtain data rates for a clock frequency other than 150 MHz, scale the rates shown in Table 5 by the appropriate ratio.

*Table 5:* **Maximum Data Rates (Mbps) for a Subset of Codes**

| Code Type | | Code Rate | Number of Iterations | | | | |
|---|---|---|---|---|---|---|---|
| x | y | | 4 | 5 | 6 | 7 | 8 |
| (64,57) | (64,57) | 0.793 | 194 | 155 | 130 | 111 | 97 |
| (64,57) | (32,31) | 0.863 | 186 | 149 | 124 | 107 | 93 |
| (64,57) | (32,26) | 0.724 | 156 | 125 | 104 | 89 | 78 |
| (64,57) | (16,15) | 0.835 | 146 | 117 | 98 | 84 | 73 |
| (32,26) | (32,26) | 0.660 | 120 | 96 | 80 | 69 | 60 |
| (64,57) | (8,7) | 0.779 | 99 | 79 | 66 | 57 | 50 |
| (32,26) | (16,11) | 0.559 | 77 | 62 | 51 | 44 | 39 |
| (16,11) | (16,11) | 0.473 | 46 | 37 | 31 | 26 | 23 |

# Simulation Software

The user has the option of using the AHA Galaxy Simulation Toolkit software to investigate the core's performance as a function of the various input parameters. This software is not included with the TPC decoder core and must be acquired separately from AHA. Alternatively, the user may investigate the core's performance using the AHA 4501 Simulation Toolkit. This simulator may be obtained from the AHA website (www.aha.com) free of charge, but has the limitation that it does not support the (64,63) or (32,31) parity codes.

There are three main types of software simulation tools available for the Galaxy core. The first is the AHA Galaxy TPC Simulation Software which is a GUI based simulator that generates BER plots for a given set of configuration parameters. The second simulation tool is a C/C++ application programmers interface (API) that allows the user to integrate the Galaxy core into an existing C-based simulation environment. The third simulation tool is a MATLAB API that allows the user to integrate the Galaxy core into their MATLAB simulation environment. A complete description of the above tools is given in [Ref 1].

Figure 4 compares the BER performance of the $(64,57)^2$ TPC (obtained with the AHA TPC Simulator) to that of Viterbi with 4-bit soft decision decoding in an AWGN channel. As seen in the figure, TPC outperforms Viterbi by more than 2 dB at a BER of $10^{-6}$. The AHA TPC Simulator was configured to have 4-bit input quantization, 5-bit input and output resolution, a maximum of 5 iterations, and the x and y feedback parameters were both set to 16/32. The Viterbi numbers are based on the Xilinx® IEEE 802 Compatible Viterbi Decoder Core.
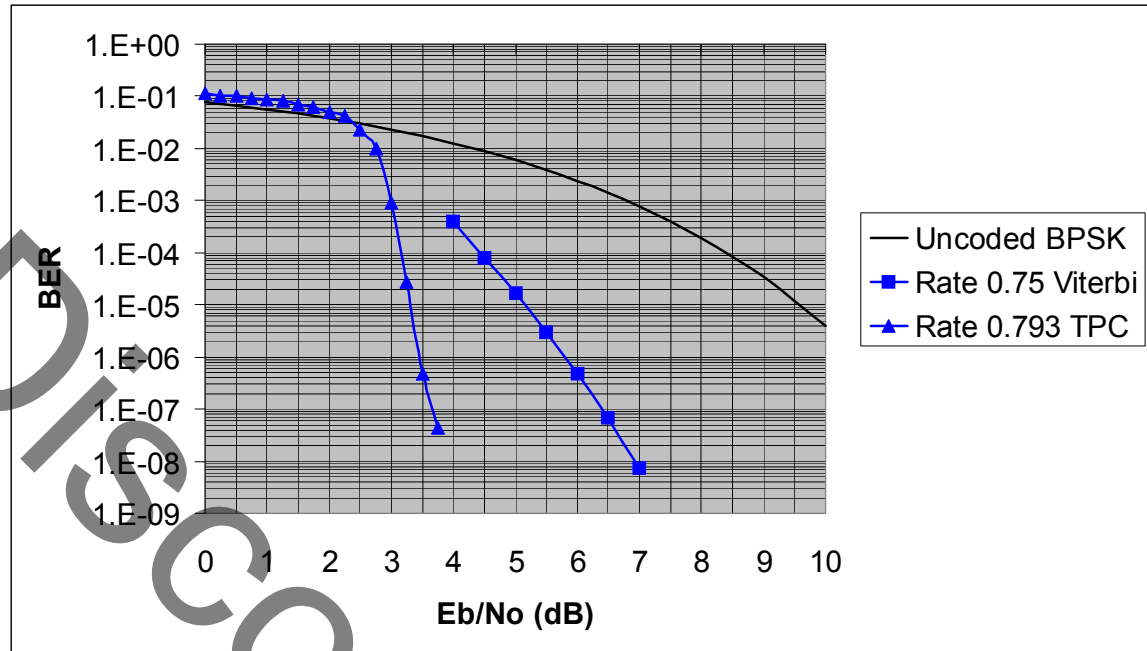
*Figure 4:* **BER Performance of TPC Compared to Viterbi**

## Core Resource Utilization

The four-SISO option of the core requires 3313 slices, 17 block RAMs, and 16 hardware multipliers when targeting Virtex-II/Pro FPGAs. The single-SISO option of the core requires 1062 slices, 13 block RAMs, and 4 hardware multipliers when targeting Virtex-II/Pro FPGAs.

When targeting Spartan-3 or Virtex-4 FPGAs, the non-RPM version of the core must be used. The non-RPM, four-SISO option of the core requires 4158 slices, 17 block RAMs, and 16 hardware multipliers. The non-RPM, single-SISO option of the core requires 1354 slices, 13 block RAMs, and 4 hardware multipliers.

## Performance Characteristics

Using the Xilinx static timing analysis tool (TRACE), both the single-SISO and four-SISO options of the core achieve a clock rate of 150 MHz in an XC2V1000-6 device (ADVANCED 1.111 2002-07-11 speed files) and a clock rate of 170 MHz in an XC2VP40-7 device (ADVANCED 1.62 2002-06-12 speed files). These speeds were obtained using a single instance of the RPM'ed core with its default placement.

The non-RPM version of the core achieves a clock rate of 111 MHz in an XC3S1500-5 (PRODUCTION 1.39 2007-08-14 speed files) and 172 MHz in an XC4VSX35-10 (PRODUCTION 1.66 2007-08-14 speed files).

## Design Verification

The TPC decoder core was verified through VHDL simulation and hardware testing. The SISO portion of the decoder was verified by generating random soft input data, applying the data simultaneously to the AHA SISO and the Xilinx optimized SISO and checking the outputs for a mismatch. The full TPC

decoder was verified by comparing its decoded output to that of the Galaxy MATLAB function given the same configuration parameters and soft input data.

The TPC decoder was tested in the lab using a Virtex-II prototyping board populated with an XC2V3000-6 FG676 device. A hardware test bench was developed to test the BER performance of the TPC decoder in an additive white Gaussian noise (AWGN) channel. The test bench consists of an LFSR-based data generator, the Xilinx TPC Encoder Core, the Xilinx AWGN Core, an adder, the TPC decoder core, a bit counter, an error counter and other miscellaneous support circuitry.

The LFSR-based data generator generates a random binary (0,1) sequence. This sequence is fed to the TPC encoder to create a code block. Bits from the code block are mapped from (0,1) to (+1,-1) and are added to noise samples from the AWGN core. The resulting n-bit samples are then mapped to 4-bit soft data using an approach similar to that used for soft Viterbi decoding. Next, a serial-to-parallel converter is used to convert the serial 4-bit samples into a single 16-bit bus containing four 4-bit data samples. The 4-sample blocks are then fed to the TPC decoder data_in[15:0] port.

The TPC decoder outputs 4-sample blocks of the decoded code block. The start_out signal is used to detect the start of a new code block out. Note that the data output is for an entire code block, not just the information block. External control logic is used to extract the information bits from the output block. The decoded information bits are then compared to the original input to the TPC encoder and the error counter counts the number of mismatches. The BER is then computed as the ratio of the error count to the bit count.

Figure 5 shows the measured BER performance of the (64,57) x (64,57) code versus the number of iterations. Figure 6 compares the performance of the (64,57) x (64,57) code to that of the (64,57) x (64,63) and (64,57) x (32,26) codes with the number of iterations equal to six. In all cases, xfeedback[4:0] and yfeedback[4:0] were set to a value of 16.
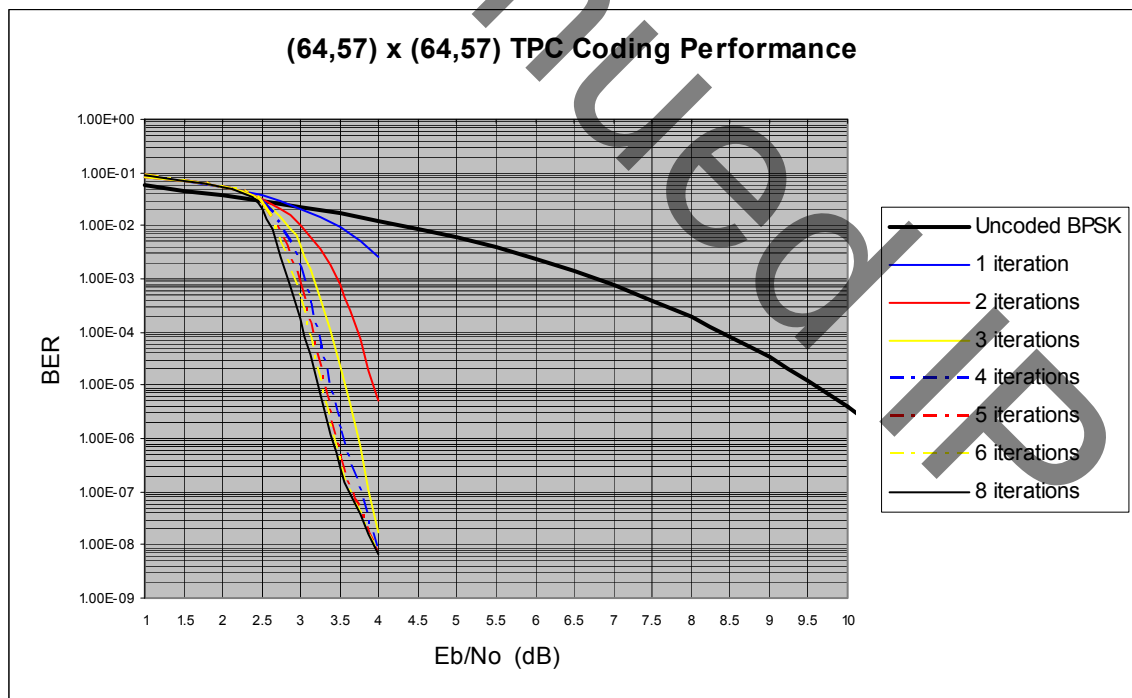


*Figure 5:* **BER Performance of (64,57) x (64,57) Code**
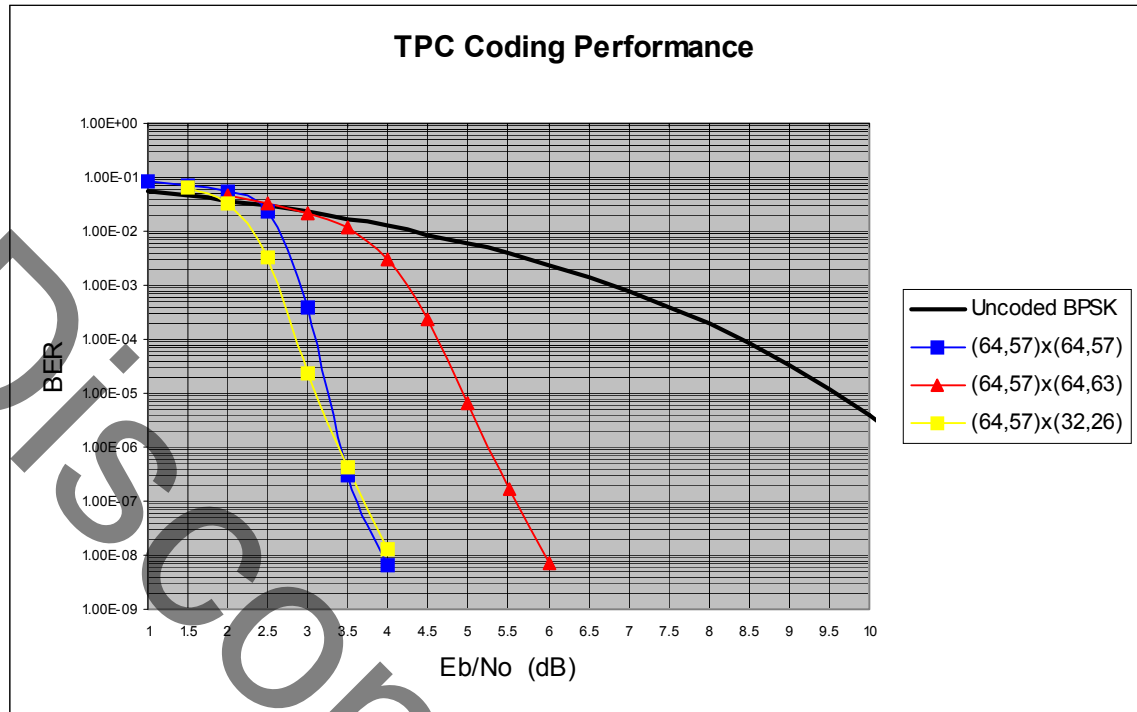
**TPC Coding Performance**



*Figure 6:* **BER Performance at Six Iterations**

## References

1. Advanced Hardware Architectures, Inc., *AHA Galaxy Simulation Tool Kit User's Guide,*

2. *Xilinx® Virtex-II Platform FPGA Handbook*, UG002 v2.2 (November 5, 2007)

## Support

Xilinx provides technical support for this LogiCORE product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled *DO NOT MODIFY*.

## Ordering Information

For details about licensing and ordering, see the TPC Decoder product page.

## Revision History

The following table shows the revision history for this document:

| Date | Version | Description of Revisions |
|------|---------|--------------------------|
| 01/21/03 | 1.0 | Initial Xilinx release. |

| Date | Version | Description of Revisions |
|------|---------|--------------------------|
| 06/30/08 | 1.1 | Update for core version 1.1. |
| 02/03/15 | 1.2 | Updated ordering information. |

## Notice of Disclaimer

Xilinx is providing this design, code, or information (collectively, the "Information") to you "**AS-IS**" with no warranty of any kind, express or implied. Xilinx makes no representation that the Information, or any particular implementation thereof, is free from any claims of infringement. You are responsible for obtaining any rights you may require for any implementation based on the Information. XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE INFORMATION OR ANY IMPLEMENTATION BASED THEREON, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF INFRINGEMENT AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Except as stated herein, none of the Information may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx.