# IEEE 802.16 Compatible Turbo Product Code Encoder v1.0

DS211 February 3, 2015 **Product Specification**

## Features

- Performs TPC encoding as defined 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.
- Block sizes from 64 bits to 4 Kbits, 64 possible product codes
- Fully synchronous design using a single clock
- Separate input and output enables allow flexible handling of data flow
- Up to 8 Kbits of internal buffering
- Low latency (13 clocks) independent of code type
- Requires 80 slices and 2 block RAMs
- Maximum clock rate and encoded data rate of 285 MHz in Virtex-II (-6), 340 MHz in Virtex-II Pro (-7), 200 MHz in Spartan-3 (-5), and 312 MHz in Virtex-4 (-10) FPGAs

## Applications

The TPC Encoder core is designed to operate in local multi-point distribution service (LMDS) and multi-channel multi-point distribution service (MMDS) communication systems.

The TPC Encoder 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 | I/O | LUTs | FFs | Block RAMs |
| | 17 | 112 | 136 | 2 |
| Special Features | RPM core | | | |
| **Provided with Core** | | | | |
| Documentation | Product Specification | | | |
| Design File Formats | VHDL | | | |
| 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 | Synplicity® Synplify Pro® 7.1 | | | |
| **Support** | | | | |
| Provided by Xilinx, Inc. | | | | |

## Pinout

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

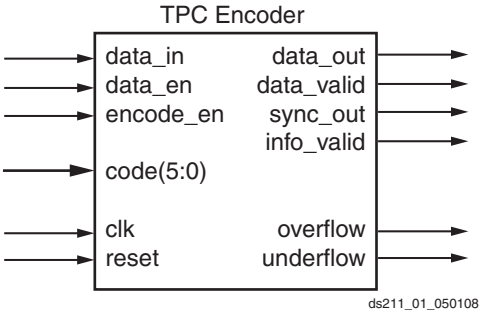


*Figure 1:* **Core Schematic Symbol**

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

| Name | Direction | Description |
|---|---|---|
| data_in | Input | Input data to be encoded |
| data_en | Input | Active High enable for input data |
| encode_en | Input | Active High enable causing encoder to generate output data |
| code(5:0) | Input | Specifies 1 of 64 possible product codes |
| clk | Input | Clock that triggers all sequential elements on its rising edge |
| reset | Input | Active High synchronous reset |
| data_out | Output | Encoded output data |
| data_valid | Output | High when output data is valid |
| sync_out | Output | High during the first bit of each output code block |
| info_valid | Output | High when the output data is an information bit |
| overflow | Output | Active High fault indicating an overflow in internal FIFO |
| underflow | Output | Active High fault indicating an underflow in internal FIFO |

## Functional Description

This core performs two-dimensional product code encoding with constituent codes that are either extended Hamming codes or simple parity codes. 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. This process is illustrated in Figure 2 for an (8,4) × (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. To achieve the lowest possible latency, the encoded data is read row-by-row from left to right. For latency considerations, the core does not support the optional block interleaving listed in the IEEE 802.16 standard. The latency of the core from encode_en to data_valid is 13 clocks.

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

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

The core supports both extended Hamming and parity only constituent codes. 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 simply 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 encoder core supports 8 possible code types for each of the constituent codes for a total of 64 possible product codes. The code(5:0) input to the core is used to specify the code type as shown in Table 3. The current data block encoding must be entirely processed before changing the code(5:0). The code in the x dimension (i. e., the code for each row) is specified with the control bits code(5:3) and the code in the y dimension (i. e., the code for each column) is specified with the control bits code(2:0). For example, setting code(5:0) = "010101" configures the encoder for the (32,26) × (16,15) code.

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

| Code(5:3)/(2:0) | X/Y Code | Code Type |
|---|---|---|
| 000 | (8,4) | Extended Hamming |
| 001 | (16,11) | Extended Hamming |
| 010 | (32,26) | Extended Hamming |
| 011 | (64,57) | Extended Hamming |
| 100 | (8,7) | Parity Only |
| 101 | (16,15) | Parity Only |
| 110 | (32,31) | Parity Only |
| 111 | (64,63) | Parity Only |

It is important to note that the underflow and overflow status signals are major faults from which the encoder cannot recover. Therefore, it is imperative that the user enable the input and output appropriately to avoid an underflow or overflow condition. For best results, the reset should be asserted upon

power up and held High until the core is ready for use. After reset is deasserted, the data_en and encode_en signals become valid. The most straightforward approach to controlling the core is to enable the input for the first $k_x k_y$ clocks out of every $n_x n_y$ clocks. With this type of input, encode_en can be asserted a constant High from the first time data_en is High onward.

This mode results in a continuous data output with a bursty data input. For example, with the (8,4) × (8,4) code, the user would provide valid data_in and assert data_en High for 16 clocks. At the same time data_en is asserted High, encode_en is asserted High. For the next 48 clocks, data_in is invalid and data_en is Low while encode_en remains High. This sequence is repeated for each information block processed as illustrated in Figure 3.

As long as the input and output enables are operated in a manner to avoid underflow/overflow, there is no restriction on the enable sequencing. For example, the core has been simulated under conditions of the data_en and encode_en signals being generated from pseudo-random bit sequences (PRBS) with external FIFO level checking. Note that the FIFO is 8K deep, but starts in an almost empty state. It will therefore take longer to cause an overflow than an underflow, but either may occur if the enables are not rate matched.

A fundamental limitation of the core is that there is no way to reset the contents of the block RAM used in the y encoder. It is therefore not possible to issue a reset after encode_en has been asserted High without getting errors in the first encoded data block after reset. The encoded data block will be incorrect because the initial states of the LFSRs for the column encoders will not be zero as expected. Under these circumstances, the encoder must be flushed for one data block before correct encoding can resume.

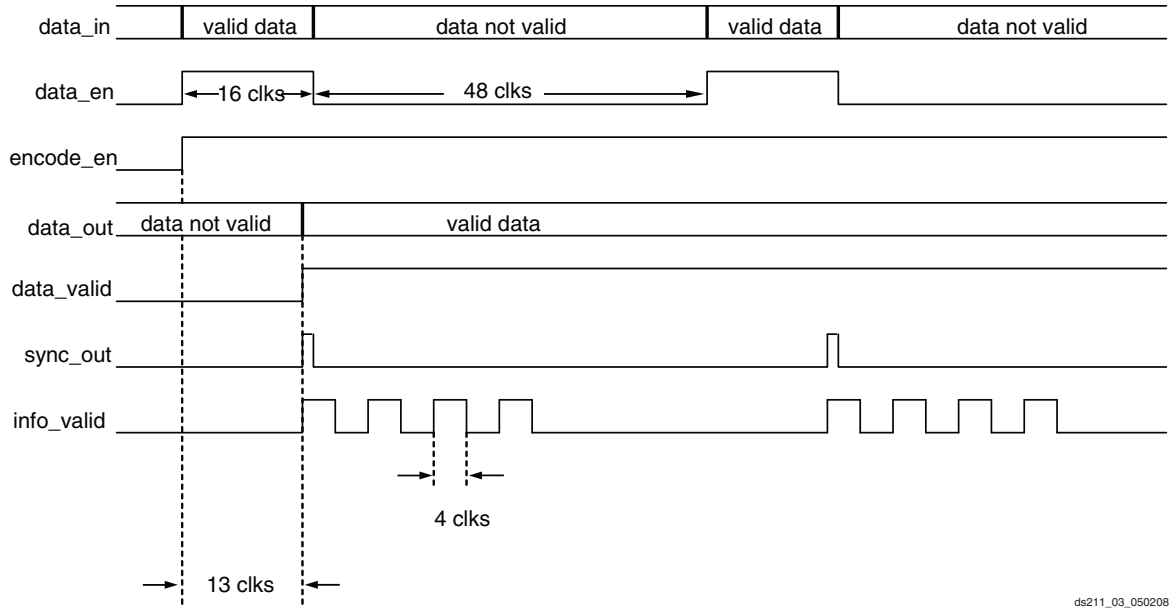


*Figure 3:* **Timing Diagram for (8,4)-by-(8,4) Example**

## Core Architecture

A full description of the core architecture is provided upon purchase of the core. This includes an overview of the design approach and a top-level block diagram that corresponds to the top-level VHDL source code. This description helps the user in understanding the internal details of the core and facilitates the user's ability to make modifications to the design if desired.

## Core Parameters

The core has three generic parameters: Target, bxloc and byloc. The Target parameter must be set to "Virtex-II." The bxloc and byloc parameters control the placement of the block RAM used in the core. Because the design is an RPM, the placement of the slice elements are controlled through the use of either an RLOC or RLOC_ORIGIN attribute on the core. The (bxloc, byloc) pair have the same functionality as an RLOC_ORIGIN attribute, but applied to the block RAM. Figure 4 shows the default placement of the core as viewed in the Xilinx® Floorplanner™.

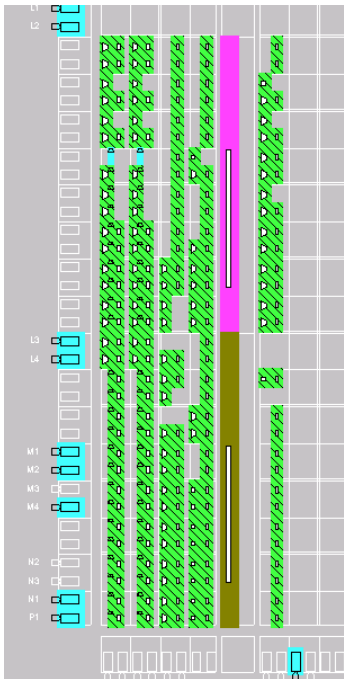

*Figure 4:* **Core Layout as Viewed in Xilinx Floorplanner**

## Core Resource Utilization

The RPM version of the core requires 80 slices (in a 5 by 16 rectangle) and 2 block RAMs. For optimal performance, the slice and block RAM origins must be set to achieve the same relative placement as illustrated in Figure 4. In particular, the user may choose any two block RAMs having the same x coordinate and offset by one in the y direction (e. g., RAMB16_X0Y0, RAMB16_X0Y1). The user must then place the slice origin of the core at the location that is aligned with the lower edge of the lower block RAM and four slices to the left (e. g., X0Y0 when using RAMB16_X0Y0). Obtaining the optimal values for the slice and block RAM origins is best done using the FPGA Editor. For example, in an XC2V1000 device, if the block RAM origin is set to RAMB16_X2Y8 (bxloc = 2, byloc = 8), then the slice origin should be set to X32Y64.

When targeting Spartan-3 or Virtex-4 FPGAs, a non-RPM version of the core must be used. The non-RPM version is obtained by commenting out all location constraint information from the VHDL source code. This includes RLOC, RLOC_ORIGIN, LOC, and BEL attributes. Using either an area group constraint or maximum packing factor in MAP allows the resource utilization to remain at 80 slices and 2 block RAMs.

## Performance Characteristics

Using the Xilinx static timing analysis tool (TRACE), the RPM version of the core achieves a clock rate of 285 MHz in an XC2V1000-6 device (ADVANCED 1.111 2002-07-11 speed files) and a clock rate of 340 MHz in an XC2VP20-7 device (ADVANCED 1.62 2002-06-12 speed files). These speeds were obtained using a single instance of the core with an RLOC_ORIGIN of X0Y0 and a block RAM origin of RAMB16_X0Y0.

The non-RPM version of the core achieves a clock rate of 200 MHz in an XC3S1000-5 (PRODUCTION 1.39 2007-08-14 speed files) and 312 MHz in an XC4VSX35-10 (PRODUCTION 1.66 2007-08-14 speed files).

## Design Verification

The core was verified using a self-checking test bench written in VHDL and simulated with ModelSim. The self-checking test bench compares the output of the core to a behavioral model that uses a *brute force* approach based on parity check equations for each code bit. All 64 possible codes were verified for both the deterministic and pseudo-random enabling schemes described in "Functional Description."

## 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 Encoder product page.

## Revision History

The following table shows the revision history for this document:

| Date | Version | Description of Revisions |
|---|---|---|
| 10/30/02 | 1.0 | Initial Xilinx release. |
| 06/30/08 | 2.0 | Updated document for Virtex-4 and Spartan-3 FPGAs. |
| 02/03/15 | 3.0 | Updated ordering information. |

## Notice o f Disclaimer