# KeyStone Architecture
# Turbo Decoder Coprocessor (TCP3d)

# User Guide

Literature Number: SPRUGS0
November 2010

# Release History

| Release | Date | Chapter/Topic | Description/Comments |
|---|---|---|---|
| 1.0 | November 2010 | All | Initial Release |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Contents

**Chapter 4**

## List of Tables

## List of Figures

# Preface

## About This Manual

This document describes the functionality, operational details, and programming information for the Turbo-Decoder Coprocessor 3 (TCP3D) in KeyStone architecture devices.

## Notational Conventions

This document uses the following conventions:

- Commands and keywords are in **boldface** font.
- Arguments for which you supply values are in *italic* font.
- Terminal sessions and information the system displays are in `screen font`.
- Information you must enter is in **`boldface screen font`**.
- Elements in square brackets ([ ]) are optional.

Notes use the following conventions:

**Note—**Means reader take note. Notes contain helpful suggestions or references to material not covered in the publication.

The information in a caution or a warning is provided for your protection. Please read each caution and warning carefully.

**CAUTION—**Indicates the possibility of service interruption if precautions are not taken.

**WARNING—**Indicates the possibility of damage to equipment if precautions are not taken.

## Related Documentation

3GPP TS 25.212 V7.5.0 (2007-05): “3GPP Technical Specification Group Radio Access Network; Multiplexing and channel coding (FDD) (Release 7); Section 4.2.3.2 Turbo coding”

3GPP TS 36.212 V2.0.0(2007-09): “3GPP Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (Release 8); Section 5.1.3.2 Turbo coding”

IEEE 802.16-2004: “Part16: Air Interface for Fixed Broadband Wireless Access Systems; 8.2.1.2.4 Convolutional Turbo Codes”

IEEE P802.16-2004/Cor1/D5: “Corrigendum to IEEE Standard for Local and Metropolitan Area Networks: Part16: Air Interface for Fixed Broadband Wireless Access Systems; 8.4.9.3 Convolutional Turbo Codes”

Change request to - R1-075111 36.212 CR001; 3GPP TSG-RAN WG1 #51 Jeju, Korea, November 5 – 9, 2007

## Trademarks

TMS320C66x and C66x are trademarks of Texas Instruments Incorporated.

All other brand names and trademarks mentioned in this document are the property of Texas Instruments Incorporated or their respective owners, as applicable.

Chapter 1

# Introduction

## 1.1 Terminology Used in This Document

Table 1-1 defines the important acronyms used in this document.

**Table 1-1 Terminology**

| Acronym | Definition |
|---|---|
| 3GPP | 3rd generation partnership project |
| CRC | Cyclic redundancy check |
| CTC | Convolutional turbo code |
| LLR | Log likelihood ratio |
| LTE | Long term evolution |
| MAP | Maximum a posteriori |
| RSCC | Recursive systematic convolutional codes |
| SNR | Signal-to-noise ratio |
| SW | Sliding window |
| TCP | Turbo decoder coprocessor |
| WiMAX | Worldwide interoperability for microwave access |
| **End of Table 1-1** | |

## 1.2 Overview

Wireless base stations currently use cost efficient DSPs for their implementations. The system cost of the base station is a function of the number of channels that the DSP can support. As the wireless systems are converted from second generation to third and forth generation systems, the system complexity has increased by an order of magnitude. The 3G wireless systems require an iterative turbo decoder as one of its forward error correction algorithms. A very cost effective and synergistic solution has been designed using a DSP and a turbo coprocessor (TCP3D).

The Turbo-Decoder Coprocessor 3 (TCP3D) is a programmable peripheral for decoding of 3GPP, LTE, and WiMAX turbo codes, integrated into Texas Instruments' DSP device. The inputs into the TCP3D are channel soft decisions for systematic and parity bits, and the outputs are hard decisions.

Turbo codes used in 3G standards are based on a parallel concatenation of Recursive Systematic Convolutional Codes (RSCC), each with a binary input. The optional Convolutional Turbo Code (CTC) included in the IEEE 802.16e standard is based on duo-binary turbo codes.

The primary focus of this document is to describe how to use the TCP3D.

**Note**—Unless otherwise noted, references to 3GPP standards in this document imply support for the 3GPP standard up to Release 7. This includes WCDMA, HSUPA, HSUPA+, and TD_SCDMA.

## 1.3 Features

- Native Code rate: 1/3
- Radix-4 binary and duo-binary MAP decoders
- Dual MAP decoders for non-contentious interleavers
- Split decoder mode: can function as 2 independent, single MAP decoders
- Max star and max log-map algorithms
- Belief propagation instead of prolog/prolog reduction (increased throughput)
- Double-buffer input memory for lower latency transfers (except in split mode)
- 128-bit data bus for higher throughput transfers
- Input data bit width: 6 bits
- Programmable output hard decision bit ordering within a 32-bit word: 0 to 31 or 31 to 0
- Soft outputs provided for systematics, parity 0 and parity 1 for all modes except split mode
- Systematic soft outputs provided for split mode
- Extrinsic scaling per MAP for up to 8 iterations (both max and max star)
- Block sizes supported: 40 to 8192
- Programmable sliding window sizes
- Programmable min and max number of iterations: 1 to 15
- Programmable SNR stopping criterion: 0 to 20 dB threshold
- LTE CRC stopping criterion
- LTE, 3GPP and WiMAX hardware interleaver generators
- Channel quality indication
- Debug mode
- Emulation support
- Low DSP pre-processing load
- Runs in parallel with DSP
- WiMAX support includes “tail-biting”

## 1.4 Theory

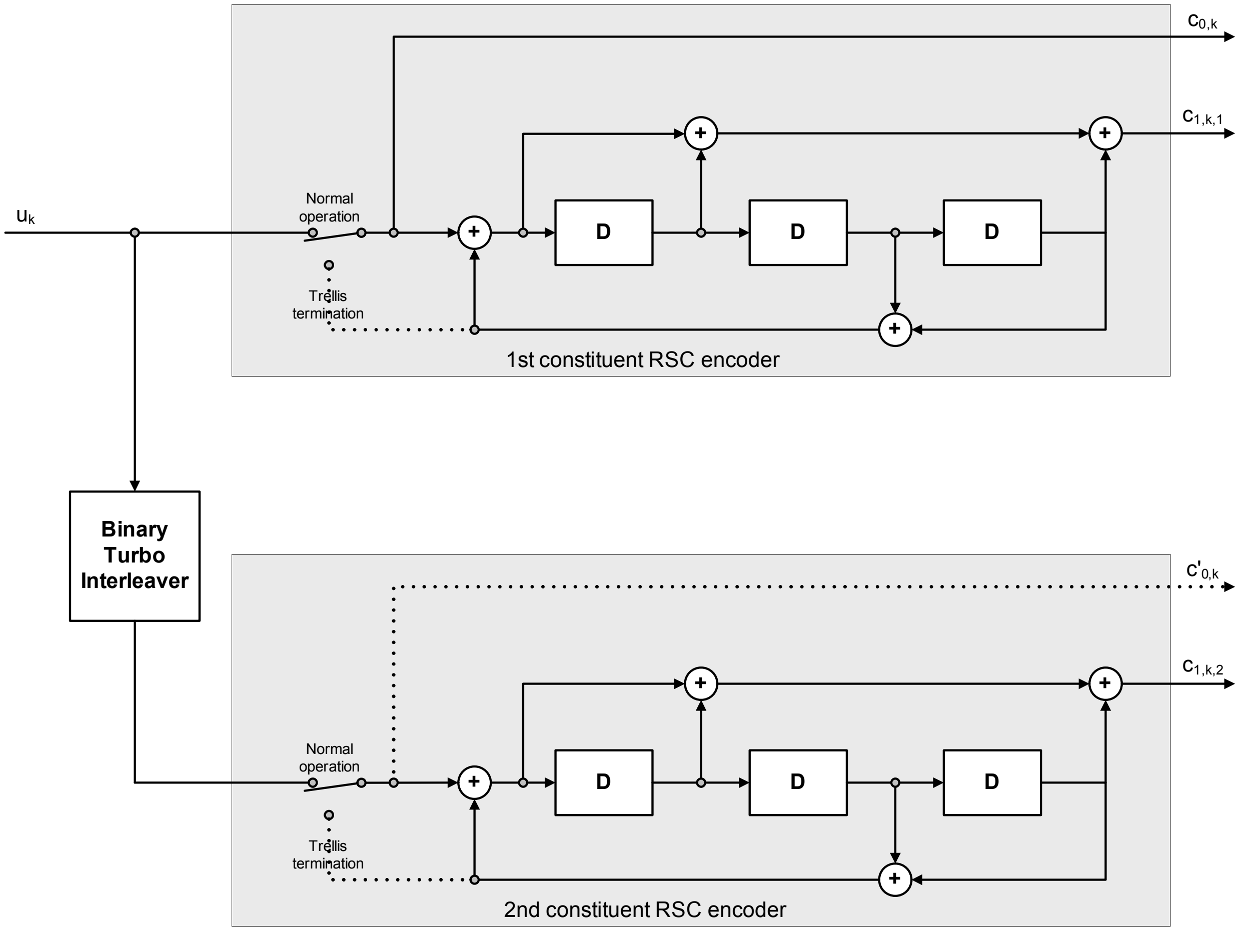
### 1.4.1 Binary Turbo Codes

Binary turbo codes are block codes, i.e. a code block of K information bits is encoded and in the process, the output block of N (N>K) encoded bits is generated (a code word or code block). Binary turbo codes have been adopted in several 3GPP standards.

#### 1.4.1.1 Encoder

The encoder consists of two Recursive Systematic Convolutional (RSC) encoders (Figure 1-1). Each RSC encoder takes as an input one information bit, $u_k$, at a time and generates one parity bit (native code rate 1/3 for 3GPP).

**Figure 1-1 Block Diagram of the Binary Turbo Encoder**



The first encoder operates on the incoming binary data sequence $u_k$ ($k = 1,2,...,K$), whereas the second encoder operates on the interleaved input sequence. The original input bit appears unchanged at the encoder output and is denoted as $c_{0,k}$ (systematic bit). In the case of the code rate 1/3, the turbo encoder for each input bit, $u_k$, generates three output bits: $c_{0,k}$, $c_{1,k}$, $c_{2,k}$. The interleaver design depends on the particular standard.

The initial trellis state for both constituent encoders is set to be the zero-state. There is also a provision in the standards to drive the last state of the trellis into zero-state. This is accomplished by sending some additional bits after all the information bits have been encoded. The dotted lines in Figure 1-1 become active during the trellis termination period. During the first three bit-clock cycles of the trellis termination period, the control switch of the first RCS encoder is kept in the lower position and the output of the first encoder is sent to the line. During the second three bit-clock cycles of the trellis termination period, the control switch of the second RCS encoder is kept in the lower position and the output of the second encoder is sent to the line. The actual pattern of the encoded bits sent out during the trellis termination period depends on the particular standard.

The encoder is normally followed by a puncturing or a rate matching block which determines how the encoded bits will be used by the following block of the transmitter.

#### 1.4.1.2 Decoder

The turbo decoder reconstructs the bits transmitted through the channel. The inputs to the turbo decoder are the channel Log-Likelihood-Ratio (LLR) values for each encoded bit passed to it by the previous receiver block. For the *n*-th encoded bit, $c_n$, the channel LLR (“soft channel output”) is commonly defined as follows:

$$\Lambda_n^c \overset{def}{=} \ln \frac{p(Y_n \mid c_n = 1)}{p(Y_n \mid c_n = 0)}$$

where $Y_n$ is the received signaling element carrying bit $c_n$.

Some authors define LLR as:

$$\ln[p(Y_n \mid c_n = 0) / p(Y_n \mid c_n = 1)]$$

in which case the sign of LLR is the opposite to what we have here, but the absolute value is the same.

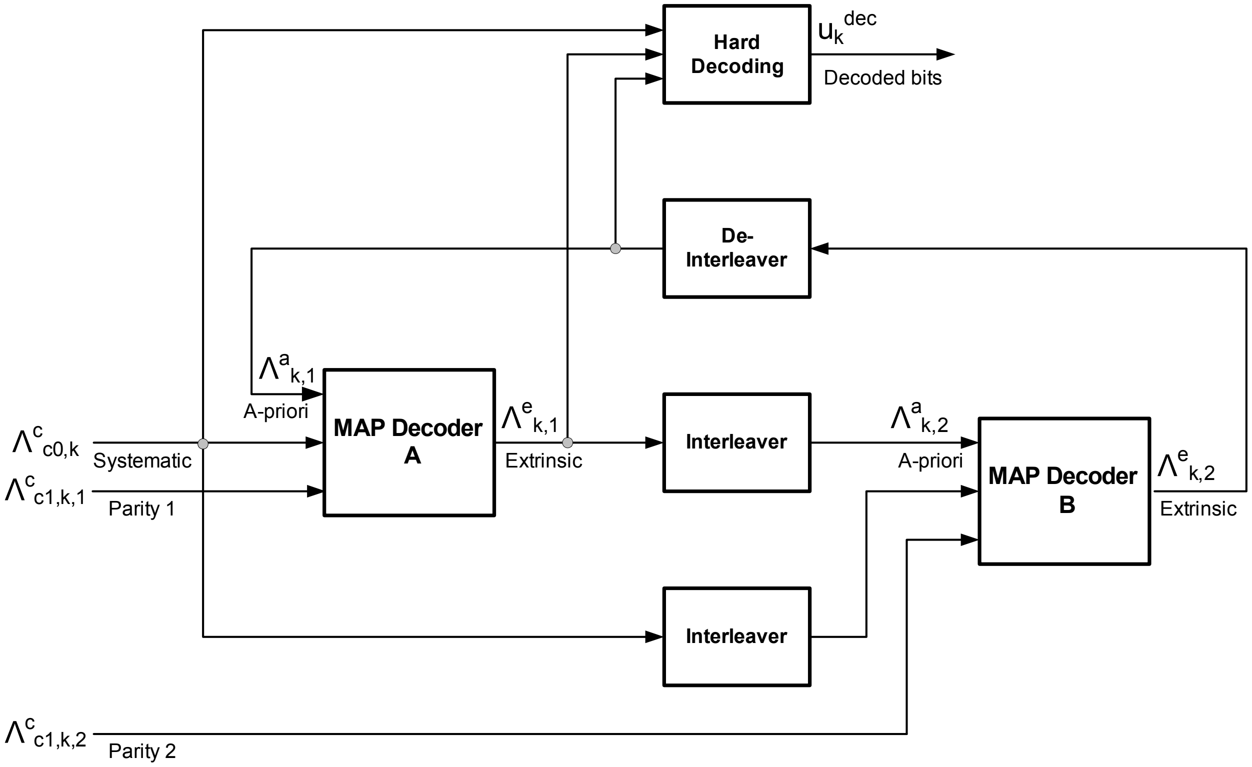
Decoding is done in an iterative fashion in which two component soft-input soft-output (SISO) decoders operate cooperatively. The goal of each component turbo decoder is to maximize each of the following estimated LLRs, corresponding to the block of K information bits:

$$\Lambda_k = \ln \frac{p(u_k = 1 \mid \mathbf{Y})}{p(u_k = 0 \mid \mathbf{Y})}, \qquad k = 1,2,\ldots,K$$

where $\mathbf{Y}$ is the array of all signal elements corresponding to the current code word related to a particular decoder. The quantities in the numerator and denominator represent a-posteriori probability density functions. Therefore, it is said that each component decoder implements Maximum A-posteriori Probability (MAP) decoding algorithm.

The overall architecture of the binary turbo decoder is shown in Figure 1-2:

**Figure 1-2 Block Diagram of the Binary Turbo Decoder**



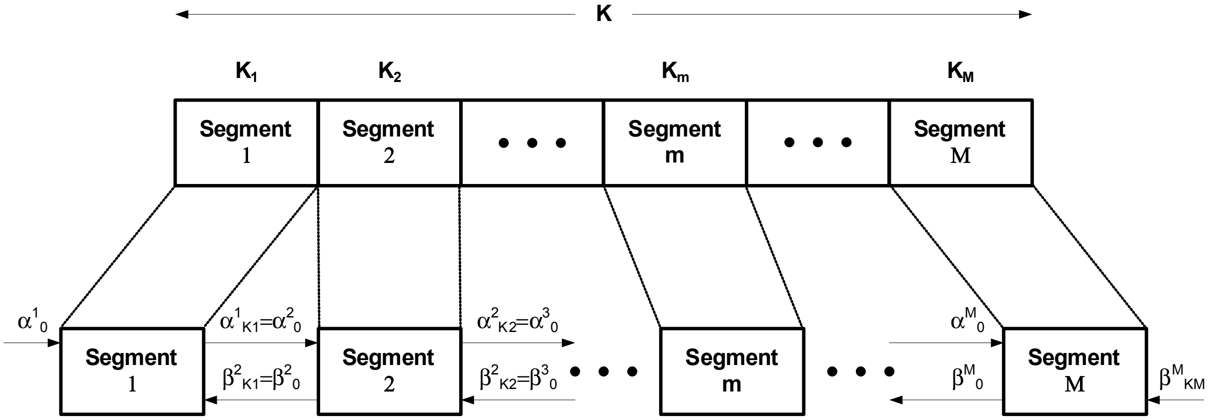
Each SISO decoder implements the MAP algorithm in which all of its internal variables ($\alpha$, $\beta$, $\gamma$) represent logarithms of their original values, therefore the name Log-MAP algorithm.

### 1.4.1.2.1 Segmented Decoding

In order to enable more efficient decoder hardware implementations long code blocks are split into several smaller segments (also referred to as *sliding windows*, SW, in the subsequent sections), which are decoded separately. They may be also decoded in parallel, which leads to higher throughput. Segments may be of different sizes. In some approaches (including earlier versions of C6x turbo coprocessors, TCP, TCP2) segments overlap. The overlapping portion of the adjacent segments (prolog) is used to compute initial values for the forward and backward metrics of the segment. Another approach is applied in this design, however. Namely, there is no prolog here. Instead, boundary metrics are passed between neighboring segments. This is often referred to as the *belief propagation* approach.

Figure 1-3 illustrates the concept of the segmented decoding method.

**Figure 1-3 Segmented Decoding**



If all segments are decoded in parallel then the boundary metrics that are passed between neighboring segments have been computed in the previous iteration. However, if this is not the case, i.e. only some of the segments are decoded in parallel, then some boundary metrics required for the decoding of a particular segment may have been computed in the current iteration. The bottom line is that the segment decoding algorithm always uses the most recent boundary metrics from the neighboring segments, which may be from the previous or the current iteration.

### 1.4.2 Duo-Binary Turbo Codes

Duo-binary turbo codes are block codes, i.e. a code block of *K* information bits (K/2 information duo-binary symbols) is encoded and in the process the output block of *N* (*N*>*K*) encoded bits is generated (a code word). They have been adopted in several telecommunication standards including IEEE 802.16 (WiMAX). Duo-binary turbo codes have two distinctive characteristics:

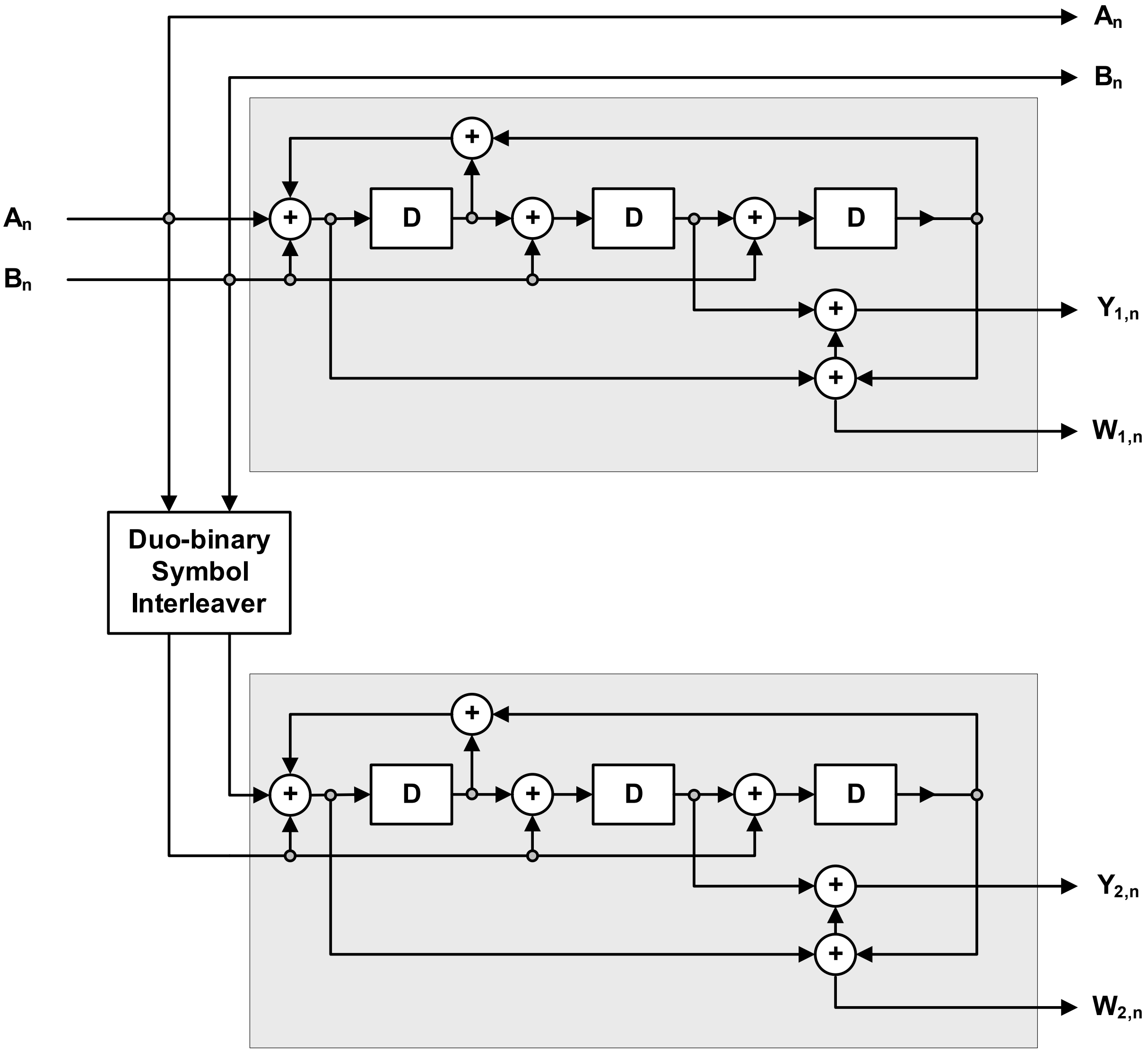
- Operation on bit pairs (duo-binary symbols) rather than on individual bits
- Use of the circular (tail-biting) Recursive Systematic Convolutional (RSC) component codes.

#### 1.4.2.1 Encoder

The encoder consists of two circular RSC encoders. First RSC encoder takes as an input a pair of information bits, $(A_n,B_n)$, representing a duo-binary symbol, at a time and generates two output parity bits $(Y_{1,n}, W_{1,n})$. Second RSC encoder takes as an input interleaved duo-binary symbol $(A_n,B_n)$, and generates two output parity bits $(Y_{2,n}, W_{2,n})$.

This encoding process is shown in Figure 1-4.

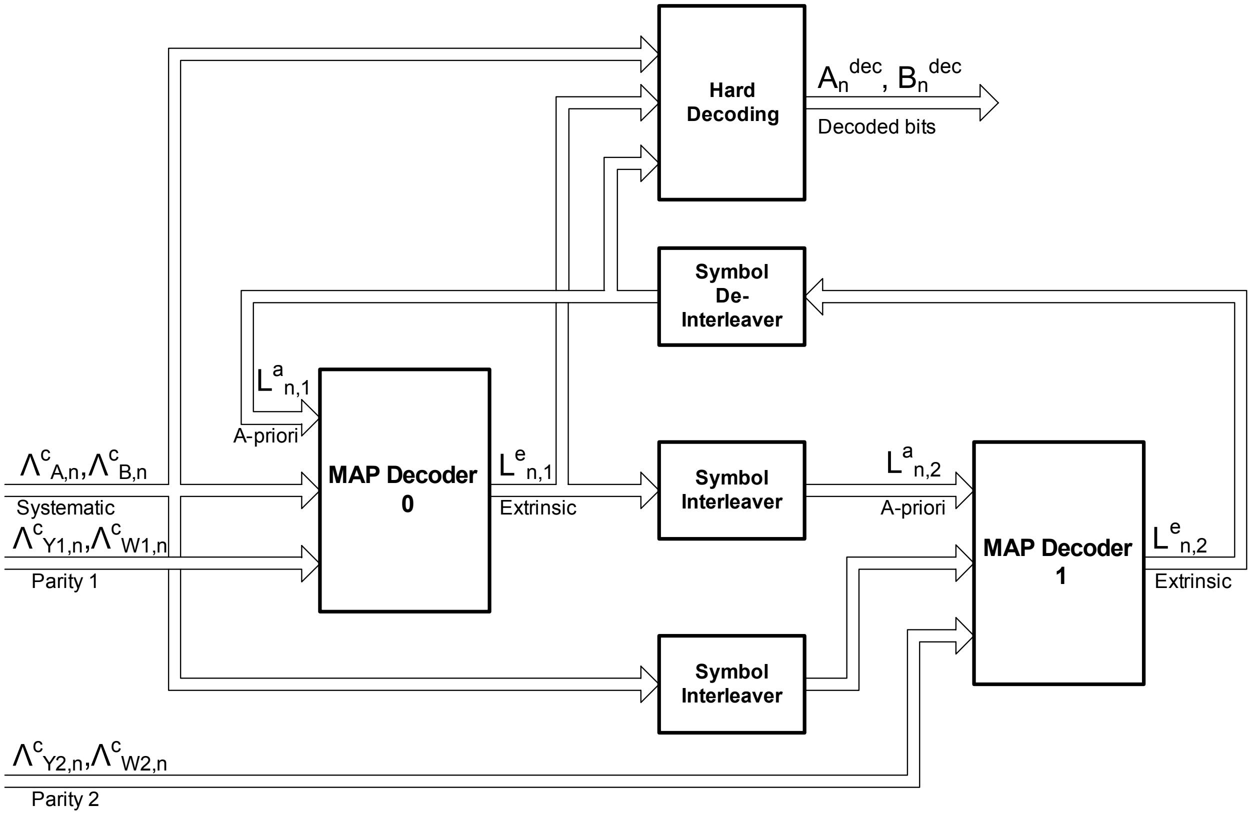
**Figure 1-4 Block Diagram of the Duo-binary Turbo Encoder**



### 1.4.2.2 Decoder

Duo-binary turbo decoder is very similar to the binary turbo decoder described previously. The main difference is that some of the decoding algorithm variables are related to the duo-binary symbols instead of to bits. Another difference is in the nature of the component convolutional codes. The beginning and ending states of the trellis are not known. However, they are the same. The decoder exploits this *tail-biting* trellis property. A block diagram of the duo-binary turbo decoder is shown in Figure 1-5.

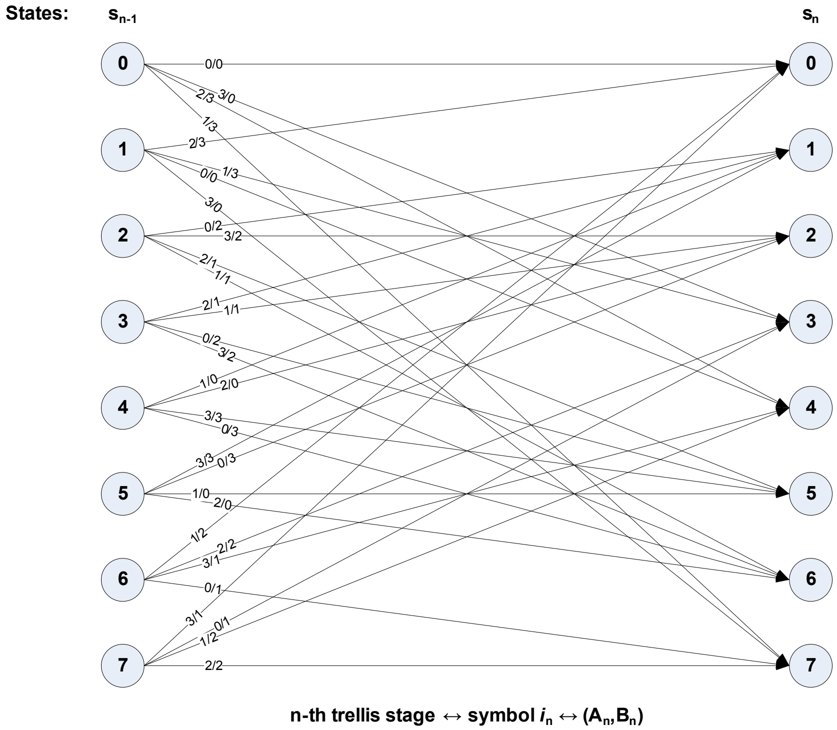
**Figure 1-5 Block Diagram of the Duo-binary Turbo Decoder**



Letter *n* denotes the duo-binary symbol index (*n* = 1, 2,..., *K/2, K* is the number of information bits). Letter L indicates quantities proportional to log-likelihoods, whereas $\Lambda$ indicates quantities proportional to log-likelihood ratios (as before). Double arrows indicate that duo-binary symbol related values (four symbol values or dual bit values) are passed between the blocks, rather than the single bit related quantities.

The trellis diagram of the duo-binary code is shown in Figure 1-6.

**Figure 1-6 Duo-Binary Trellis Transition Diagram**



### 1.4.2.2.1 Segmented Decoding

The same *no-prolog* approach described in the case of the binary turbo codes is adopted here as well. Boundary metrics are passed between neighboring segments in a similar fashion. Figure 1-3, which illustrates the concept of the binary segment turbo decoding method, is applicable to the duo-binary turbo decoders, too. If all the segments are decoded in parallel, then the boundary metrics that are passed between neighboring segments have been computed in the previous iteration. However, if this is not the case, i.e. only some of the segments are decoded in parallel, then some boundary metrics required for the decoding of a particular segment may have been computed in the current iteration. The bottom line is that the segmented decoding algorithm always uses the most recent boundary metrics from the neighboring segments, which may be from the previous or the current iteration.

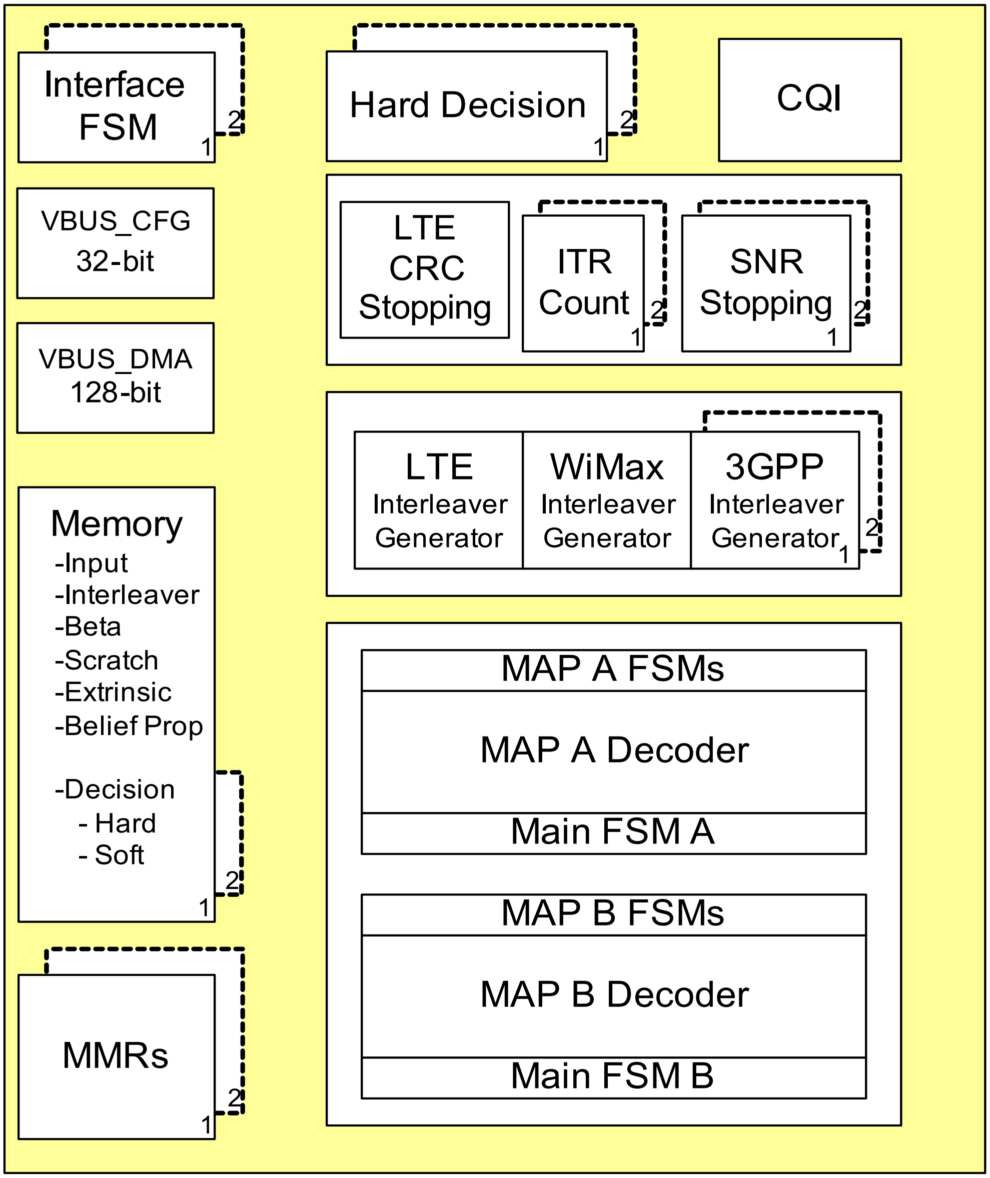
Chapter 2

# Architecture

## 2.1 Block Diagram

The functional partitioning of the TCP3D is shown in Figure 2-1. Note that some internal blocks are duplicated and used for *split* mode only.

**Figure 2-1 TCP3D Block Diagram**



## 2.2 VBUS Interfaces

The TCP3D interfaces to the system via two VBUSP interfaces. A 32-bit configuration VBUSP interface (VBUS_CFG) is provided as the interface to the configuration and status registers. A 128-bit VBUSP interface (VBUS_DMA) is provided as the DMA interface to the TCP3D. The per code block input data, output data, and configuration registers are transferred to the TCP3D over this interface.

The VBUS_CFG occupies a 1KB space in system memory, while the VBUS_DMA occupies a 1MB space. Absolute addresses are determined at the chip level.

The VBUS_CFG supports only aligned 32-bit accesses to the memory mapped registers. No byte, 16-bit, or misaligned accesses are supported.

The VBUS_DMA does not require aligned addresses. The VBUSP address is aligned to a 128-bit boundary on this bus and just uses the VBUSP byte enables to determine which bytes are accessed. With the exception of big endian data swapping, it does not provide any data steering. As an example for little endian mode, data written into bits [127:120] of an input configuration register must always appear on VBUSP data bits [127:120]. Big endian data swapping is supported regardless of the address alignment but is fixed per the schemes shown in Section 3.3.9 "Endianess Considerations" on page 3-22.

Chapter 3

# Usage

This section describes how to setup and use the TCP3D from a user’s point of view.

**Note—**Throughout the rest of the document, Parity 0 refers to non-interleaved parity bits and Parity 1 refers to interleaved parity bits. This is slightly different notation than was used in the introduction section.

## 3.1 Usage Overview

The overall usage and data flow for the TCP3D is described in the following steps.

1. DSP performs TCP3D initialization (data processing mode, etc.) via CFG VBUS interface after TCP3D is reset.
2. DSP prepares input data and parameters for decoding. This includes:
    a. Input LLR scaling, saturation, and packing
    b. External interleaver table generation (optional)
    c. Input configuration parameter calculations including (but not limited to):
        i. Calculation of interleaver table generation parameters (LTE and WiMAX only) if table is generated internally by TCP3D
        ii. Determination of the lengths and the numbers of sliding windows
        iii. Calculation of initial beta states from tail bits (3GPP only)
3. DSP configures and initiates EDMA3 to transfer the input configuration registers as well as the systematic, parity 0, and parity 1 data via the 128-bit DMA VBUS interface. The interleaver table can also optionally be loaded (instead of TCP3D generating it internally). All of the input configuration registers corresponding to the process being programmed must be written, even if their value is not changing.
4. The TCP3D is triggered by the EDMA3 performing a write to the trigger register(s) (or TCP3D automatically triggers once all input configuration registers and data are written if in auto-trigger mode) and TCP3D decodes the code block.
5. TCP3D issues a receive event, REVT0/1 to EDMA3 that causes it to transfer the decoded block (and optionally soft bits and/or output parameters) out of the TCP3D.
6. EDMA3 generates an interrupt to the DSP to signal that the decoding process is finished. If there were multiple code blocks setup to be processed, then the EDMA3 loads the next one and does not send an interrupt to the DSP until the last code block is processed.

The following sections describe each step in more detail.

## 3.2 Initialization

Initialization includes resetting the TCP3D and programming its control registers (TCP3D_MODE, TCP3D_END, TCP3D_EMU, TCP3D_EXE_P0, and potentially TCP3D_EXE_P1) found in Section 4.3. This step must be performed before any code blocks are decoded and is generally repeated only if the mode is changed.

### 3.2.1 Reset

The TCP3D can be reset by either the external reset input or by setting the SOFT_RESET bit in the TCP3D_SOFT_RESET register. Portions of the TCP3D logic can also be reset by entering a TCP3D CLEAR command in the TCP3D_EXE_P0 or TCP3D_EXE_P1 registers. All of these will be described in the following sections.

#### 3.2.1.1 External Reset

The external reset input to the TCP3D resets all of the flip-flops inside the TCP3D. The contents of the embedded RAMs are not affected by the external reset.

#### 3.2.1.2 Soft Reset

Writing a 1 to the SOFT_RESET bit in the TCP3D Soft Reset register generates an internal reset inside the TCP3D that holds all of the internal flip-flops in reset with the exception of those used for the VBUS_CFG interface access, VBUS_DMA interface access, and the clock management interface. This allows the TCP3D to acknowledge all VBUS accesses while in soft reset, thereby preventing any bus timeouts. All memory mapped registers in both the VBUS_CFG and VBUS_DMA with the exception of this one are also reset. The reset will persist until SOFT_RESET is cleared to a 0.

#### 3.2.1.3 TCP3D CLEAR command

Portions of the TCP3D logic can also be reset by issuing a TCP3D CLEAR command to the TCP3D_EXE_P0 or TCP3D_EXE_P1 (used only for split mode) registers. The TCP3D CLEAR command is used to clear out selected flip-flops in the TCP3D in a manner that allows it to resume operation without being re-initialized. This is normally used to recover after an error is detected as described in Section 3.5.4.5. A TCP3D CLEAR command causes all state machines to go to the IDLE state and clears other control logic. The DSP must wait a minimum of 20 TCP3D clocks before issuing an enable command. The TCP3D will remain in IDLE state until another command is issued to the TCP3D_EXE_P0 or TCP3D_EXE_P1 registers. No registers, including those used for interrupts, are affected by a TCP3D CLEAR command, so no new initialization is required to recover from it. The contents of the embedded RAMs and the interrupt logic are not affected by this command.

In all modes but split mode, writing a TCP3D CLEAR command to TCP3D_EXE_P0 clears out the active logic in the TCP3D. In split mode, writing a TCP3D CLEAR command to TCP3D_EXE_P0 clears out only the Process 0 logic, while writing a TCP3D CLEAR command to TCP3D_EXE_P1 clears out only the Process 1 logic.

### 3.2.2 Mode Control

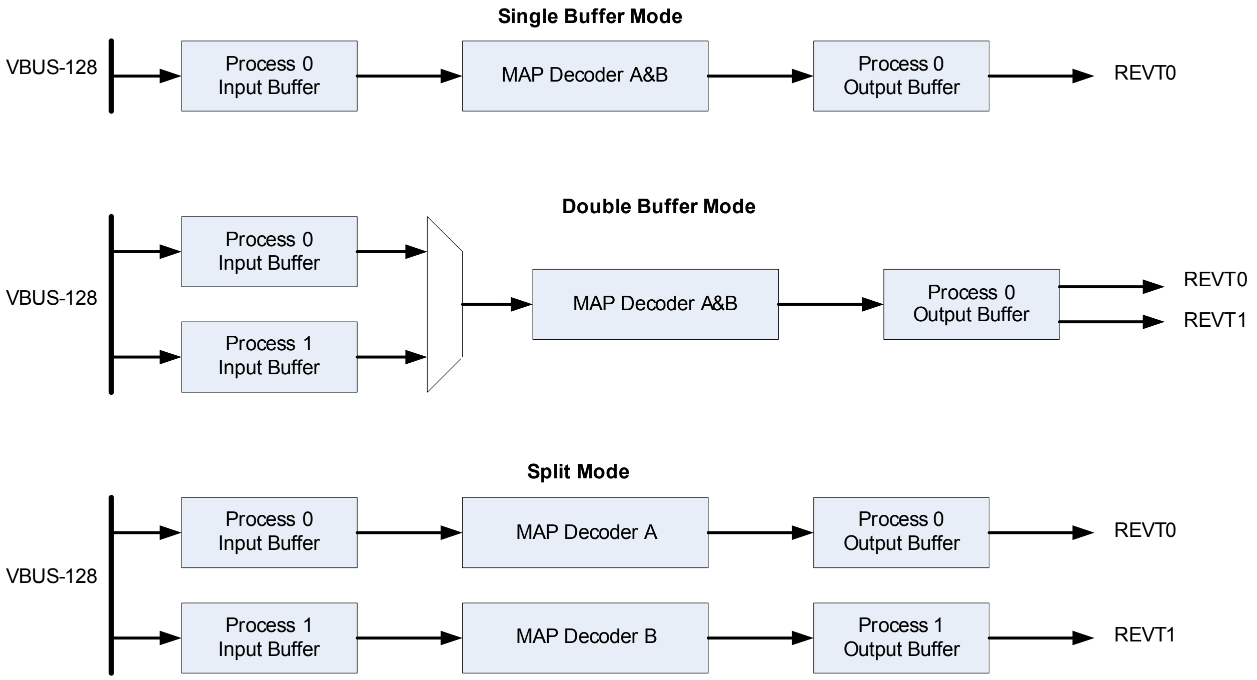
The TCP3D_MODE register controls different aspects of the operating mode of the TCP3D. Each field in the register is described in the following sections.

### 3.2.2.1 MODE_SEL and IN_MEM_DB_EN Fields

After resetting the TCP3D, the data processing mode should be configured via the *MODE_SEL* and *IN_MEM_DB_EN* fields in the TCP3D_MODE register (Section 4.3.2). The TCP3D must be reset (either external reset or soft reset) to make sure the TCP3D *Interface State Machine* is in the IDLE state before changing the *MODE_SEL* field.

The three categories of processing are shown in Figure 3-1 and are described below.

**Figure 3-1 TCP3D Data Processing Configurations**



#### 3.2.2.1.1 Single-Buffered Mode

In this mode, the TCP3D operates as a single turbo-decoder with a single set of input buffers for the code block and a single set of input configuration registers. Single-buffered mode supports LTE, WiMAX and 3GPP code blocks.

#### 3.2.2.1.2 Double-Buffered Mode

Double-buffered mode (i.e. ping/pong mode) provides a very efficient system interface to the TCP3D. In this mode, the TCP3D operates as a single turbo-decoder with 2 complete sets of input configuration registers and input data buffers. The system can utilize this mode by loading the next code block to be decoded while the TCP3D is decoding the current code block. This approach can result in a reduction in the data transfer latency for the next code block. The two sets of resources are referred to as **Process 0** and **Process 1**. Double-buffered mode supports LTE, WiMAX and 3GPP code blocks.

#### 3.2.2.1.3 Split Mode

Split mode configures the TCP3D resources into two independent turbo decoder coprocessors that share a common VBUS interface. In this mode, the TCP3D can decode two code blocks in parallel. The two coprocessors are referred to as **Process 0** and **Process 1**. Split mode supports only 3GPP code blocks.

Table 3-1 summarizes the possible processing mode combinations as a function of the *MODE_SEL* and *IN_MEM_DB_EN* fields.

**Table 3-1 TCP3D Mode Combinations**

| MODE_SEL | IN_MEM_DB_EN | PROC_0 | PROC_1 | MODE |
|---|---|---|---|---|
| 0 | 0 | Y | N | WCDMA; Single Buffered |
| 0 | 1 | Y | Y | WCDMA; Double Buffered |
| 1 | 0 | Y | N | LTE; Single Buffered |
| 1 | 1 | Y | Y | LTE; Double Buffered |
| 2 | 0 | Y | N | WiMAX; Single Buffered |
| 2 | 1 | Y | Y | WiMAX; Double Buffered |
| 3 | 0 (only) | Y | Y | WCDMA (HSUPA+); Single Buffered; Split |

**End of Table 3-1**

#### 3.2.2.2 ITG_EN Field

In order to offload the DSP from the task of generating the interleaver indices, the TCP3D provides separate interleaver table generators for each of the 3 supported standards (3GPP, LTE, WiMAX). When a new interleaver table needs to be generated (INTER_LOAD_SEL bit in TCP3D_IC_CFG2_P0 is set to 1), the ITG_EN field in the TCP3D_MODE register determines whether the interleaver table is generated internally by the TCP3D or externally by the DSP. If generated internally, the MODE_SEL field in TCP3D_MODE determines which of the three interleaver table generators is enabled.

#### 3.2.2.3 ERR_IGNORE_EN Field

The *ERR_IGNORE_EN* field in the TCP3D_MODE register determines whether the TCP3D stops after detecting an enabled error condition. See Section 3.5.1 for more details.

#### 3.2.2.4 AUTO_TRIG_EN Field

The *AUTO_TRIG_EN* field in the TCP3D_MODE register determines whether the TCP3D decoding is triggered manually by writing to the trigger registers or automatically after all the input config registers and input data have been written. See Section 3.4.2 for more details.

#### 3.2.2.5 LTE_CRC_ISEL Field

The *LTE_CRC_ISEL* field in the TCP3D_MODE register controls the initial value used for the CRC calculation in LTE mode for the LTE CRC stopping criterion. It was added for additional flexibility in the future, but should currently always be set to 0. See Section 3.5.2.3 for more details.

### 3.2.3 Big-Endian Control

The TCP3D_END register controls the format conversion of the input interleaver indices and input data (systematic and parity) when the system is in big-endian mode. It has no effect in little-endian mode. The ENDIAN_INTR field controls the conversion of input interleaver indices and is described in Section 3.3.9.4. The ENDIAN_INDATA field controls the conversion of the input data and is described in Section 3.3.9.5.

### 3.2.4 Emulation Mode Control

The TCP3D_EMU register controls the behavior of the TCP3D during an emulation halt. Each field in the register is described in the following sections. For more information on emulation mode usage see Section 3.6.2.

**Note—**When halted in emulation mode, the TCP3D will still respond to accesses on both of its VBUS interfaces.

#### 3.2.4.1 FREERUN Field

The *FREERUN* field in the TCP3D_EMU register determines if an emulation halt of the DSP will also halt the TCP3D. If *FREERUN* is set to 1, and an emulation halt of the DSP occurs, the TCP3D will continue processing normally. If *FREERUN* is set to 0, and an emulation halt of the DSP occurs, then the TCP3D will be halted in a manner determined by the setting of the SOFT bit. Note that when the *FREERUN* is set to 1, the *SOFT* field has no effect.

#### 3.2.4.2 SOFT Field

The *SOFT* field in the TCP3D_EMU register determines how the TCP3D stops during an emulation halt. If *SOFT* is set to 0, the TCP3D performs a *hard* stop. In this case, the TCP3D is required to halt as soon as it can. As such, there is no guarantee that the TCP3D will complete any decodes in progress before halting. See Section 3.6.2.2 for more detail about the internal operation of the TCP3D during a hard stop.

If *SOFT* is set to 1, the TCP3D performs a soft stop. In this case, the TCP3D is allowed to halt gracefully. Any decodes in progress are allowed to complete and the results are transferred out via DMA. See Section 3.6.2.2 for more detail about the internal operation of the TCP3D during a soft stop.

#### 3.2.4.3 RT_SEL Field

If the TCP3D will not be needed to temporarily exit the emulation suspend state while the system processes any *real-time* interrupts, the *RT_SEL* field should be set to its default value of 0. The following text provides more details on the effect of this setting.

There is an emulation mode option that allows *real-time* interrupts to be processed while in emulation suspend. These are interrupts related to time-sensitive processing (such as streaming audio, etc.) that must be performed even when in emulation suspend. Selected IPs that are required to process these real-time interrupts are temporarily released from emulation suspend while the system processes the interrupts. The *RT_SEL* field in the TCP3D_EMU register determines whether the TCP3D is set up to assist in the real-time interrupt processing during an emulation suspend. This is accomplished by selecting which hardware signal the TCP3D uses to detect emulation halt: 0 = use TCP3D_emu_dbgsusp signal and 1 = use TCP3D_emu_dbgsusp_rt signal.

The first signal is used for normal emulation usage, meaning that when the *FREERUN* field is set to 0 the TCP3D will halt during emulation suspend and will not be temporarily removed from the emulation suspend state in response to any real-time interrupts. The second signal is used for real-time interrupt processing, meaning that when the *FREERUN* field is set to 0 the TCP3D will halt during emulation suspend until there is a real-time interrupt. During the real-time interrupt service routine the TCP3D will operate even though the system is in emulation suspend. The RT_SEL field may be changed dynamically so that it only operates during real-time interrupts that require the TCP3D.

### 3.2.5 Execution Control

The *EXE_CMD* field in the TCP3D_EXE_P0/P1 registers controls the operation of the *TCP3D Interface State Machine* as described in Section 3.5.1. For normal operation, the EXE_CMD field should be set to 1 to enable the TCP3D. Separate registers are provided for each process to support split mode. The TCP3D_EXE_P1 register is only used in split mode to control second process. If the TCP3D is not in split mode, TCP3D_EXE_P1 has no effect on the operation of the TCP3D.

**Note—**The TCP3D_EXE_P0/P1 registers should be programmed after the other initialization registers (TCP3D_MODE, TCP3D_END, and TCP3D_EMU) for proper operation.

## 3.3 Input Data/Parameter Preparation

This section describes the expected format and packing of the input data as well as how to calculate several of the input parameters. All of the input configuration registers corresponding to the process being programmed must be written, even if their value is not changing.

### 3.3.1 Input Data / Interleaver Table Length Extension for 3GPP

In order to take advantage of the parallel processing in the TCP3D in 3GPP mode, the number of input LLRs and input interleaver indices must be extended if the code block size is not a multiple of 4. This section summarizes the implications of this extension and gives some examples.

Table 3-2 shows the examples for a set of code block lengths that range from K+3 being a multiple of 4 to K+0 being a multiple of 4 (K ranges from 41 to 44 in the example). The Input LLR RAMs (i.e. Systematic, Parity0, and Parity1) are loaded with an extended number of entries. The length is extended by twice the number of bits that must be added to make the code block size a multiple of 4. The interleaver table RAM is also loaded (when generated externally) with an extended number of entries, but the length is extended only by the number of bits that must be added to make the code block size a multiple of 4.

**Table 3-2 Example of 3GPP Extended Block Length Configurations**

| Block Length (K) | Value Programmed In BLK_LN[12:0] | Input LLR RAM Entries | Interleaver Table Entries | Hard/Soft Decisions Generated | Hard/Soft Decisions DMAed |
|---|---|---|---|---|---|
| 41 | 40 | 47 | 44 | 44 | 41 |
| 42 | 41 | 46 | 44 | 44 | 42 |
| 43 | 42 | 45 | 44 | 44 | 43 |
| 44 | 43 | 44 | 44 | 44 | 44 |
| **End of Table 3-2** | | | | | |

The number of hard/soft decisions generated internally is equal to number of interleaver table entries, but only those hard/soft decisions associated with the original, un-extended, code block length are actually transferred out of the TCP3D. The details of the data placed in the extended input LLR RAM entries and interleaver table entries are given in Section 3.3.4 and Section 3.3.6.2.2, respectively.

### 3.3.2 Input Data Formatting

In the DSP, after scaling and depuncturing, the channel LLRs (soft bits) are stored as one LLR value per byte. However, only the lowest 6 bits of the byte are stored to the TCP3D memory. The two most significant bits of the byte are ignored by TCP3D. So, the range of the values is from -32 to 31 in Q0 format. It is left to the user to choose the Q format for the LLRs. Figure 3-2 shows the range of the LLR values for various Q formats and how these values are interpreted.

It is the user's responsibility to saturate the LLR values. Also, note that the threshold and the constant values of the max star function need to be in the same Q format as the LLR values.

**Figure 3-2 Input LLR Formats**

| Input LLR interpretation | Integer range (Max-Log-MAP) | | Fractional range (Max-star) | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | Q1 | | Q2 | | Q3 | |
| | Binary | Decimal | Binary | Decimal | Binary | Decimal | Binary | Decimal |
| Most reliable "1" | 011111 | 31 | 01111.1 | 15.5 | 0111.11 | 7.75 | 011.111 | 3.825 |
| | 011110 | 30 | 01111.0 | 15.0 | 0111.10 | 7.50 | 011.110 | 3.750 |
| ⋮ | ⋮ | | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| Least reliable "1" | 000001 | 1 | 00000.1 | 0.5 | 0000.01 | 0.25 | 000.001 | 0.125 |
| Neutral | 000000 | 0 | 00000.0 | 0.0 | 0000.00 | 0.00 | 000.000 | 0.000 |
| Least reliable "0" | 111111 | -1 | 11111.1 | -0.5 | 1111.11 | -0.25 | 111.111 | -0.125 |
| ⋮ | ⋮ | | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| | 100001 | -31 | 10000.1 | -15.5 | 1000.01 | -7.75 | 100.001 | -3.825 |
| Most reliable "0" | 100000 | -32 | 10000.0 | -16.0 | 1000.00 | -8.00 | 100.000 | -4.000 |

**Note—**The parity soft bits of the second RSC encoder are not de-interleaved as was required for TCP2.

### 3.3.3 Input Data Ordering for WiMAX

This section describes the input data ordering and memory placement for WiMAX.

Given (for k=0,…,K/2-1)

A[k] — LLR for the systematic bit A
B[k] — LLR for the systematic bit B
Y1[k] — LLR for the parity bit Y of the first RSC encoder
W1[k] — LLR for the parity bit W of the first RSC encoder
Y2[k] — LLR for the parity bit Y of the second RSC encoder
W2[k] — LLR for the parity bit W of the second RSC encoder

Sys_a — Base address for first half of Systematic Input Memory
Sys_b — Base address for second half of Systematic Input Memory
Par0_a — Base address for first half of Parity 0 Input Memory
Par0_b — Base address for second half of Parity 0 Input Memory
Par1_a — Base address for first half of Parity 1 Input Memory
Par1_b— Base address for second half of Parity 1 Input Memory

The LLRs are stored in the input memories as shown in Figure 3-3. Note that the input data is broken into two halves and stored in the corresponding half of the input memory. This is done for the dual-MAP decode operation.

**Figure 3-3 Data Ordering for WiMAX**

| Address (bytes) | Systematic Input Mem | Address (bytes) | Parity 0 Input Mem | Address (bytes) | Parity 1 Input Mem |
|---|---|---|---|---|---|
| Sys_a+0 | A[0] | Par0_a+0 | Y1[0] | Par1_a+0 | Y2[0] |
| Sys_a+1 | B[0] | Par0_a+1 | W1[0] | Par1_a+1 | W2[0] |
| Sys_a+2 | A[1] | Par0_a+2 | Y1[1] | Par1_a+2 | Y2[1] |
| Sys_a+3 | B[1] | Par0_a+3 | W1[1] | Par1_a+3 | W2[1] |
| . | . | . | . | . | . |
| . | . | . | . | . | . |
| Sys_a+K/2-2 | A[K/4-1] | Par0_a+K/2-2 | Y1[K/4-1] | Par1_a+K/2-2 | Y2[K/4-1] |
| Sys_a+K/2-1 | B[K/4-1] | Par0_a+K/2-1 | W1[K/4-1] | Par1_a+K/2-1 | W2[K/4-1] |
| Sys_b+0 | A[K/4] | Par0_b+0 | Y1[K/4] | Par1_b+0 | Y2[K/4] |
| Sys_b+1 | B[K/4] | Par0_b+1 | W1[K/4] | Par1_b+1 | W2[K/4] |
| Sys_b+2 | A[K/4+1] | Par0_b+2 | Y1[K/4+1] | Par1_b+2 | Y2[K/4+1] |
| Sys_b+3 | B[K/4+1] | Par0_b+3 | W1[K/4+1] | Par1_b+3 | W2[K/4+1] |
| . | . | . | . | . | . |
| . | . | . | . | . | . |
| Sys_b+K/2-2 | A[K/2-1] | Par0_b+K/2-2 | Y1[K/2-1] | Par1_b+K/2-2 | Y2[K/2-1] |
| Sys_b+K/2-1 | B[K/2-1] | Par0_b+K/2-1 | W1[K/2-1] | Par1_b+K/2-1 | W2[K/2-1] |

### 3.3.4 Input Data Ordering for LTE

This section describes the input data ordering and memory placement for LTE.

Given (for k=0,…,K-1)

X[k] - LLR for the systematic bit

P1[k]- LLR for the parity bit of the first RSC encoder

P2[k]- LLR for the parity bit of the second RSC encoder

Sys_a - Base address for first half of Systematic Input Memory

Sys_b - Base address for second half of Systematic Input Memory

Par0_a - Base address for first half of Parity 0 Input Memory

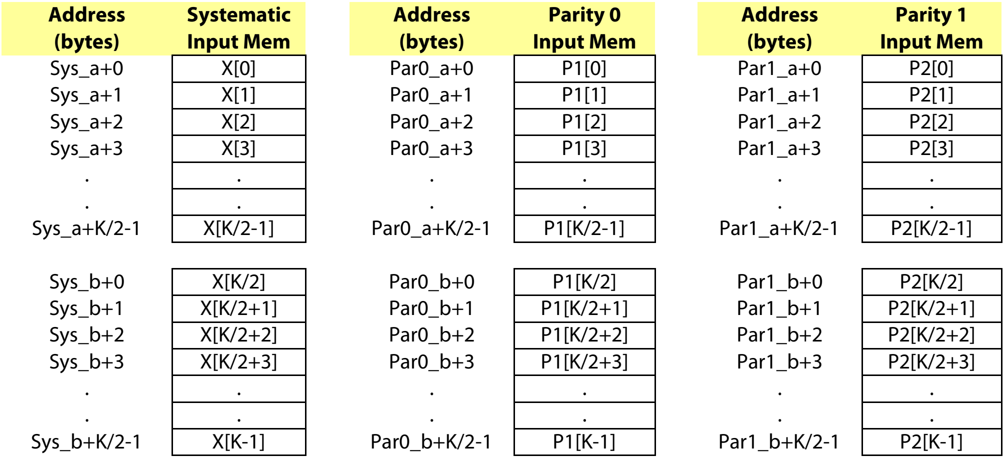
Par0_b - Base address for second half of Parity 0 Input Memory

Par1_a - Base address for first half of Parity 1 Input Memory

Par1_b - Base address for second half of Parity 1 Input Memory

The LLRs are stored in the input memories as shown in Figure 3-4.

**Figure 3-4 Data Ordering for LTE**



| Address (bytes) | Systematic Input Mem | Address (bytes) | Parity 0 Input Mem | Address (bytes) | Parity 1 Input Mem |
|---|---|---|---|---|---|
| Sys_a+0 | X[0] | Par0_a+0 | P1[0] | Par1_a+0 | P2[0] |
| Sys_a+1 | X[1] | Par0_a+1 | P1[1] | Par1_a+1 | P2[1] |
| Sys_a+2 | X[2] | Par0_a+2 | P1[2] | Par1_a+2 | P2[2] |
| Sys_a+3 | X[3] | Par0_a+3 | P1[3] | Par1_a+3 | P2[3] |
| . | . | . | . | . | . |
| . | . | . | . | . | . |
| Sys_a+K/2-1 | X[K/2-1] | Par0_a+K/2-1 | P1[K/2-1] | Par1_a+K/2-1 | P2[K/2-1] |
| Sys_b+0 | X[K/2] | Par0_b+0 | P1[K/2] | Par1_b+0 | P2[K/2] |
| Sys_b+1 | X[K/2+1] | Par0_b+1 | P1[K/2+1] | Par1_b+1 | P2[K/2+1] |
| Sys_b+2 | X[K/2+2] | Par0_b+2 | P1[K/2+2] | Par1_b+2 | P2[K/2+2] |
| Sys_b+3 | X[K/2+3] | Par0_b+3 | P1[K/2+3] | Par1_b+3 | P2[K/2+3] |
| . | . | . | . | . | . |
| . | . | . | . | . | . |
| Sys_b+K/2-1 | X[K-1] | Par0_b+K/2-1 | P1[K-1] | Par1_b+K/2-1 | P2[K-1] |

### 3.3.5 Input Data Ordering for 3GPP

This section describes the input data ordering and memory placement for 3GPP. Note that if the block size, K, is not a multiple of 4, some or all of the trellis termination (tail) bits are placed at the end of the normal systematic and parity input data.

#### 3.3.5.1 3GPP Rel6 When K is a Multiple of 4

Given (for k=0,…,K-1)

X[k] - LLR for the systematic bit

P1[k]- LLR for the parity bit of the first RSC encoder

P2[k]- LLR for the parity bit of the second RSC encoder

and given (for k=0,…,2)

Xt1[k]- LLR for the trellis termination systematic bit of the first RSC encoder

Xt2[k]- LLR for the trellis termination systematic bit of the second RSC encoder

Pt1[k]- LLR for the trellis termination parity bit of the first RSC encoder

Pt2[k]- LLR for the trellis termination parity bit of the second RSC encoder

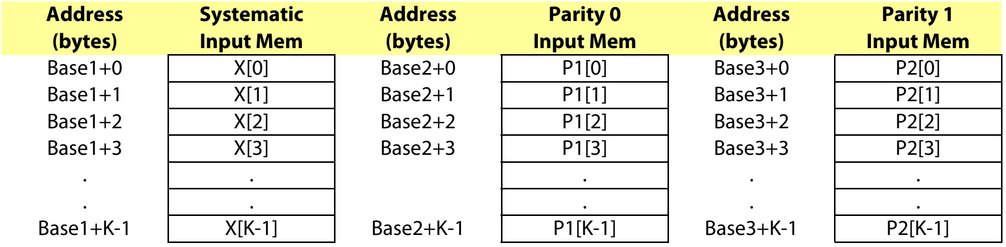
Base1 - Base address for Systematic Input Memory

Base2 - Base address for Parity 0 Input Memory

Base3 - Base address for Parity 1 Input Memory

The LLRs are stored in the input memories as shown in Figure 3-5.

**Figure 3-5** **Data Ordering for 3GPP Rel6 When K is a Multiple of 4**

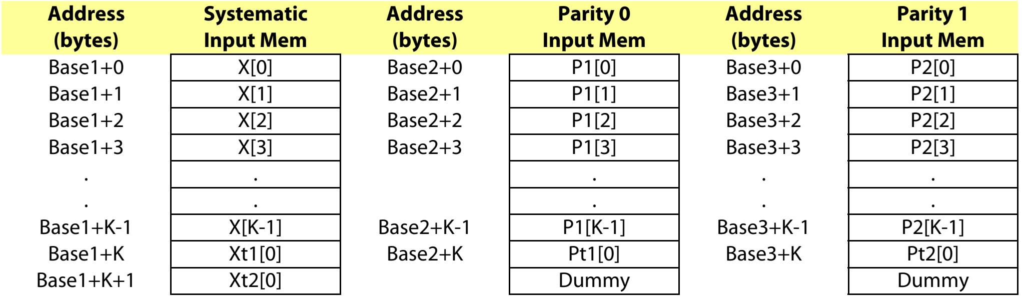


| Address (bytes) | Systematic Input Mem | Address (bytes) | Parity 0 Input Mem | Address (bytes) | Parity 1 Input Mem |
|---|---|---|---|---|---|
| Base1+0 | X[0] | Base2+0 | P1[0] | Base3+0 | P2[0] |
| Base1+1 | X[1] | Base2+1 | P1[1] | Base3+1 | P2[1] |
| Base1+2 | X[2] | Base2+2 | P1[2] | Base3+2 | P2[2] |
| Base1+3 | X[3] | Base2+3 | P1[3] | Base3+3 | P2[3] |
| . | . |  | . | . | . |
| . | . |  | . | . | . |
| Base1+K-1 | X[K-1] | Base2+K-1 | P1[K-1] | Base3+K-1 | P2[K-1] |

#### 3.3.5.2 3GPP Rel6 When (K+1) is a Multiple of 4

The LLRs are stored in the input memories as shown in Figure 3-6.

**Figure 3-6** **Data Ordering for 3GPP When K+1 is a Multiple of 4**



| Address (bytes) | Systematic Input Mem | Address (bytes) | Parity 0 Input Mem | Address (bytes) | Parity 1 Input Mem |
|---|---|---|---|---|---|
| Base1+0 | X[0] | Base2+0 | P1[0] | Base3+0 | P2[0] |
| Base1+1 | X[1] | Base2+1 | P1[1] | Base3+1 | P2[1] |
| Base1+2 | X[2] | Base2+2 | P1[2] | Base3+2 | P2[2] |
| Base1+3 | X[3] | Base2+3 | P1[3] | Base3+3 | P2[3] |
| . | . |  | . | . | . |
| . | . |  | . | . | . |
| Base1+K-1 | X[K-1] | Base2+K-1 | P1[K-1] | Base3+K-1 | P2[K-1] |
| Base1+K | Xt1[0] | Base2+K | Pt1[0] | Base3+K | Pt2[0] |
| Base1+K+1 | Xt2[0] |  | Dummy |  | Dummy |

**Note—**Dummy bits are included in the EDMA transfer, but they are not processed in TCP3D.

#### 3.3.5.3 3GPP Rel6 When (K+2) is a Multiple of 4

The LLRs are stored in the input memories as shown in Figure 3-7.

**Figure 3-7 Data Ordering for 3GPP When K+2 is a Multiple of 4**

| Address (bytes) | Systematic Input Mem | Address (bytes) | Parity 0 Input Mem | Address (bytes) | Parity 1 Input Mem |
|---|---|---|---|---|---|
| Base1+0 | X[0] | Base2+0 | P1[0] | Base3+0 | P2[0] |
| Base1+1 | X[1] | Base2+1 | P1[1] | Base3+1 | P2[1] |
| Base1+2 | X[2] | Base2+2 | P1[2] | Base3+2 | P2[2] |
| Base1+3 | X[3] | Base2+3 | P1[3] | Base3+3 | P2[3] |
| . | . |  | . | . | . |
| . | . |  | . | . | . |
| Base1+K-1 | X[K-1] | Base2+K-1 | P1[K-1] | Base3+K-1 | P2[K-1] |
| Base1+K | Xt1[0] | Base2+K | Pt1[0] | Base3+K | Pt2[0] |
| Base1+K+1 | Xt1[1] | Base2+K+1 | Pt1[1] | Base3+K+1 | Pt2[1] |
| Base1+K+2 | Xt2[0] |  | Dummy |  | Dummy |
| Base1+K+3 | Xt2[1] |  | Dummy |  | Dummy |

#### 3.3.5.4 3GPP Rel6 When (K+3) is a Multiple of 4

The LLRs are stored in the input memories as shown in Figure 3-8.

**Figure 3-8 Data Ordering for 3GPP When K+3 is a Multiple of 4**

| Address (bytes) | Systematic Input Mem | Address (bytes) | Parity 0 Input Mem | Address (bytes) | Parity 1 Input Mem |
|---|---|---|---|---|---|
| Base1+0 | X[0] | Base2+0 | P1[0] | Base3+0 | P2[0] |
| Base1+1 | X[1] | Base2+1 | P1[1] | Base3+1 | P2[1] |
| Base1+2 | X[2] | Base2+2 | P1[2] | Base3+2 | P2[2] |
| Base1+3 | X[3] | Base2+3 | P1[3] | Base3+3 | P2[3] |
| . | . |  | . | . | . |
| . | . |  | . | . | . |
| Base1+K-1 | X[K-1] | Base2+K-1 | P1[K-1] | Base3+K-1 | P2[K-1] |
| Base1+K | Xt1[0] | Base2+K | Pt1[0] | Base3+K | Pt2[0] |
| Base1+K+1 | Xt1[1] | Base2+K+1 | Pt1[1] | Base3+K+1 | Pt2[1] |
| Base1+K+2 | Xt1[2] | Base2+K+2 | Pt1[2] | Base3+K+2 | Pt2[2] |
| Base1+K+3 | Xt2[0] |  | Dummy |  | Dummy |
| Base1+K+4 | Xt2[1] |  | Dummy |  | Dummy |
| Base1+K+5 | Xt2[2] |  | Dummy |  | Dummy |

### 3.3.6 Interleaver Table Generation

The interleaver table may be generated internally by the TCP3D or externally by the DSP. This section describes the usage of both options in more detail.

#### 3.3.6.1 Internal Interleaver Table Generation Parameters

Using the internal interleaver table generation in LTE and WiMAX modes requires the calculation of several input parameters. These parameters are placed in the input configuration registers **TCP3D_IC_CFG12_P0/P1**, **TCP3D_IC_CFG13_P0/P1**, and **TCP3D_IC_CFG14_P0/P1** as shown in the register descriptions. The following subsections show how to calculate these input parameters.

#### 3.3.6.1.1 LTE Interleaver Table Generation Parameters

Table 3-3 shows the input parameter definitions for LTE.

**Table 3-3 Input Parameters Required for LTE Interleaver Table Generator**

| Register | Field | Value |
|---|---|---|
| TCP_IC_CFG12_P0/P1 | ITG_PARAM0 | $(2 \times f2) \bmod K$ |
| TCP_IC_CFG12_P0/P1 | ITG_PARAM1 | $g(0)$ |
| TCP_IC_CFG13_P0/P1 | ITG_PARAM2 | $\pi(K/4)$ |
| TCP_IC_CFG13_P0/P1 | ITG_PARAM3 | $\pi(K/2)$ |
| TCP_IC_CFG14_P0/P1 | ITG_PARAM4 | $\pi(3K/4)$ |
| **End of Table 3-3** | | |

where:

- $\pi$: interleaver index
- $K$: code block size
- $f1$, $f2$: constants from LTE spec that are determined by code block size
- *mod:* modulo operator (finds the remainder of division of one number by another)

The equation to calculate $g(0)$ is shown here:
$g(0) = [f1 + f2] \bmod K$

The equation used to generate the interleaver indices $\pi(i)$ is shown here:
$p(i) = (f1 \times i + f2 \times i2) \bmod K$

#### 3.3.6.1.2 WiMAX Interleaver Table Generation Parameters

Table 3-4 shows the input parameter definitions for WiMAX.

**Table 3-4 Input Parameters Required for WiMAX Interleaver Table Generator**

| Register | Field | Value |
|---|---|---|
| TCP_IC_CFG12_P0/P1 | ITG_PARAM0 | *0 (not used)* |
| TCP_IC_CFG12_P0/P1 | ITG_PARAM1 | $P_{inc}$ |
| TCP_IC_CFG13_P0/P1 | ITG_PARAM2 | $\pi_1 (0)$ |
| TCP_IC_CFG13_P0/P1 | ITG_PARAM3 | $\pi_2 (0)$ |
| TCP_IC_CFG14_P0/P1 | ITG_PARAM4 | $\pi 3 (0)$ |
| **End of Table 3-4** | | |

The equations used to calculate the input parameters are shown here:

$$P_{inc} = (4 \times P_0) \bmod (K/2)$$
$$\pi_1(0) = (P_0 + 1 + K/4 + P_1) \bmod (K/2)$$
$$\pi_2(0) = (2 \times P_0 + 1 + P_2) \bmod (K/2)$$
$$\pi_3(0) = (3 \times P_0 + 1 + K/4 + P_3) \bmod (K/2)$$

where:

- $\pi$: interleaver index
- $K$: code block size
- $P_x$: constants from WiMAX spec that are determined by code block size

### 3.3.6.2 External Interleaver Table Generation

The TCP3D can generate the interleaver table internally in all modes (3GPP, LTE, and WiMAX), but the interleaver table can also be computed in the DSP and then loaded into the TCP3D. In the DSP, the table is stored in a 16-bit array. Several special rules must be applied to the externally generated interleaver table to work properly with the TCP3D.

#### 3.3.6.2.1 LTE and WiMAX Interleaver Rules

When the TCP3D operates with two MAP decoders in parallel, i.e. in LTE or WiMAX mode, the two halves of the interleaver table are stored in two separate memory banks in the TCP3D. In this case the original interleaver values, for LTE, (bit interleaver): y[k], k=0, K-1, and for WiMAX, (symbol interleaver): y[k], k=0, K/2-1, have to be modified in the DSP according to the following rules:

In LTE mode:

```
if $(y[k] \geq K/2)$ then $y[k] = ( (y[k] - K/2)$   OR 0x1000)
```

In WiMAX mode:

```
if $(y[k] \geq K/4)$  then
    $y[k] = ( 2 \times (y[k] - K/4)$   OR 0x1000 )
else
    $y[k] = 2 \times y[k]$
end if
```

**Note—**In WiMAX mode, the factor of 2 is needed because the TCP3 interleaver indices do not address symbols but the first bit of the symbols.

#### 3.3.6.2.2 3GPP Interleaver Rules

When the code block length K is not multiple of 4, the interleaver table length must be extended to be a multiple of 4 as $K_{EXT}$ = 4ceil(K/4). The first K elements of the interleaver array contain original interleaver values Y[k] for k=0,…,K-1. The extended elements, at positions from K to $K_{EXT}$-1, contain the values according to Table 3-5.

**Table 3-5 Extended Interleaver for 3GPP When K is Not Multiple of 4**

| K+1 multiple of 4 | | K+2 multiple of 4 | | K+3 multiple of 4 | |
|---|---|---|---|---|---|
| **Address (bytes)** | **Interleaver Array** | **Address (bytes)** | **Interleaver Array** | **Address (bytes)** | **Interleaver Array** |
| Base+0 | Y[0] | Base+0 | Y[0] | Base+0 | Y[0] |
| • | • | | • | • | • |
| • | • | | • | • | • |
| • | • | | • | • | • |
| Base+K-1 | Y[K-1] | Base+K-1 | Y[K-1] | Base+K-1 | Y[K-1] |
| Base+K | K+1 | Base+K | K+2 | Base+K | K+3 |
| | | Base+K+1 | K+3 | Base+K+1 | K+4 |
| | | | | Base+K+2 | K+5 |
| **End of Table 3-5** | | | | | |

## 3.3.7 Sliding Window Parameter Calculations

Sliding windows are used to limit the amount of scratch memory needed to decode each section of the code block. The code block is split into 4 (WiMAX or LTE) or 2 (3GPP) sub-blocks of equal lengths. Each sub-block is further split into smaller segments, (referred as sliding windows, SW), of the same or different lengths. The largest sliding window length is 128 information bits. The sliding windows are divided into 3 categories, labeled SW0, SW1, and SW2. There can be a multiple number of SW0s but only one SW1 and one SW2. The length of SW0, expressed in number of information bits, can have one of the values from the set {0, 16, 32, 48, 64, 96 and 128}. The number of SW0s and SW lengths required for the input configuration registers are determined according to the following procedure. The tradeoffs in the selection of the nominal size of SW0 (denoted sw0NomLen in the procedure) are given in section Section 3.5.5

```
Given

    blockLen  - code block length expressed in number of info bits
    numMap    - number of MAP decoders (1 or 2) (numMap = 2 for LTE and WiMAX,
    numMap = 1 for 3GPP mode)

    sw0NomLen - Nominal length of sw0 (16, 32, 48, 64, 96 or 128)

Any of the possible values for sw0NomLen may be chosen as long as the condition
(blockLen ≤ numMap*128*sw0NomLen) holds.  The choice will affect the decoding
throughput and to some degree the BER performance.  The tradeoffs in the selection
are discussed in section Section 3.5.5.

Procedure:
    Radix = 4;
    Step     = log2(Radix);
    %Make block length divisible by Step*numMap*2: (4 for 3GPP, 8 for LTE/WiMAX)
    blockLenExt = (Step*numMap*2) * ceil(blockLen /(Step*numMap*2));

    %Sub-block length: (2 sub-blocks for 3GPP, 4 sub-blocks for LTE/WiMAX)
    subBlockLen = blockLenExt / (numMap * 2);

    %Total number of sliding windows in a sub-block:
    numSlidingWin = ceil(subBlockLen / sw0NomLen);

    %Calculate lengths of SW0, SW1, and SW2 and number of SW0:
```

```
if(numSlidingWin == 1)
    sw0Len = 0;
    sw1Len = subBlockLen;
    sw2Len = 0;
    num_sw0 = 0;

elseif(numSlidingWin == 2)
    sw0Len = 0;
    sw1Len = Step * ceil(subBlockLen/(Step*2));
    sw2Len = Step * floor(subBlockLen/(Step*2));
    num_sw0 = 0;

elseif (subBlockLen % sw0NomLen) <= (sw0NomLen / 2)
    sw0Len = sw0NomLen;
    sw1Len = Step * ceil((subBlockLen - (numSlidingWin-2)*sw0NomLen)/(Step*2));
    sw2Len = Step * floor((subBlockLen -(numSlidingWin-2)*sw0NomLen)/(Step*2));
    num_sw0 = numSlidingWin-2;

else
    sw0Len = sw0NomLen;
    sw1Len = subBlockLen % sw0NomLen;
    sw2Len = 0;
    num_sw0 = numSlidingWin-1;

end
```

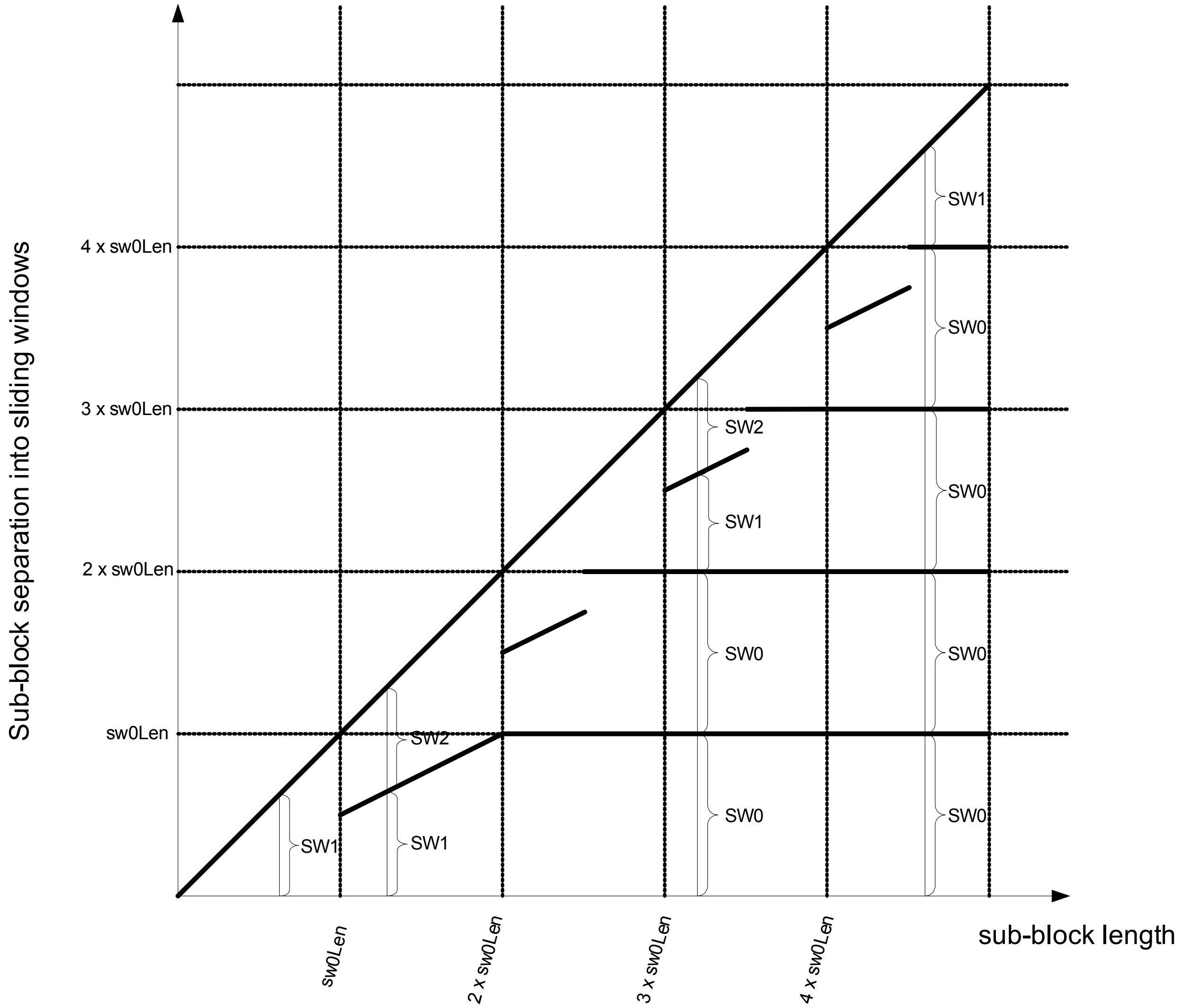
**Note—**Unpredictable operation will result if the procedure above is not followed and the following rules are broken:

- If (num_sw0 > 0) then SW0 length ≥ SW1 length
- SW1 length ≥ 10
- If (num_sw0 > 0) and (SW1 length is not equal to SW2 length) and (SW0 length - SW1 length < 4) then SW2 length must be 0 (off)
- Block length ≤ Number of MAP decoders × 128 × SW0 length

Where: the Number of MAP decoders is 2 in LTE or WiMAX mode, and 1 in 3GPP mode.

The partition of sub-block into sliding windows SW0, SW1, and SW2 is shown in Figure 3-9.

**Figure 3-9 Sub-Block Partition Into Sliding Windows**



### 3.3.8 Initial Beta States Computation

In 3GPP (including HSUPA+) and LTE modes, the trellis termination (tail) bits are used to bring the encoder to zero state. Two sets of the initial beta states are computed in the DSP and placed in the TCP3D input configuration registers, one for the first MAP decoder (non-interleaved pass) and the other for the second MAP decoder (interleaved pass). They are computed based on the trellis termination soft bits. This initial beta state computation is not required for WiMAX mode. The number of termination trellis stages used to compute initial beta states is equal to $K_T = 3 - (K_{EXT}-K)$ (where $K_{EXT}$ is defined in Section 3.3.6.2.2).

Given (for k=0,1,2):

Xt1[k] — LLR for the trellis termination systematic bit of the first RSC encoder

Xt2[k] — LLR for the trellis termination systematic bit of the second RSC encoder

Pt1[k] — LLR for the trellis termination parity bit of the first RSC encoder

Pt2[k] — LLR for the trellis termination parity bit of the second RSC encoder

Computation of initial beta metrics is done in two steps:

1. Appropriate formulas are used to compute beta metrics with the precision greater than 8 bits (e.g. 16-bit precision)
2. Computed values are normalized (maximum beta value is subtracted and 127 added to all computed beta metrics) and lower 8 bits are provided to TCP3D

Initial beta values computation method is described for all four cases of interest: $K_T$ =3, $K_T$ =2, $K_T$ =1 and $K_T$ =0. In the last case ($K_T$ =0) there is no need for the 2-step computation.

**$K_T$ = 3:**

First MAP decoder:

Step 1:

$$\begin{array}{llllllll}
\beta_1[0] & = & 0 & & & & & \\
\beta_1[1] & = & Xt1[0] & + Pt1[0] & & & & \\
\beta_1[2] & = & Xt1[0] & & + Xt1[1] & + Pt1[1] & & \\
\beta_1[3] & = & & Pt1[0] & + Xt1[1] & + Pt1[1] & & \\
\beta_1[4] & = & & Pt1[0] & + Xt1[1] & & + Xt1[2] & + Pt1[2] \\
\beta_1[5] & = & Xt1[0] & & + Xt1[1] & & + Xt1[2] & + Pt1[2] \\
\beta_1[6] & = & Xt1[0] & + Pt1[0] & & + Pt1[1] & + Xt1[2] & + Pt1[2] \\
\beta_1[7] & = & & & & Pt1[1] & + Xt1[2] & + Pt1[2]
\end{array}$$

$$\beta_1,max = max(\beta1[0], \beta_1[1], \beta_1[2], \beta_1[3], \beta_1[4], \beta_1[5], \beta_1[6], \beta_1[7])$$

Step 2:

$$\begin{aligned}
\beta_1[0]_{8\text{-bit}} &= \beta_1[0] - \beta_{1,max} + 127 \\
\beta_1[1]_{8\text{-bit}} &= \beta_1[1] - \beta_{1,max} + 127 \\
\beta_1[2]_{8\text{-bit}} &= \beta_1[2] - \beta_{1,max} + 127 \\
\beta_1[3]_{8\text{-bit}} &= \beta_1[3] - \beta_{1,max} + 127 \\
\beta_1[4]_{8\text{-bit}} &= \beta_1[4] - \beta_{1,max} + 127 \\
\beta_1[5]_{8\text{-bit}} &= \beta_1[5] - \beta_{1,max} + 127 \\
\beta_1[6]_{8\text{-bit}} &= \beta_1[6] - \beta_{1,max} + 127 \\
\beta_1[7]_{8\text{-bit}} &= \beta_1[7] - \beta_{1,max} + 127
\end{aligned}$$

Second MAP decoder:

Step 1:

$$\begin{array}{llllll}
\beta_2[0] & = & 0 & & & \\
\beta_2[1] & = & Xt2[0] & + Pt2[0] & & \\
\beta_2[2] & = & Xt2[0] & & + Xt2[1] & + Pt2[1] \\
\beta_2[3] & = & & Pt2[0] & + Xt2[1] & + Pt2[1]
\end{array}$$

$$\begin{array}{lllllll}
\beta_2[4] = & & \text{Pt2}[0] & + \text{Xt2}[1] & & + \text{Xt2}[2] & + \text{Pt2}[2] \\
\beta_2[5] = & \text{Xt2}[0] & & + \text{Xt2}[1] & & + \text{Xt2}[2] & + \text{Pt2}[2] \\
\beta_2[6] = & \text{Xt2}[0] & + \text{Pt2}[0] & & + \text{Pt2}[1] & + \text{Xt2}[2] & + \text{Pt2}[2] \\
\beta_2[7] = & & & & \text{Pt2}[1] & + \text{Xt2}[2] & + \text{Pt2}[2]
\end{array}$$

$$\beta_{2,max} = \max(\beta_2[0],\ \beta_2[1],\ \beta_2[2],\ \beta_2[3],\ \beta_2[4],\ \beta_2[5],\ \beta_2[6],\ \beta_2[7])$$

Step 2:

$$\begin{array}{l}
\beta_2[0]_{8\text{-bit}} = \beta_2[0] - \beta_{2,max} + 127 \\
\beta_2[1]_{8\text{-bit}} = \beta_2[1] - \beta_{2,max} + 127 \\
\beta_2[2]_{8\text{-bit}} = \beta_2[2] - \beta_{2,max} + 127 \\
\beta_2[3]_{8\text{-bit}} = \beta_2[3] - \beta_{2,max} + 127 \\
\beta_2[4]_{8\text{-bit}} = \beta_2[4] - \beta_{2,max} + 127 \\
\beta_2[5]_{8\text{-bit}} = \beta_2[5] - \beta_{2,max} + 127 \\
\beta_2[6]_{8\text{-bit}} = \beta_2[6] - \beta_{2,max} + 127 \\
\beta_2[7]_{8\text{-bit}} = \beta_2[7] - \beta_{2,max} + 127
\end{array}$$

**$K_T$ =2:**

First MAP decoder:

Step 1:

$$\begin{array}{lllll}
\beta_1[0] = 0 & & & & \\
\beta_1[1] = & \text{Xt1}[1] & + \text{Pt1}[1] & & \\
\beta_1[2] = & \text{Xt1}[1] & & + \text{Xt1}[2] & + \text{Pt1}[2] \\
\beta_1[3] = & & \text{Pt1}[1] & + \text{Xt1}[2] & + \text{Pt1}[2]
\end{array}$$

$$\beta_{1,max} = \max(\beta_1[0],\ \beta_1[1],\ \beta_1[2],\ \beta_1[3])$$

Step 2:

$$\begin{array}{l}
\beta_1[0]_{8\text{-bit}} = \beta_1[0] - \beta_{1,max} + 127 \\
\beta_1[1]_{8\text{-bit}} = \beta_1[1] - \beta_{1,max} + 127 \\
\beta_1[2]_{8\text{-bit}} = \beta_1[2] - \beta_{1,max} + 127 \\
\beta_1[3]_{8\text{-bit}} = \beta_1[3] - \beta_{1,max} + 127 \\
\beta_1[4]_{8\text{-bit}} = -128 \\
\beta_1[5]_{8\text{-bit}} = -128 \\
\beta_1[6]_{8\text{-bit}} = -128 \\
\beta_1[7]_{8\text{-bit}} = -128
\end{array}$$

Second MAP decoder:

Step 1:

$$\begin{array}{lllll}
\beta_2[0] = 0 & & & & \\
\beta_2[1] = & \text{Xt2}[1] & + \text{Pt2}[1] & & \\
\beta_2[2] = & \text{Xt2}[1] & & + \text{Xt2}[2] & + \text{Pt2}[2] \\
\beta_2[3] = & & \text{Pt2}[1] & + \text{Xt2}[2] & + \text{Pt2}[2]
\end{array}$$

$$\beta_{2,max} = \max(\beta_2[0],\ \beta_2[1],\ \beta_2[2],\ \beta_2[3])$$

Step 2:

$$\begin{array}{l}
\beta_2[0]_{8\text{-bit}} = \beta_2[0] - \beta_{2,max} + 127 \\
\beta_2[1]_{8\text{-bit}} = \beta_2[1] - \beta_{2,max} + 127 \\
\beta_2[2]_{8\text{-bit}} = \beta_2[2] - \beta_{2,max} + 127 \\
\beta_2[3]_{8\text{-bit}} = \beta_2[3] - \beta_{2,max} + 127 \\
\beta_2[4]_{8\text{-bit}} = -128 \\
\beta_2[5]_{8\text{-bit}} = -128 \\
\beta_2[6]_{8\text{-bit}} = -128 \\
\beta_2[7]_{8\text{-bit}} = -128
\end{array}$$

**$K_T$ =1:**

First MAP decoder:

Step 1:

$$\begin{aligned}
\beta_1[0] &= 0 \\
\beta_1[1] &= \qquad\qquad\qquad\qquad \mathrm{Xt1}[2] + \mathrm{Pt1}[2] \\
\beta_{1,\mathrm{max}} &= \max(\beta_1[0],\ \beta_1[1])
\end{aligned}$$

Step 2:

$$\begin{aligned}
\beta_1[0]_{8\text{-bit}} &= \beta_1[0] - \beta_{1,\mathrm{max}} + 127 \\
\beta_1[1]_{8\text{-bit}} &= \beta_1[1] - \beta_{1,\mathrm{max}} + 127 \\
\beta_1[2]_{8\text{-bit}} &= -128 \\
\beta_1[3]_{8\text{-bit}} &= -128 \\
\beta_1[4]_{8\text{-bit}} &= -128 \\
\beta_1[5]_{8\text{-bit}} &= -128 \\
\beta_1[6]_{8\text{-bit}} &= -128 \\
\beta_1[7]_{8\text{-bit}} &= -128
\end{aligned}$$

Second MAP decoder:

Step 1:

$$\begin{aligned}
\beta_2[0] &= 0 \\
\beta_2[1] &= \qquad\qquad\qquad\qquad \mathrm{Xt2}[2] + \mathrm{Pt2}[2] \\
\beta_{2,\mathrm{max}} &= \max(\beta_2[0],\ \beta^2[1])
\end{aligned}$$

Step 2:

$$\begin{aligned}
\beta_2[0]_{8\text{-bit}} &= \beta_2[0] - \beta_{2,\mathrm{max}} + 127 \\
\beta_2[1]_{8\text{-bit}} &= \beta_2[1] - \beta_{2,\mathrm{max}} + 127 \\
\beta_2[2]_{8\text{-bit}} &= -128 \\
\beta_2[3]_{8\text{-bit}} &= -128 \\
\beta_2[4]_{8\text{-bit}} &= -128 \\
\beta_2[5]_{8\text{-bit}} &= -128 \\
\beta_2[6]_{8\text{-bit}} &= -128 \\
\beta_2[7]_{8\text{-bit}} &= -128
\end{aligned}$$

**$K_T$ =0:**

First MAP decoder:

$$\begin{aligned}
\beta_1[0]_{8\text{-bit}} &= 127 \\
\beta_1[1]_{8\text{-bit}} &= -128 \\
\beta_1[2]_{8\text{-bit}} &= -128 \\
\beta_1[3]_{8\text{-bit}} &= -128 \\
\beta_1[4]_{8\text{-bit}} &= -128 \\
\beta_1[5]_{8\text{-bit}} &= -128 \\
\beta_1[6]_{8\text{-bit}} &= -128 \\
\beta_1[7]_{8\text{-bit}} &= -128
\end{aligned}$$

Second MAP decoder:

$$\begin{aligned}
\beta_2[0]_{8\text{-bit}} &= 127 \\
\beta_2[1]_{8\text{-bit}} &= -128 \\
\beta_2[2]_{8\text{-bit}} &= -128 \\
\beta_2[3]_{8\text{-bit}} &= -128 \\
\beta_2[4]_{8\text{-bit}} &= -128 \\
\beta_2[5]_{8\text{-bit}} &= -128 \\
\beta_2[6]_{8\text{-bit}} &= -128 \\
\beta_2[7]_{8\text{-bit}} &= -128
\end{aligned}$$

### 3.3.9 Endianess Considerations

The endianess manager is responsible for managing the endianess of the data transferred over the 128-bit VBUS_DMA when the DSP is configured in big endian mode. Since the TCP3D is a native little endian coprocessor, when the DSP is configured in little endian mode, the endianess manager has no effect on this data. The operational mode is determined by a system wide signal which is connected to the TCP3D_*endian_big_endian* input of the TCP3D. When high, big endian mode is selected. When low, little endian mode is selected.

The TCP3D always works in little endian mode, so the input/output data to/from the processing unit is always formatted as little endian. Therefore the role of the endianess manager is to order the data correctly when the DSP is configured in big endian mode.

The endianess manager has no effect on the 32-bit configuration data transferred on the VBUS_CFG bus, since byte endianess is not an issue with this interface.

**Note—**The text in the following sections refers to “native mode” data formats. “Native mode” refers to the way in which the TCP3D treats data from the VBUS_DMA. As an example, the “native mode” for the systematic data on the VBUS_DMA is bytes because each byte contains a new systematic data quantity. If the DSP also loads and stores the systematic data as bytes, it is treating it in the “native mode.”

**Note—**The text in the following sections refers to “packed” data formats. “Packed” refers to the way in which the DSP treats the data. As an example, if the DSP loads and stores 8-bit systematic data as 32-bit words, the systematic data is considered to be “packed into 32-bit words.”

#### 3.3.9.1 VBUS_DMA Big Endian Word Swapping

All of the memories and configuration registers shown in Table 4-3, Table 4-4, and Table 4-5 are accessed via the 128-bit VBUS_DMA interface. As such, all of data to/from this interface passes through the endianess manager and is reformatted when the DSP is in big endian mode.

For all of the big endian accesses to the input configuration registers and output configuration registers the endianess manager swaps the 32-bit words within the 128-bit quad word on the VBUS_DMA as data is transferred to/from the TCP3D internal logic.

Figure 3-10 and Figure 3-11 show how the data is swapped.

**Figure 3-10 TCP3D Internal Format for 128-bit Data (i.e. Native Little Endian Format)**

| 127 96 | 95 64 | 63 32 | 31 0 |
|---|---|---|---|
| A | B | C | D |

**Figure 3-11 28-bit VBUS_DMA Data Format for Big Endian Mode**

| 127 96 | 95 64 | 63 32 | 31 0 |
|---|---|---|---|
| D | C | B | A |

Additional swapping capability is provided for the Output/Decision, Soft Output, Interleaver, Systematic, and Parity memories, as shown in Section 3.3.9.2, Section 3.3.9.3, Section 3.3.9.4, and Section 3.3.9.5.

#### 3.3.9.2 Output/Decision Data Formatting

The decisions are packed into a 128-bit word inside the TCP3D and stored as 32-bit words in the DSP. The internal format of the data in the output/decision memory data generated by the TCP3D is shown in Figure 3-12. The format of this data on the VBUS_DMA is modified by the endianess manager based on the endian mode and the state of the *OUT_ORDER_SEL* bit in TCP3D_IC_CFG2_P0 shown in Section 4.6.3. Figure 3-13, Figure 3-14, Figure 3-15, and Figure 3-16 show how the VBUS_DMA data is formatted for all 4 possible combinations of the endian mode and *OUT_ORDER_SEL*.

**Figure 3-12 TCP3D Internal Format of Hard Decision Data**

| 127 | 126 | | 96 | 95 | | 64 | 63 | | 32 | 31 | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit N | Bit N-1 | | Bit N-31 | Bit N-32 | | Bit N-63 | Bit N-64 | | Bit N-95 | Bit N-96 | | Bit N-127 |

**Figure 3-13 VBUS_DMA Format of Hard Decision Data in Little Endian Mode (OUT_ORDER = 0)**

| 127 | 126 | | 96 | 95 | | 64 | 63 | | 32 | 31 | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit N | Bit N-1 | | Bit N-31 | Bit N-32 | | Bit N-63 | Bit N-64 | | Bit N-95 | Bit N-96 | | Bit N-127 |

**Figure 3-14 VBUS_DMA Format of Hard Decision Data in Little Endian Mode (OUT_ORDER = 1)**

| 127 | 126 | | 96 | 95 | | 64 | 63 | | 32 | 31 | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit N-31 | Bit N-30 | | Bit N | Bit N-63 | | Bit N-32 | Bit N-95 | | Bit N-64 | Bit N-127 | | Bit N-96 |

**Figure 3-15 VBUS_DMA Format of Hard Decision Data in Big Endian Mode (OUT_ORDER = 0)**

| 127 | 126 | | 96 | 95 | | 64 | 63 | | 32 | 31 | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit N-96 | Bit N-97 | | Bit N-127 | Bit N-64 | | Bit N-95 | Bit N-32 | | Bit N-63 | Bit N | | Bit N-31 |

**Figure 3-16 VBUS_DMA Format of Hard Decision Data in Big Endian Mode (OUT_ORDER = 1)**

| 127 | 126 | | 96 | 95 | | 64 | 63 | | 32 | 31 | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit N-127 | Bit N-126 | | Bit N-96 | Bit N-95 | | Bit N-64 | Bit N-63 | | Bit N-32 | Bit N-31 | | Bit N |

**Note—**For hard decision data in big endian mode and OUT_ORDER=0, a multiple of 4-bytes must always be read to assure that the correct data is written into system memory by the EDMA controller. As an example, reading 5-bytes of data to transfer Bit[N-127:N-88] to system memory would result in Bit[N-127:N-96] and Bit[N-71:N-64] being written into system memory instead. To eliminate this problem, read 8-bytes and that would correspond to Bit[N-127:N-64] being written into system memory.

### 3.3.9.3 Soft Output Data Formatting

The TCP3D systematic, parity 0, and parity 1 soft output bits are 8-bits wide, and 16 are packed into a 128-bit word. Figure 3-17 shows the internal format of all of the soft output memories.

**Figure 3-17 TCP3D Internal Format of Soft Output Data**

| 127 | 119 | 111 | 103 | 95 | 87 | 79 | 71 | 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SO 15 | SO 14 | SO 13 | SO 12 | SO 11 | SO 10 | SO 9 | SO 8 | SO 7 | SO 6 | SO 5 | SO 4 | SO 3 | SO 2 | SO 1 | SO 0 |

In little endian mode, the format of the VBUS_DMA data is always the same as the internal memory. In big endian mode, the format of the data on the VBUS_DMA is modified by the endianess manager based on the state of the *SOFT_OUT_ORDER_SEL* bit in TCP3D_IC_CFG2_P0. *SOFT_OUT_ORDER_SEL* is set to a 1 for native mode or a 0 for packed 32-bit words (Figure 3-19). In native mode, the swapping is done on 8-bit boundaries, as shown in Figure 3-18

**Figure 3-18 VBUS_DMA Soft Output Data Format in Big Endian Mode and SOFT_OUT_ORDER_SEL = 1 (Native Mode)**

| 127 | 119 | 111 | 103 | 95 | 87 | 79 | 71 | 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SO 0 | SO 1 | SO 2 | SO 3 | SO 4 | SO 5 | SO 6 | SO 7 | SO 8 | SO 9 | SO 10 | SO 11 | SO 12 | SO 13 | SO 14 | SO 15 |

**Figure 3-19 VBUS_DMA Soft Output Data Format in Big Endian Mode and SOFT_OUT_ORDER_SEL = 0 (Packed Mode)**

| 127 | 119 | 111 | 103 | 95 | 87 | 79 | 71 | 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SO 3 | SO 2 | SO 1 | SO 0 | SO 7 | SO 6 | SO 5 | SO 4 | SO 11 | SO 10 | SO 9 | SO 8 | SO 15 | SO 14 | SO 13 | SO 12 |

When the input data is packed into 32-bit words, the swapping is done on 32-bit word boundaries, as shown in 3.2.2 .

**Note—**For packed soft output data in big endian mode, a multiple of 4-bytes must always be read to assure that the correct data is written into system memory by the EDMA controller. As an example, reading 7-bytes of data to transfer SO[6:0] to system memory would result in SO[3:0] and SO[7:5] being written into system memory instead. To eliminate this problem, read 8-bytes and that would correspond to SO[7:0].

### 3.3.9.4 Interleaver Data Formatting

The TCP3D uses 13-bit interleaver indices that are stored right justified in a 16-bit field in the interleaver memory, as shown in Figure 3-20. The DSP can store these indices as either individual 16-bit (i.e. native) quantities or packed into 32-bit words. In little endian mode, the format of the VBUS_DMA data is the same as the internal memory regardless of how the DSP stored the indices.

**Figure 3-20 TCP3D Internal Format of Interleaver Indices**

| 127 112 | 111 96 | 95 80 | 79 64 | 63 48 | 47 32 | 31 16 | 15 0 |
|---|---|---|---|---|---|---|---|
| INTER7 | INTER6 | INTER5 | INTER4 | INTER3 | INTER2 | INTER1 | INTER0 |

In big endian mode, the format of the VBUS_DMA data will differ depending on the state of the *ENDIAN_INTR* bit in TCP3D_END. *ENDIAN_INTR* is set to a 1 for native mode or a 0 when the interleaver indices are packed into 32-bit words. In native mode, the swapping is done on 16-bit half word boundaries, as shown in Figure 3-21.

**Figure 3-21 VBUS_DMA Interleaver Index Format in Big Endian Mode and ENDIAN_INTR = 1 (Native Mode)**

| 127 112 | 111 96 | 95 80 | 79 64 | 63 48 | 47 32 | 31 16 | 15 0 |
|---|---|---|---|---|---|---|---|
| INTER0 | INTER1 | INTER2 | INTER3 | INTER4 | INTER5 | INTER6 | INTER7 |

When the interleaver indices are packed into 32-bit words, the swapping is done on 32-bit word boundaries, as shown in Figure 3-22.

**Figure 3-22 VBUS_DMA Interleaver Index Format in Big Endian Mode and ENDIAN_INTR = 0 (Packed Mode)**

| 127 112 | 111 96 | 95 80 | 79 64 | 63 48 | 47 32 | 31 16 | 15 0 |
|---|---|---|---|---|---|---|---|
| INTER1 | INTER0 | INTER3 | INTER2 | INTER5 | INTER4 | INTER7 | INTER6 |

**Note—**For packed interleaver data in big endian mode, a multiple of 4-bytes must always be written. As an example, if 6 bytes of data were written in this mode, they would correspond to INTER1, INTER0, and INTER3 instead of the INTER[2:0]. To eliminate this problem, write 8-bytes and that would correspond to INTER [3:0].

### 3.3.9.5 Input Data Formatting

The TCP3D uses 6 bit parity and systematic input data values that are stored right justified on 8 bit boundaries, as shown in Figure 3-23. The DSP can store these as either individual 8 bit (i.e. native) quantities or packed into 32-bit words. In little endian mode, the format of the VBUS_DMA data is the same as the internal memory regardless of how the DSP stored the input data.

**Figure 3-23 Figure 30: TCP3D Internal Format of Input Data**

| 127 | 119 | 111 | 103 | 95 | 87 | 79 | 71 | 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LLR 15 | LLR 14 | LLR 13 | LLR 12 | LLR 11 | LLR 10 | LLR 9 | LLR 8 | LLR 7 | LLR 6 | LLR 5 | LLR 4 | LLR 3 | LLR 2 | LLR 1 | LLR 0 |

In big endian mode, the format of the VBUS_DMA data will differ depending on the state of the *ENDIAN_INDATA* bit in TCP3D_END. *ENDIAN_INDATA* is set to a 1 for native mode or a 0 for packed 32-bit words. In native mode, the swapping is done on 8 bit boundaries, as shown in Figure 3-24.

**Figure 3-24 VBUS_DMA Input Data Format in Big Endian Mode and ENDIAN_INDATA = 1 (Native Mode)**

| 127 | 119 | 111 | 103 | 95 | 87 | 79 | 71 | 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LLR 0 | LLR 1 | LLR 2 | LLR 3 | LLR 4 | LLR 5 | LLR 6 | LLR 7 | LLR 8 | LLR 9 | LLR 10 | LLR 11 | LLR 12 | LLR 13 | LLR 14 | LLR 15 |

When the input data is packed into 32-bit words, the swapping is done on 32-bit word boundaries, as shown in Figure 3-25.

**Figure 3-25 VBUS_DMA Input Data Format in Big Endian Mode and ENDIAN_INDATA = 0 (Packed Mode)**

| 127 | 119 | 111 | 103 | 95 | 87 | 79 | 71 | 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LLR 3 | LLR 2 | LLR 1 | LLR 0 | LLR 7 | LLR 6 | LLR 5 | LLR 4 | LLR 11 | LLR 10 | LLR 9 | LLR 8 | LLR 15 | LLR 14 | LLR 13 | LLR 12 |

**Note—**For packed input data in big endian mode, a multiple of 4-bytes must always be written. As an example, if 7 bytes of data were written in this mode, they would correspond to LLR[3:0] and LLR[7:5] instead of the desired LLR[6:0]. To eliminate this problem, write 8 bytes and that would correspond to LLR[7:0].

## 3.4 EDMA3 Setup

This section describes how to setup and use the EDMA3 with the TCP3D.

### 3.4.1 EDMA3 Events

The TCP3D generates a receive event to the EDMA3 controller when it finishes decoding a code block in order to transfer the decoded code block, soft outputs (optional), and output status (optional) back to DSP memory.

To support double-buffered and split modes, the TCP3D provides two independent event signals (REVT0 and REVT1) to the EDMA3 controller. In split mode, the process 0 TCP3D interface state machine generates REVT0 and the process 1 state machine generates REVT1. In double-buffered mode, REVT0 is generated for code blocks that were loaded into the process 0 input buffers, and REVT1 is generated for code blocks that were loaded into the process 1 input buffers. In single-buffered mode only REVT0 is used. To facilitate system debug, the TCP3D provides a method for software to trigger each of these events by writing to the TCP3D_EV_SET_P0/**P1** registers (Section 4.3.7).

### 3.4.2 Auto Trigger Mode

The TCP3D supports an auto trigger mode that eliminates the need for the EDMA controller to write to one of the trigger registers (TCP3D_TRIG_P0 or TCP3D_TRIG_P1) to initiate the decoding process. The benefit of this mode is that is saves EDMA controller resources by requiring fewer DMA transfers to use the TCP3D. In auto trigger mode, writes to TCP3D_TRIG_P0 and TCP3D_TRIG_P1 will not trigger the *Interface State Machine*.

The auto trigger mode is enabled by setting the *AUTO_TRIG_EN* bit in the TCP3D_MODE register to a 1. In auto trigger mode, the TCP3D will self trigger the *Interface State Machin*e when all of the input data for a code block has been loaded into the TCP3D. This includes all of the input configuration registers and systematic, parity 0, and parity 1 input data. The TCP3D uses the *BLK_LN* field in TCP3D_IC_CFG1 to determine how much data it expects to be loaded into the input data memories. Note that it needs to add 1 to the BLK_LN field to get the actual code block size because BLK_LN is defined as code block size - 1.

For 3GPP, the input data and interleaver table lengths may have to be extended as described in Section 3.3.1. In these cases, the auto trigger logic must use the extended lengths to decide when to trigger the decoding.

### 3.4.3 EDMA3 Programming

#### 3.4.3.1 Transfer Overview

The EDMA3 controller has to chain up to four transfers of data from the system memory to the TCP3D to provide all of the input data for a block. The interleaver indices transfer is optional and the trigger register transfer is not required if the TCP3D is in auto trigger mode. The four transfers are as follows:

- Input configuration registers
- Input systematic, parity 0, and parity 1 data
- Trigger register (not required in auto trigger mode)
- Interleaver indices (optional)

**Note—**All of the input configuration registers must be loaded for each code block regardless of whether all of the values are used for that code block and regardless of whether the values are the same as the previous code block. They must also be loaded as the first transfer for a code block (i.e. before systematic, parity 0, parity1, and interleaver data).

The EDMA3 controller has to chain up to three DMA transfers of data from the TCP3D to the system memory to transfer the decoded data and status information out of the TCP3D. The soft output memory and output status registers transfers are optional.

- Output/Decision Data Transfer

  Note: In all modes except split mode, only the Output/Decision Memory 0 is read. For split mode, the Output/Decision Memory 0 is read for process 0 code blocks, and the Output/Decision Memory 1 is read for process 1 code blocks.

- Soft Output Data Transfer (optional)
  - Systematic soft output memory

    Note: In all modes other than split mode, the contents of all of the soft output memories (i.e. systematic, parity 0 and parity 1) must be transferred if the *soft_out_flag_en* bit in the TCP3D_IC_CFG2_P0 register is set to a 1. Failure to do so will cause the TCP3D to hang. In split mode, only the contents of the systematic soft output memory must be transferred.

  - Parity 0 soft output memory (not supported in split mode)
  - Parity 1 soft output memory (not supported in split mode)

- Output Status Registers Transfer (optional)

  Note: In all modes except split mode, the process 0 output status registers are read if the *OUT_FLAG_EN* bit is set in TCP3D_IC_CFG2_P0 or TCP3D_IC_CFG2_P1. For split mode, the process 0 output status registers are read for the process 0 code blocks, and the process 1 output status registers are read for the process 1 code blocks.

#### 3.4.3.2 Data Arrangement

The data arrangements in the system memory and TCP3D memory for single MAP (3GPP mode) and dual MAP (LTE and WiMAX modes) decoding are shown in Figure 3-26 and Figure 3-27, respectively.

**Figure 3-26 Memory Arrangement For Single MAP (3GPP mode) Decoding for K-bit Code Block.**

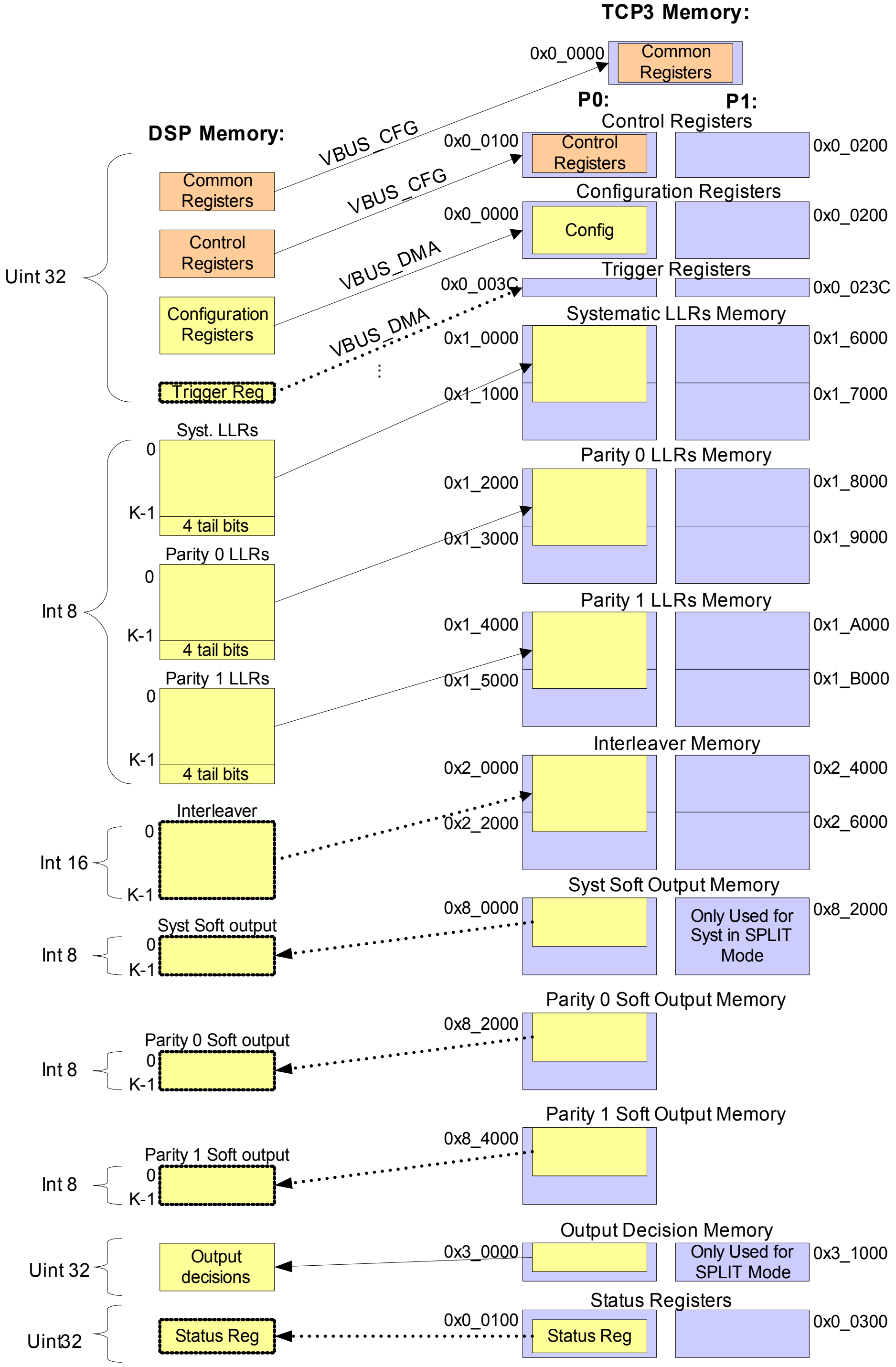
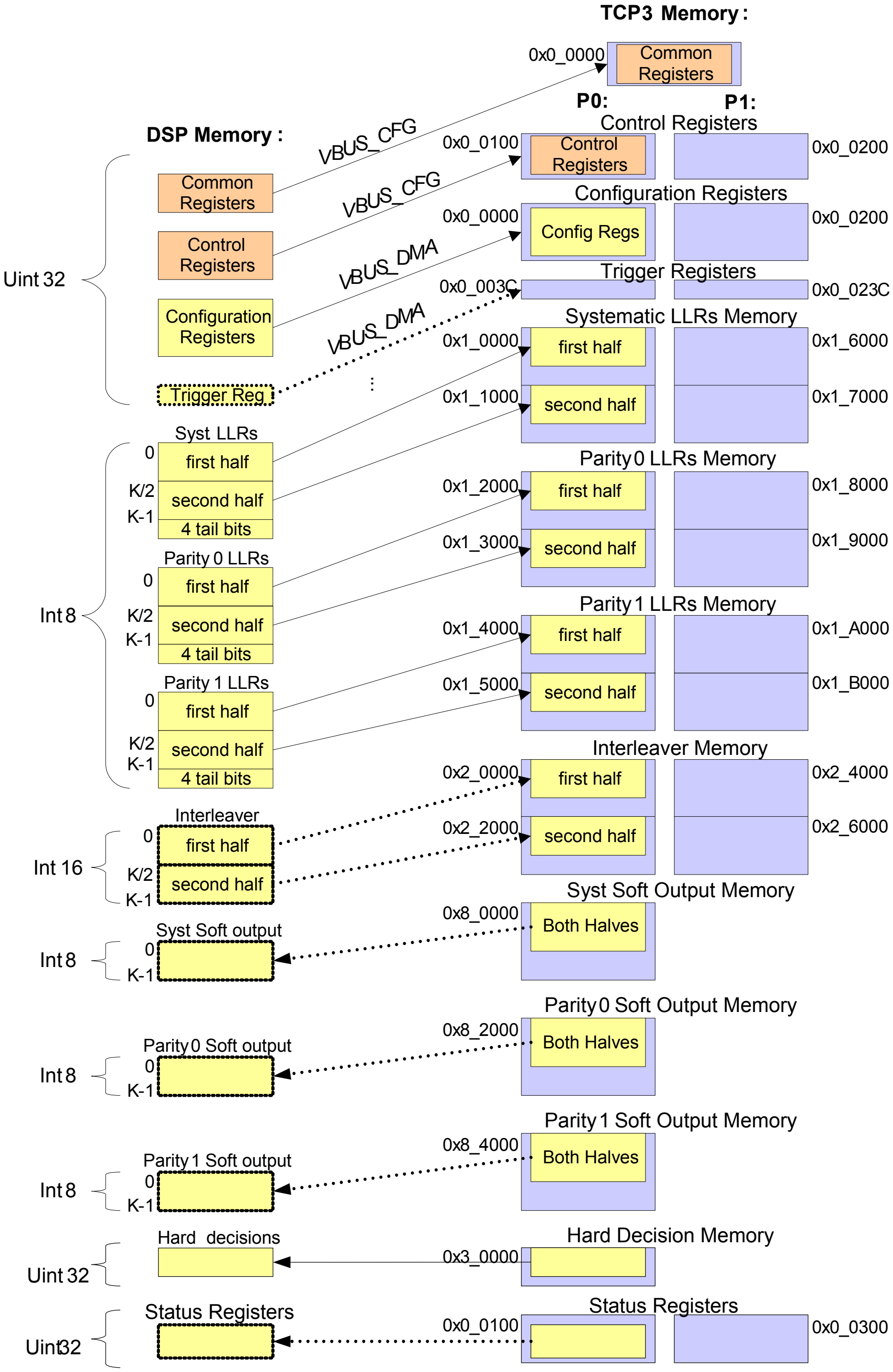


**Figure 3-27 Memory Arrangement for Dual MAP (LTE and WiMAX modes) Decoding for K-bit Code Block.**



Note that the input data is divided in half and written into different halves of the input memories in dual MAP (LTE and WiMAX) modes. Here are some notes concerning Figure 3-26 and Figure 3-27:

- In Figure 3-26 (3GPP mode), the code block size K is potentially extended as discussed in Section 3.3.1.
- In Figure 3-26 (3GPP mode), the tail bits are processed by the DSP to create the initial beta states loaded into the TCP3D input configuration registers (e.g. TCP3D_IC_CFG4_P0 and TCP3D_IC_CFG4_P1). These calculations are described in Section 3.3.8. Some or all of the tail bits are also potentially appended to the end of the input data transferred as explained in Section 3.3.4.
- In Figure 3-27 (LTE and WiMAX modes), the 12 tail bits shown are only for LTE. These are used in the same way as the 3GPP tail bits. The location shown of the 12 tail bits in DSP memory is not a requirement of the TCP3D. These can be located anywhere in DSP memory and Figure 3-34 shows the implications of two different possible locations on the EDMA3 programming. WiMAX does not use tail bits.
- In both Figure 3-26 and Figure 3-27, the value of K must also be modified if the TCP3D is setup for big-endian packed applications. In that case, the number of bytes of systematic, parity 0, parity 1 and interleaver input data written into the TCP3D must be a multiple of 4 bytes as described in Section 3.3.9.5 and Section 3.3.9.4.

  Also, the number of bytes of soft output systematic, parity 0, and parity 1 data read out of the TCP3D must be a multiple of 4 bytes as described in Section 3.3.9.3. The number of bytes of hard decision data read out of the TCP3D for big endian applications where the *OUT_ORDER_SEL* bit in **TCP3D_IC_CFG2_P0** or **TCP3D_IC_CFG2_P1** is set to 0 must also be a multiple of 4-bytes as described in Section 3.3.9.2.

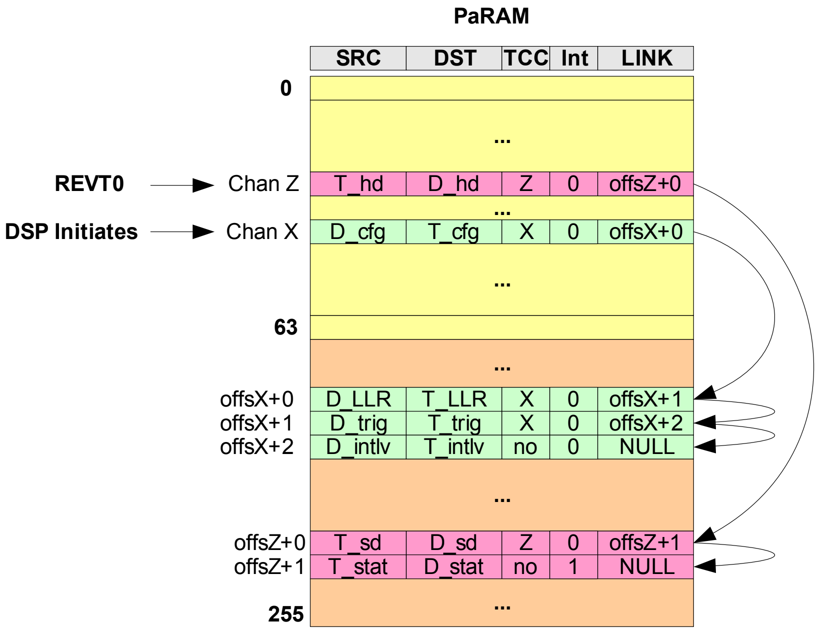
#### 3.4.3.3 Single Code Block PaRAM Entry Setup

This section shows how to setup the EDMA3 PaRAM entries to decode a single code block. The first example shows the entries required to transfer the mandatory data as well as all the optional data. The second example shows the entries required to transfer only the mandatory data. An example of decoding multiple code blocks is given in Section 3.4.3.4.

#### 3.4.3.3.1 Transfer All Optional Data

Figure 3-28 shows how to set up the EDMA to transfer the mandatory data as well as all the optional data. The optional data includes the interleaver table and trigger register input transfers as well as the soft outputs and status register output transfers.

**Figure 3-28 Single Code Block PaRAM Entry Setup with All Optional Data**

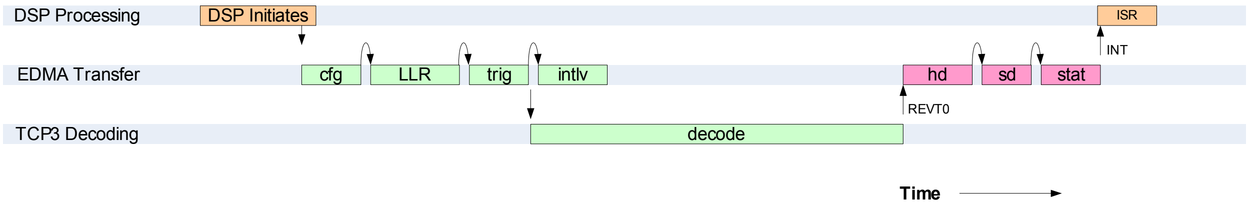


PaRAM

| | SRC | DST | TCC | Int | LINK |
|---|---|---|---|---|---|
| 0 | | | | | |
| | | | ... | | |
| REVT0 → Chan Z | T_hd | D_hd | Z | 0 | offsZ+0 |
| | | | ... | | |
| DSP Initiates → Chan X | D_cfg | T_cfg | X | 0 | offsX+0 |
| | | | ... | | |
| 63 | | | | | |
| | | | ... | | |
| offsX+0 | D_LLR | T_LLR | X | 0 | offsX+1 |
| offsX+1 | D_trig | T_trig | X | 0 | offsX+2 |
| offsX+2 | D_intlv | T_intlv | no | 0 | NULL |
| | | | ... | | |
| offsZ+0 | T_sd | D_sd | Z | 0 | offsZ+1 |
| offsZ+1 | T_stat | D_stat | no | 1 | NULL |
| 255 | | | ... | | |

Figure 3-29 shows the entries required to transfer only the mandatory data.

**Figure 3-29 Single Code Block Transfer Sequence with All Optional Data**



The sequence of events is:

- DSP initiates process by setting the bit corresponding to EDMA3 channel X in the EDMA controller event set register.
- Using linking and self chaining approach, 4 chunks of data are transferred from DSP to TCP3D.
- The last transfer, the interleaver data, is linked to NULL.
- After receiving trigger register, TCP3D starts decoding first half of the first iteration and waits for interleaver data.
- Once decoding is done, TCP3D sends REVT0, which initiates the transfer of the results, hard decisions, soft decisions, and status register, using linking and self chaining approach.
- After the last transfer is completed, EDMA generates interrupt to DSP to indicate that decoding is completed.

- The transfer of the LLRs can be either 2D or 3D transfer.

#### 3.4.3.3.2 Transfer Only Necessary Data

The decoding process can be reduced to only 3 event transfers when:

- TCP3D auto trigger option is used: the decoding automatically starts once the LLRs are transferred. In this case the trigger register does not need to be sent.
- TCP3D is configured to generate the interleaver table internally.
- User needs only the hard decisions.

Figure 3-30 illustrates PaRAMEntry setup and Figure 3-31 shows the transfer sequence with only necessary data for single code blocks.

**Figure 3-30 Single Code Block PaRAM Entry Setup with Only Necessary Data**

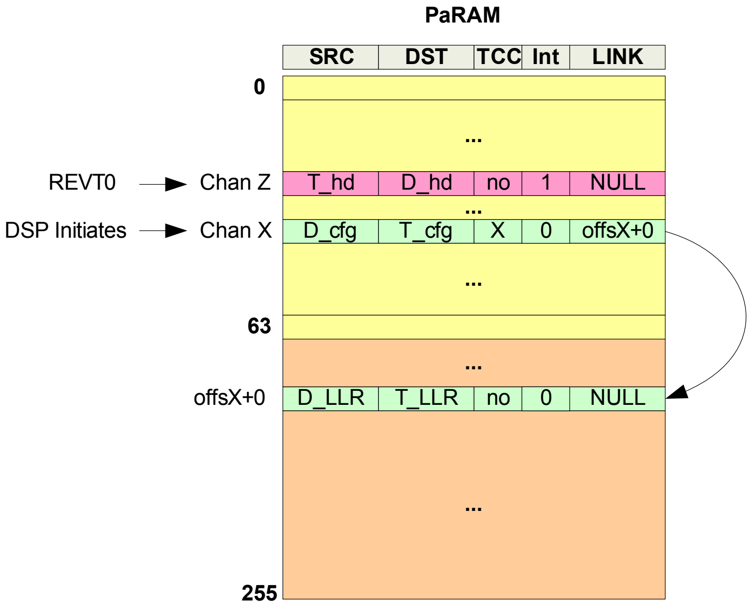
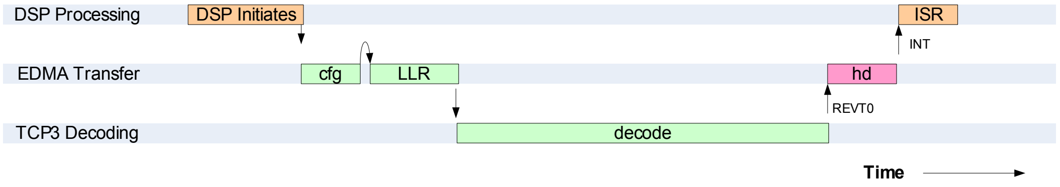


**Figure 3-31 Single Code Block Transfer Sequence with Only Necessary Data**

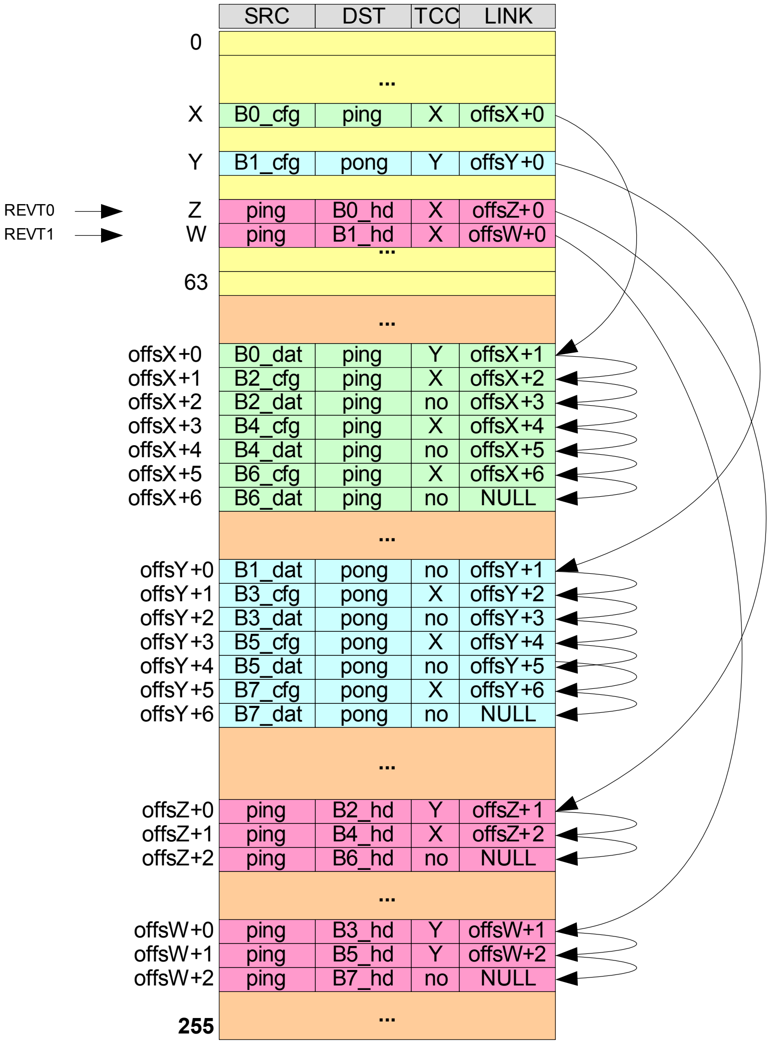


### 3.4.3.4 Multiple Code Block PaRAM Entry Setup

In this example, the DSP programs the decoding of 8 code blocks and only transfers the necessary data. The TCP3D operates in double-buffered mode. After all 8 blocks are decoded the EDMA3 generates an interrupt to the DSP. Some advantages to this kind of multiple code block setup is that it requires minimal DSP involvement and the TCP3D is fully engaged. Some disadvantages are an increase in the latency of accessing the decoded data for the earlier code blocks and the resource usage of many PaRAM entries.

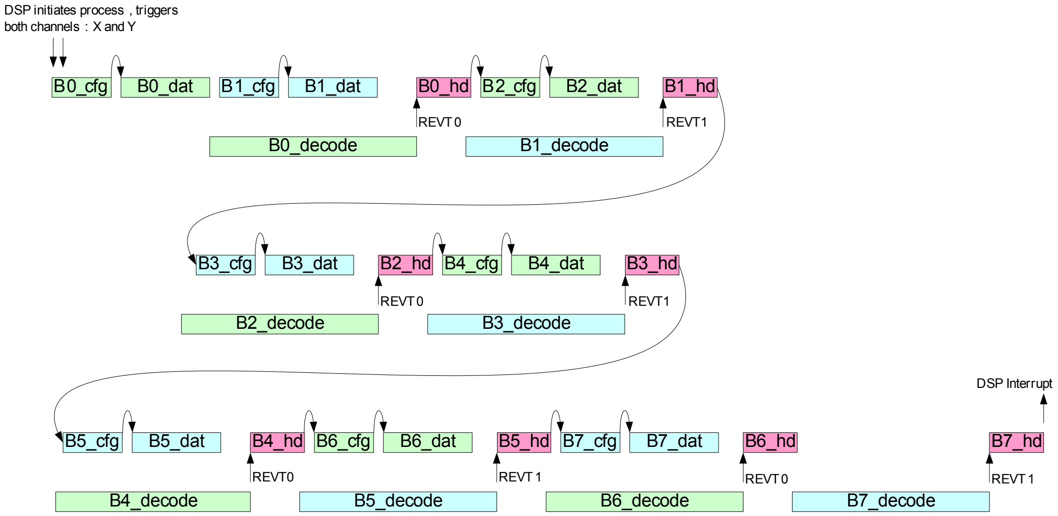
Figure 3-32 illustrates PaRAMEntry setup and Figure 3-33 shows the transfer sequence with only necessary data for multiple code blocks.

**Figure 3-32 Multiple Code Block PaRAM Entry Setup with Only Necessary Data**



| | SRC | DST | TCC | LINK |
|---|---|---|---|---|
| 0 | | | | |
| | ... | | | |
| X | B0_cfg | ping | X | offsX+0 |
| | | | | |
| Y | B1_cfg | pong | Y | offsY+0 |
| | | | | |
| Z | ping | B0_hd | X | offsZ+0 |
| W | ping | B1_hd | X | offsW+0 |
| | ... | | | |
| 63 | | | | |
| | ... | | | |
| offsX+0 | B0_dat | ping | Y | offsX+1 |
| offsX+1 | B2_cfg | ping | X | offsX+2 |
| offsX+2 | B2_dat | ping | no | offsX+3 |
| offsX+3 | B4_cfg | ping | X | offsX+4 |
| offsX+4 | B4_dat | ping | no | offsX+5 |
| offsX+5 | B6_cfg | ping | X | offsX+6 |
| offsX+6 | B6_dat | ping | no | NULL |
| | ... | | | |
| offsY+0 | B1_dat | pong | no | offsY+1 |
| offsY+1 | B3_cfg | pong | X | offsY+2 |
| offsY+2 | B3_dat | pong | no | offsY+3 |
| offsY+3 | B5_cfg | pong | X | offsY+4 |
| offsY+4 | B5_dat | pong | no | offsY+5 |
| offsY+5 | B7_cfg | pong | X | offsY+6 |
| offsY+6 | B7_dat | pong | no | NULL |
| | ... | | | |
| offsZ+0 | ping | B2_hd | Y | offsZ+1 |
| offsZ+1 | ping | B4_hd | X | offsZ+2 |
| offsZ+2 | ping | B6_hd | no | NULL |
| | ... | | | |
| offsW+0 | ping | B3_hd | Y | offsW+1 |
| offsW+1 | ping | B5_hd | Y | offsW+2 |
| offsW+2 | ping | B7_hd | no | NULL |
| 255 | ... | | | |

Figure 3-33 Multiple Code Block Transfer Sequence with Only Necessary Data



**Note—**Channels X and Y can be merged to one channel X. In this case, if the transfer *B0_dat* is a 3D transfer, it could chain to the next transfer *B1_cfg*. Both OPT fields, ITCCHEN and TCCHEN, would be set to 1 and TCC to X. The whole process would be started by triggering channel X.

### 3.4.3.5 PaRAM Entry Details

This section provides the PaRAM entry details for each transfer.

#### 3.4.3.5.1 Input Configuration Register Transfer

The PaRAM entry details for this transfer are as follows.

- OPT (options):
  - ITCCHEN = 0 (Intermediate transfer complete chaining is disabled)
  - TCCHEN = 1 (Transfer complete chaining is enabled)
  - ITCINTEN = 0 (Intermediate transfer complete interrupt is disabled)
  - TCINTEN = 0 (Transfer complete interrupt is disabled)
  - TCC = varies, see sections 3.4.3.3 and 3.4.3.4 (Transfer complete code)
  - TCCMODE = 0 (Normal completion)
  - FWID = 0 (FIFO width, don't care because src and dest are not FIFOs)
  - STATIC = 0 (Entry is updated as normal)
  - SYNCDIM = 0 (A-sync. Each event triggers the transfer of ACNT elements.)
  - DAM = 0 (Dest addressing within an array increments. Dest is not a FIFO.)
  - SAM = 0 (Source addressing within an array increments. Source is not a FIFO.)
- SRC (address) = User input configuration parameters global start address
- ACNT = 60 (Number of bytes in an array) (All 15 input configuration registers must be loaded and each is 32-bits, or 4 bytes)
- BCNT = 1 (Number of arrays of length ACNT)
- DST (address) = TCP3D_IC_CFG0_P0/P1 global address
- SRCBIDX = 0 (Source 2nd dimension index)

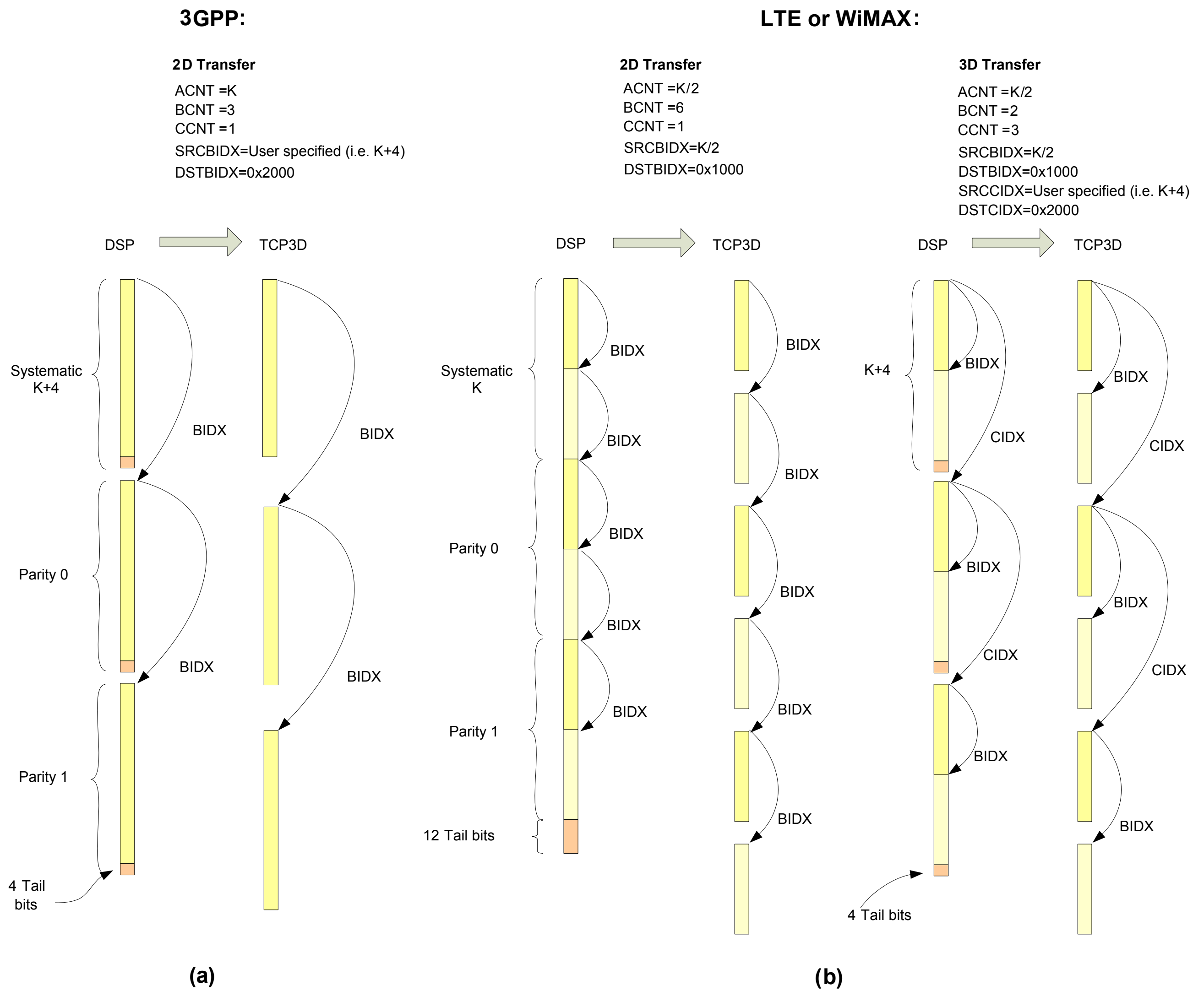
- DSTBIDX = 0 (Destination 2nd dimension index)
- BCNTRLD = 0 (BCNT reload, don't care because CCNT is 1)
- SRCCIDX = 0 (Source 3rd dimension index)
- DSTCIDX = 0 (Destination 3rd dimension index)
- CCNT = 1 (Number of frames in a block)
- LINK (address) = Address in the EDMA3 PaRAM of the EDMA3 parameters associated with the corresponding input data transfer (see sections 3.4.3.3 and 3.4.3.4 )

#### 3.4.3.5.2 Input Data Transfer

The systematic, parity 0 and parity 1 data can be moved from system memory to TCP3D as a single entry, AB-synchronized transfer when the starting addresses of the three arrays in system memory are separated by the same offset amount as illustrated in Figure 3-34. The 2D and 3D transfer options are shown for LTE or WiMAX to account for the different possible locations of the tail bits (LTE only) in DSP memory.

**Note—**The value of K (block size) may need to be modified as described in Section 3.4.3.2for 3GPP or big-endian packed applications.

**Figure 3-34 EDMA3 Transfer Of Code Block LLRs in (a) 3GPP and (b) LTE/WiMAX Modes.**



The PaRAM entry details for this transfer are:

- OPT (Options):
  - ITCCHEN = 0 (Intermediate transfer complete chaining is disabled)
  - TCCHEN = varies, see sections 3.4.3.3 and 3.4.3.4 (Transfer complete chaining)
  - ITCINTEN = 0 (Intermediate transfer complete interrupt is disabled)
  - TCINTEN = 0 (Transfer complete interrupt is disabled)
  - TCC = varies, see sections 3.4.3.3 and 3.4.3.4 (Transfer complete code)
  - TCCMODE = 0 (Normal completion)
  - FWID = 0 (FIFO Width, don't care because src and dest are not FIFOs)
  - STATIC = 0 (Entry is updated as normal)
  - SYNCDIM = 1 (AB-Sync. Each event triggers the transfer of BCNT arrays of ACNT bytes.)
  - DAM = 0 (Dest addressing within an array increments. Dest is not a FIFO.)
  - SAM = 0 (Source addressing within an array increments. Source is not a FIFO.)
- SRC (address) = User systematic input data global start address
- ACNT = see Figure 3-34 and text at beginning of section 3.4.3.2 (because the value of K may need to be modified for 3GPP mode or big-endian packed applications) (Number of bytes in an array)
- BCNT = see Figure 3-34 (Number of arrays of length ACNT)
- DST (address) = Systematic Data Memory 0/1 global base address
- SRCBIDX = see Figure 3-34 (Source 2nd Dimension Index)
- DSTBIDX = see Figure 3-34 (Destination 2nd Dimension Index)
- BCNTRLD = same value used for BCNT (BCNT reload)
- SRCCIDX = see Figure 3-34, only used for 3D transfer, don't care for 2D transfers (Source 3rd Dimension Index)
- DSTCIDX = see Figure 3-34, only used for 3D transfer, don't care for 2D transfers (Destination 3rd Dimension Index)
- CCNT = see Figure 3-34 (Number of frames in a block)
- LINK (address) = Address in the EDMA3 PaRAM of the EDMA3 parameters associated with the next transfer (see sections 3.4.3.3 and 3.4.3.4 )

#### 3.4.3.5.3 Input Trigger Register Transfer (Optional)

The PaRAM entry details for this transfer are as follows.

- OPT (Options):
  - ITCCHEN = 0 (Intermediate transfer complete chaining is disabled)
  - TCCHEN = varies, see sections 3.4.3.3 and 3.4.3.4 (Transfer complete chaining)
  - ITCINTEN = 0 (Intermediate transfer complete interrupt is disabled)
  - TCINTEN = 0 (Transfer complete interrupt is disabled)
  - TCC = varies, see sections 3.4.3.3 and 3.4.3.4 (Transfer Complete Code)
  - TCCMODE = 0 (Normal Completion)
  - FWID = 0 (FIFO Width, don't care because src and dest are not FIFOs)
  - STATIC = 0 (Entry is updated as normal)
  - SYNCDIM = 0 (A-Sync. Each event triggers the transfer of ACNT elements.)
  - DAM = 0 (Dest addressing within an array increments. Dest is not a FIFO.)
  - SAM = 0 (Source addressing within an array increments. Source is not a FIFO.)
- SRC (address) = User trigger register data global address
- ACNT = 4 (Number of bytes in an array) (Trigger register is 32-bits, or 4 bytes)
- BCNT = 1 (Number of arrays of length ACNT)
- DST (address) = TCP3D_TRIG_P0/P1 register global address
- SRCBIDX = 0 (Source 2nd Dimension Index)
- DSTBIDX = 0 (Destination 2nd Dimension Index)
- BCNTRLD = 0 (BCNT reload, don't care since CCNT is 1)
- SRCCIDX = 0 (Source 3rd Dimension Index)
- DSTCIDX = 0 (Destination 3rd Dimension Index)
- CCNT = 1 (Number of frames in a block)
- LINK (address) = Address in the EDMA3 PaRAM of the EDMA3 parameters associated with the next transfer (see sections 3.4.3.3 and 3.4.3.4 )

#### 3.4.3.5.4 Input Interleaver Table Transfer (Optional)

The PaRAM entry details for this transfer are:

- OPT (Options):
  - ITCCHEN = 0 (Intermediate transfer complete chaining is disabled)
  - TCCHEN = varies, see sections 3.4.3.3 and 3.4.3.4 (Transfer complete chaining)
  - ITCINTEN = 0 (Intermediate transfer complete interrupt is disabled)
  - TCINTEN = 0 (Transfer complete interrupt is disabled)
  - TCC = varies, see sections 3.4.3.3 and 3.4.3.4 (Transfer Complete Code)
  - TCCMODE = 0 (Normal Completion)
  - FWID = 0 (FIFO Width, don't care since src and dest are not FIFOs)
  - STATIC = 0 (Entry is updated as normal)
  - SYNCDIM = 0 (A-Sync. Each event triggers the transfer of ACNT elements.)
  - DAM = 0 (Dest addressing within an array increments. Dest is not a FIFO.)
  - SAM = 0 (Source addressing within an array increments. Source is not a FIFO.)

- SRC (address) = User interleaver table data global start address
- ACNT = code block size K × 2, check text at beginning of section 3.4.3.2 because the value of K may need to be modified for 3GPP mode or big-endian packed applications (Number of bytes in an array) (Each interleaver index is 16 bits, or 2 bytes)
- BCNT = 1 (Number of arrays of length ACNT)
- DST (address) = Interleaver Memory 0/1 global base address
- SRCBIDX = 0 (Source 2nd Dimension Index)
- DSTBIDX = 0 (Destination 2nd Dimension Index)
- BCNTRLD = 0 (BCNT reload, don't care because CCNT is 1)
- SRCCIDX = 0 (Source 3rd Dimension Index)
- DSTCIDX = 0 (Destination 3rd Dimension Index)
- CCNT = 1 (Number of frames in a block)
- LINK (address) = Address in the EDMA3 PaRAM of the EDMA3 parameters associated with the next transfer (see sections 3.4.3.3 and 3.4.3.4 )

#### 3.4.3.5.5 Output Decision Data Transfer

The PaRAM entry details for this transfer are:

- OPT (Options):
  - ITCCHEN = 0 (Intermediate transfer complete chaining is disabled)
  - TCCHEN = varies, see sections 3.4.3.3 and 3.4.3.4 (Transfer complete chaining)
  - ITCINTEN = 0 (Intermediate transfer complete interrupt is disabled)
  - TCINTEN = varies, see sections 3.4.3.3 and 3.4.3.4 (Transfer complete interrupt)
  - TCC = varies, see sections 3.4.3.3 and 3.4.3.4 (Transfer Complete Code)
  - TCCMODE = 0 (Normal Completion)
  - FWID = 0 (FIFO Width, don't care because src and dest are not FIFOs)
  - STATIC = 0 (Entry is updated as normal)
  - SYNCDIM = 0 (A-Sync. Each event triggers the transfer of ACNT elements.)
  - DAM = 0 (Dest addressing within an array increments. Dest is not a FIFO.)
  - SAM = 0 (Source addressing within an array increments. Source is not a FIFO.)
- SRC (address) = Output/Decision Memory 0 global base address for single and double-buffering modes, Output/Decision Memory 0/1 global base address for split mode
- ACNT = ceil[code block size K ÷ 8], check text at beginning of section 3.4.3.2 because the value of K may need to be modified for big-endian packed applications (Number of bytes in an array) (Each hard decision is a single bit)
- BCNT = 1 (Number of arrays of length ACNT)
- DST (address) = User decoded data memory global start address for this code block
- SRCBIDX = 0 (Source 2nd Dimension Index)
- DSTBIDX = 0 (Destination 2nd Dimension Index)
- BCNTRLD = 0 (BCNT reload, don't care since CCNT is 1)
- SRCCIDX = 0 (Source 3rd Dimension Index)
- DSTCIDX = 0 (Destination 3rd Dimension Index)

- CCNT = 1 (Number of frames in a block)
- LINK (address) = Address in the EDMA3 PaRAM of the EDMA3 parameters associated with the next transfer (see sections 3.4.3.3 and 3.4.3.4 )

#### 3.4.3.5.6 Soft Output Data Transfer (Optional)

The PaRAM entry details for this transfer are:

- OPT (Options):
  - ITCCHEN = 0 (Intermediate transfer complete chaining is disabled)
  - TCCHEN = varies, see sections 3.4.3.3 and 3.4.3.4 (Transfer complete chaining)
  - ITCINTEN = 0 (Intermediate transfer complete interrupt is disabled)
  - TCINTEN = varies, see sections 3.4.3.3 and 3.4.3.4 (Transfer complete interrupt)
  - TCC = varies, see sections 3.4.3.3 and 3.4.3.4 (Transfer Complete Code)
  - TCCMODE = 0 (Normal Completion)
  - FWID = 0 (FIFO Width, don't care because src and dest are not FIFOs)
  - STATIC = 0 (Entry is updated as normal)
  - SYNCDIM = 1 (AB-Sync. Each event triggers the transfer of BCNT arrays of ACNT bytes.)
  - DAM = 0 (Dest addressing within an array increments. Dest is not a FIFO.)
  - SAM = 0 (Source addressing within an array increments. Source is not a FIFO.)
- SRC (address) = Systematic Soft Output Memory global base address for single and double-buffered modes, Systematic Soft Output Memory or Parity 0 Soft Output Memory global base address for split mode (for process 0 or process 1, respectively)
- ACNT = code block size K, check text at beginning of section 3.4.3.2 because the value of K may need to be modified for big-endian packed applications (Number of bytes in an array) (Each soft output is 8 bits, or 1 byte)
- BCNT = 3 for single and double-buffered modes, 1 for split mode (Number of arrays of length ACNT)
- DST (address) = User soft output data storage global start address for this code block
- SRCBIDX = 0x2000 (Source 2nd dimension index) (Offset between soft output memory base addresses)
- DSTBIDX = User specified, but should be at least as large as ACNT to avoid overwriting data (Destination 2nd dimension index)
- BCNTRLD = 0 (BCNT reload, don't care because CCNT is 1)
- SRCCIDX = 0 (Source 3rd dimension index)
- DSTCIDX = 0 (Destination 3rd dimension index)
- CCNT = 1 (Number of frames in a block)
- LINK (address) = Address in the EDMA3 PaRAM of the EDMA3 parameters associated with the next transfer (see sections 3.4.3.3 and 3.4.3.4 )

#### 3.4.3.5.7 Output Status Register Transfer (Optional)

The PaRAM entry details for this transfer are:

- OPT (Options):
  - ITCCHEN = 0 (Intermediate transfer complete chaining is disabled)
  - TCCHEN = varies, see sections 3.4.3.3 and 3.4.3.4 (Transfer complete chaining)
  - ITCINTEN = 0 (Intermediate transfer complete interrupt is disabled)
  - TCINTEN = varies, see sections 3.4.3.3 and 3.4.3.4 (Transfer complete interrupt)
  - TCC = varies, see sections 3.4.3.3 and 3.4.3.4 (Transfer complete code)
  - TCCMODE = 0 (Normal Completion)
  - FWID = 0 (FIFO Width, don't care because src and dest are not FIFOs)
  - STATIC = 0 (Entry is updated as normal)
  - SYNCDIM = 0 (A-Sync. Each event triggers the transfer of ACNT elements.)
  - DAM = 0 (Dest addressing within an array increments. Dest is not a FIFO.)
  - SAM = 0 (Source addressing within an array increments. Source is not a FIFO.)
- SRC (address) = TCP3D_OUT_STS0_P0/P1 global address
- ACNT = 12 (Number of bytes in an array) (Three 32-bit registers, or 12 bytes)
- BCNT = 1 (Number of arrays of length ACNT)
- DST (address) = User output status memory global start address for this code block
- SRCBIDX = 0 (Source 2nd Dimension Index)
- DSTBIDX = 0 (Destination 2nd Dimension Index)
- BCNTRLD = 0 (BCNT reload, don't care because CCNT is 1)
- SRCCIDX = 0 (Source 3rd dimension index)
- DSTCIDX = 0 (Destination 3rd dimension index)
- CCNT = 1 (Number of frames in a block)
- LINK (address) = Address in the EDMA3 PaRAM of the EDMA3 parameters associated with the next transfer (see sections 3.4.3.3 and 3.4.3.4 )

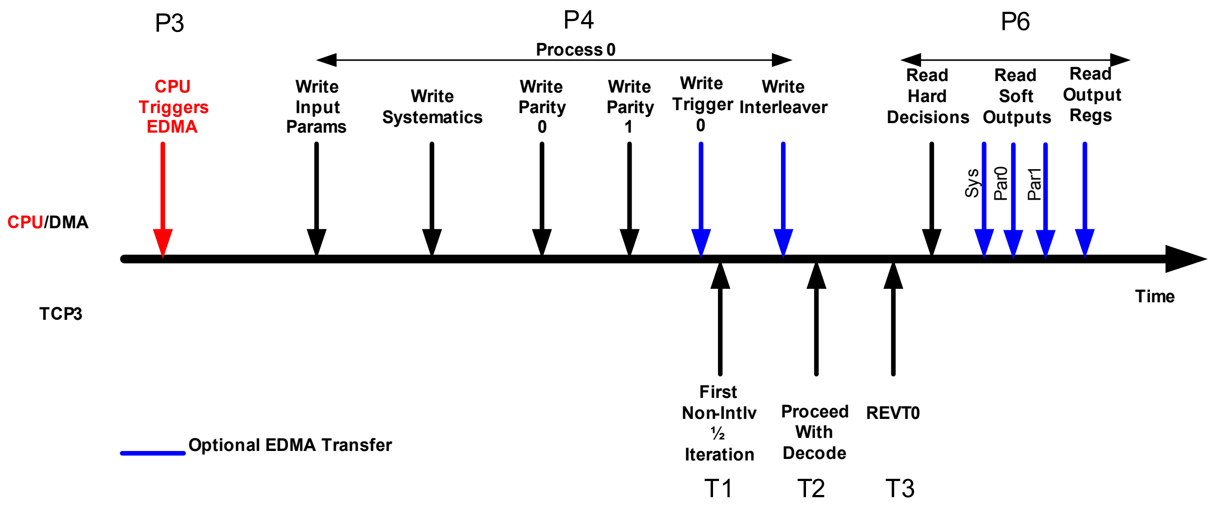
### 3.4.4 Example Event Sequences

This section contains an example event sequence for each of the three data processing modes: single-buffered, double-buffered, and split mode.

#### 3.4.4.1 Example Single-Buffered Mode Sequence

The following sequence shows an example of how the single-buffered mode is used in conjunction with the DSP and EDMA controller. In this mode, only the Process 0 resources are used. A time line of the sequence is shown in Figure 3-35 and an explanation is provided below. The time line shows the sequence if the TCP3D is not in auto trigger mode. The *write trigger 0* transfer would be eliminated for auto trigger mode. The time line also has cross references to the explanation.

**Figure 3-35 Figure 42: TCP3D Single Buffered Mode Flow**



##### 3.4.4.1.1 DSP and EDMA Processing

| Step | Explanation |
|---|---|
| P1 | The DSP receives an indication that code block 0 is ready for turbo decoding in system memory and it knows that the TCP3D is idle. |
| P2 | The DSP configures the EDMA controller to transfer the configuration registers and input data, including the interleaver indices, for code block 0 to the process 0 registers and memories. It also configures the EDMA controller to transfer the decoded data and status out of TCP3D when the decoding is finished. In addition, the DSP also sets up the EDMA controller to generate an interrupt when the decoded data and output status registers have been transferred to system memory. Section 3.4.3 details the four DMA transfers that must be set up in this case. |
| P3 | The DSP triggers the EDMA controller to kick off the transfers. |
| P4 | The EDMA controller executes the four data transfers (three for auto trigger mode) to move the input data from the system memory to the TCP3D. |
| P5 | The DSP receives an indication that code block 1 is ready for turbo decoding in system memory, but it cannot send it to the TCP3D because it has not yet received the completion interrupt from the EDMA controller. |
| P6 | The EDMA controller receives the REVT0 event from the TCP3D. It moves the decoded code block (i.e. from output/decision memory 0), all of the soft output bits (i.e. systematic, parity 0, and parity 1), and the process 0 output status registers from TCP3D to system memory. |
| P7 | The DSP receives the completion interrupt from the EDMA controller and can now process code block 1. It repeats the processing for code block 1 from Step P2. |

#### 3.4.4.1.2 TCP3D Processing

| Step | Explanation |
|---|---|
| T1 | The first non-interleaved iteration of the MAP decoder begins when the DMA transfer to **TCP3D_TRIG_P0** occurs. (For auto trigger mode, the decoder begins after *Write Parity 1* transfer.) |
| T2 | The first interleaved half iteration of the MAP decoder begins when both the first non-interleaved half iteration completes and the DMA transfer of the interleaver indices completes. |
| T3 | The TCP3D finishes decoding code block 0 and generates an REVT0 event to the EDMA controller. It now waits for the decoded code block to be read by the EDMA controller. Once this is done, the TCP3D waits for all of the soft output bits (i.e. systematic, parity 0, and parity 1) to be read by the EDMA controller if the *soft_out_flag_en* bit in the **TCP3D_IC_CFG2_P0** register is set, and the output status registers to be read by the EDMA controller if the *out_flag_en* bit in the **TCP3D_IC_CFG2_P0** register is set. |
| T4 | When the previous DMA transfer finishes, the TCP3D Interface State Machine keeps waiting for another trigger (i.e. created by writing to **TCP3D_TRIG_P0**) to indicate that there is another code block (i.e. code block 1) ready to be decoded. (For auto trigger mode, the TCP3D polls the internal self-trigger bit.) |

The processing repeats from Step T1 for code block 1.

#### 3.4.4.1.3 Cascading Multiple Blocks

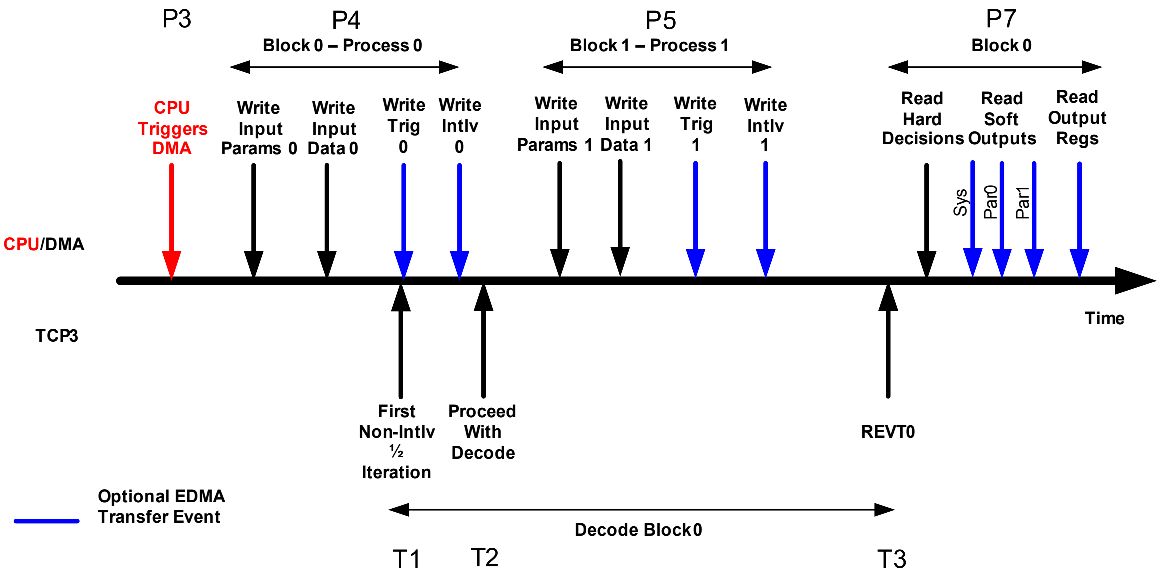
In cases where the DSP has multiple blocks of data ready for decoding at the same time, it can configure the EDMA controller to cascade the DMA transfers for each code block without further DSP intervention. **It does this by chaining the DMA transfers for the next code block to the last DMA transfer (i.e. triggered by the last REVT0) for the current block.**

### 3.4.4.2 Example Double-Buffered Mode Sequence

Double-buffered mode (i.e. ping/pong mode) provides a very efficient system interface to the TCP3D. In this mode, both the Process 0 and Process 1 input resources are used. Since the TCP3D contains two complete sets of configuration registers and input data memories, the system can load the next code block to be decoded while the TCP3D is decoding the current block. This approach can result in a reduction in the data transfer latency for the next block.

The following sequence shows how the double-buffered mode is used in conjunction with the DSP and EDMA controller for decoding two blocks that are available at the same time. A time line of the sequence is shown in Figure 3-36. In this figure, only the first and last two DMA transfers for the input data are shown. Also, it shows only the MAP decode and output DMA transfers for one of the two blocks. An explanation of the flow is provided below. The time line shows the sequence if the TCP3D is not in auto trigger mode. The *Write Trigger* transfer would be eliminated for auto trigger mode. The time line also has cross references to the explanation.

**Figure 3-36 TCP3D Double Buffered Mode Flow**



### 3.4.4.2.1 DSP and EDMA Processing

| Step | Explanation |
|---|---|
| P1 | The DSP receives an indication that blocks 0 and 1 are ready for Turbo decoding in system memory, and it knows that both Process 0 and Process 1 are idle. |
| P2 | The DSP configures the EDMA controller to transfer the configuration registers and input data, including the interleaver indices, for code block 0 to the Process 0 registers and memories and for code block 1 to the Process 1 registers and memories. It also configures the EDMA controller to transfer the decoded data out of TCP3D when the decoding is finished for each block. The DSP also sets up the EDMA controller to generate an interrupt when the decoded data and output status registers have been transferred to system memory for both blocks. The DSP fills up both buffers inside the TCP3D by chaining the end of the DMA for the code block 0 input data to the start of the DMA for the code block 1 input data. Section 3.4.3 details the four DMA transfers that must be set up for each block. |
| P3 | The DSP triggers the EDMA controller to kick off the transfers. |
| P4 | The EDMA controller executes the four data transfers (three for auto trigger mode) to move the code block 0 input data from the system memory to the TCP3D. |
| P5 | The EDMA controller executes the four data transfers to move the code block 1 input data from the system memory to the TCP3D. |
| P6 | The DSP receives an indication that code block 2 is ready for Turbo decoding in system memory, but it cannot send it to the TCP3D because it has not yet received a completion interrupt from the EDMA controller for code block 1. |
| P7 | The EDMA controller receives the REVT0 event from the TCP3D and moves the decoded code block (i.e. from Output/Decision Memory 0), all of the soft output bits (i.e. systematic, parity 0, and parity 1), and the Process 0 output status registers for code block 0 from TCP3D to system memory. |
| P8 | The EDMA controller receives a REVT1 event from the TCP3D and moves the decoded code block (i.e. from the Output/Decision Memory 0 even though the code block was stored in the Process 1 input registers and memories), all of the soft output bits (i.e. systematic, parity 0, and parity 1), and the Process 0 output status registers for code block 1 from TCP3D to system memory. |
| P9 | The DSP receives the completion interrupt for code blocks 0 and 1 from the EDMA controller and can now configure the EDMA controller for code block 2 starting at Step P2. |

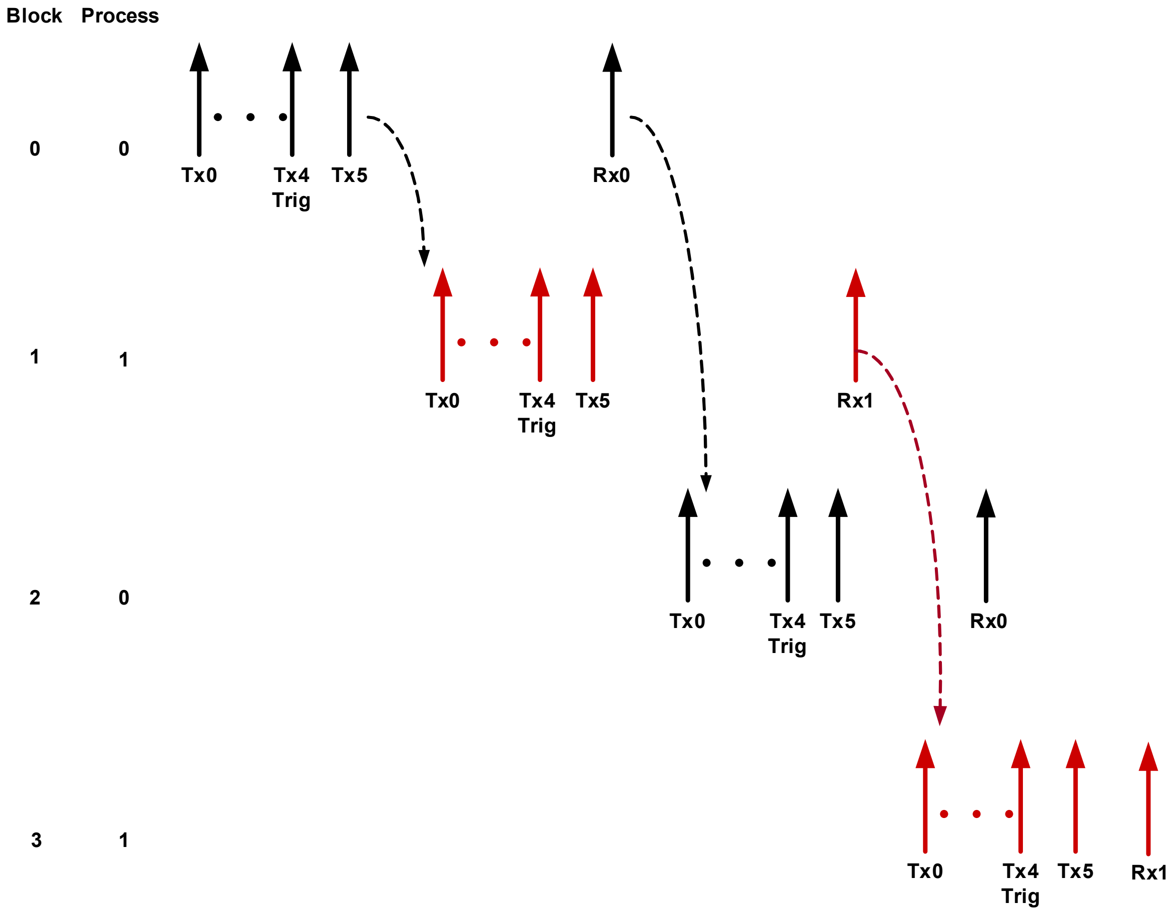
#### 3.4.4.2.2 TCP3D Processing

| Step | Explanation |
|---|---|
| T1 | The first non-interleaved iteration of the MAP decoder for code block 0 begins when the DMA transfer to **TCP3D_TRIG_P0** occurs. (For auto trigger mode, the decoder starts after all of the input data has been loaded into the TCP3D.) |
| T2 | The first interleaved half iteration of the MAP decoder begins when both the first non-interleaved half iteration completes and the DMA transfer of the interleaver indices completes. |
| T3 | The TCP3D finishes decoding code block 0 and generates an REVT0 event to the EDMA controller. It now waits for the decoded code block to be read by the EDMA controller. Once this is done, the TCP3D waits for all of the code block 0 soft output bits (i.e. systematic, parity 0, and parity 1), to be read by the EDMA controller if the *soft_out_flag_en* bit in the **TCP3D_IC_CFG2_P0** register is set, and the code block 0 output status registers to be read by the EDMA controller if the *out_flag_en* bit in the **TCP3D_IC_CFG2_P0** register is set. |
| T4 | When the previous DMA transfer finishes, the TCP3D Interface State Machine detects that code block 1 is available in bank 1 because of the trigger created by writing to T**CP3D_TRIG_P1**. (For auto trigger mode, the TCP3D reads an internal self-trigger bit.) |
| T5 | The first non-interleaved half iteration of the MAP decoder for code block 1 begins immediately since the trigger is already set. |
| T6 | The first interleaved half iteration of the MAP decoder begins immediately after the completion of the first non-interleaved half iteration since the interleaver indices have already been transferred to the TCP3D. |
| T7 | The TCP3D finishes decoding code block 1 and generates an REVT1 event to the EDMA controller. It now waits for the decoded code block 1 to be read by the EDMA controller. Once this is done, the TCP3D waits for all of the code block 1 soft output bits (i.e. systematic, parity 0, and parity 1), to be read by the EDMA controller if the soft_out_flag_en bit in the **TCP3D_IC_CFG2_P1** register is set, and the code block 1 output status registers to be read by the EDMA controller if the out_flag_en bit in the **TCP3D_IC_CFG2_P1** register is set. |
| T8 | When the previous DMA transfer finishes, the TCP3D Interface State Machine keeps waiting for another trigger (i.e. created by writing to **TCP3D_TRIG_P0** or **TCP3D_TRIG_P1**) to indicate that there is another code block (i.e. code block 2) ready to be decoded. (For auto trigger mode, the TCP3D reads the internal self-trigger bits.) |

#### Cascading Multiple Blocks

Figure 3-37 shows how the DMA transfers for a group of four blocks can be chained when the TCP3D is in double-buffered mode. It shows the initial chaining from code block 0 to code block 1 used to load the 2 buffers. It also shows the chaining from the end of code block 0 to the start of code block 2, and the chaining from the end of code block 1 to the start of code block 3. The end of code block 0 indicates that the bank 0 data can be over written, and the code block 2 data can be moved to the bank 0 memories while code block 1 is being decoded. (For auto trigger mode, the *Tx4 Trigger* transfer does not exist.)

Figure 3-37 TCP3D Double Buffered Mode DMA Chaining for 4 Blocks Available at the Same Time



### 3.4.4.3 Example Split Mode Sequence

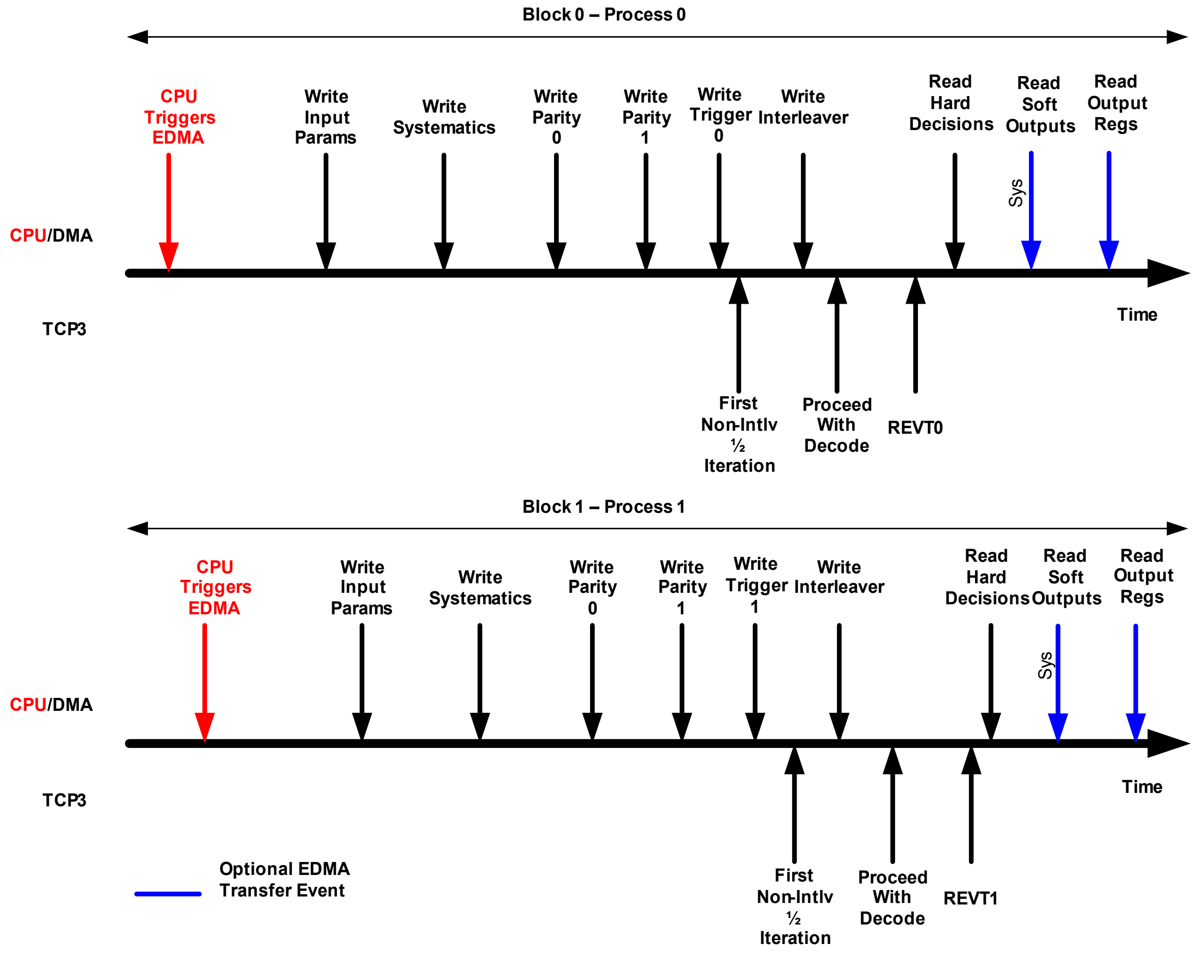
Split mode configures the TCP3D resources (i.e. memory, MAP decoder A and MAP decoder B) into 2 independent turbo decoder coprocessors that share common VBUS interfaces. In this mode, the TCP3D can decode 2 blocks in parallel.

Figure 3-38 shows a sample usage sequence for split mode. The sequence assumes that each MAP decoder is controlled by a separate software driver and that the drivers are not synchronized. Also, this scheme requires separate EDMA channels for each MAP decoder and separate receive events for each process (REVT0 and REVT1).

Figure 3-38 shows the parallelism that can exist in split mode. It shows the separate EDMA channels for each MAP decoder starting very close to each other. The transfers for each process are skewed in time to illustrate the parallelism. A description of the DSP, DMA and TCP3D processing for split mode is not provided because, with the exception of the soft output bits, each code block is processed exactly the same as the code block described for the single-buffered mode. For split mode, only the systematic soft output bits are provided by TCP3D, and Figure 3-38 illustrates this. The "*Write Trigger*" transfers would be eliminated for auto trigger mode.

For split mode, the Output/Decision Memory 0 is read for Process 0 code blocks, and the Output/Decision Memory 1 is read for Process 1 code blocks. The Process 0 output status registers are read for the Process 0 code blocks, and the Process 1 output status registers are read for the Process 1 code blocks.

**Figure 3-38 TCP3D Split Mode Flow**



## 3.5 TCP3D Control and Decoding

This section describes how to control different aspects of the TCP3D operation.

### 3.5.1 TCP3D Interface State Machine (ISM)

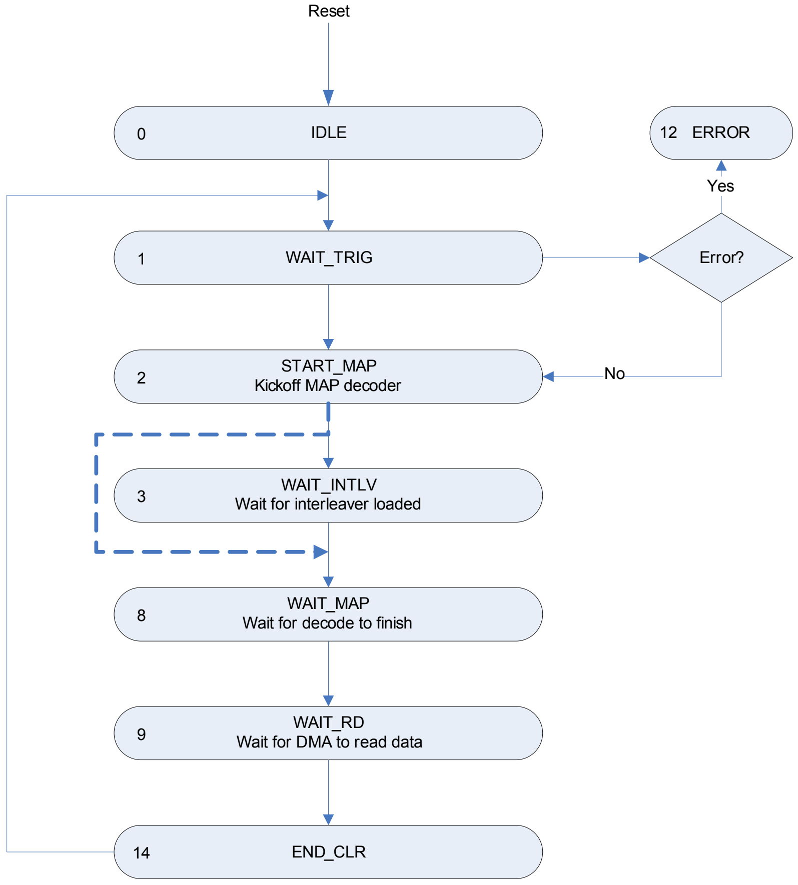
The Interface State Machine (ISM) manages the interface between the system and the TCP3D decoding process. It receives a command from the system to start decoding a code block, and in response, it kicks off the MAP decoders. When the MAP decoders are finished they notify the ISM and it requests that the decoded code block be transferred to system memory by generating a DMA event.

The TCP3D has two completely independent ISMs. The second one is only used for split mode process 1. In the descriptions below, unless otherwise noted, all of the register references are to the process 0 registers that are used for single-buffered mode, double-buffered mode, and split mode process 0.

The TCP3D has two main modes of operation: normal mode and debug mode. The sequencing of states for the normal mode is described in this section. The debug mode usage is discussed in Section 3.6.1 and the hooks added for emulation support are discussed in Section 3.6.2.2.

The state diagram for the normal mode operation is shown in Figure 3-39. Solid lines in the figures represent required states and dashed lines represent optional states.

**Figure 3-39 TCP3D Interface State Machine - Normal Mode**



After the TCP3D is reset (or cleared via the TCP3D_EXE_P0 register), the ISM is in the IDLE state. Writing a 1 (enable) to the TCP3D_EXE_P0 register will move the ISM to the WAIT_TRIG state where it is waiting for a set of input configuration registers and input data to be written before being triggered by the trigger register(s) or automatically in auto-trigger mode. If any enabled error conditions exist when triggered, the ISM will enter the ERROR state and stay there until the TCP3D is cleared via the TCP3D_EXE_P0 register (or reset).

If there are no errors, the ISM enters the START_MAP state where it initiates the first non-interleaved half iteration of the MAP decoder (which does not need the interleaver table) and the internal LTE and WiMAX interleaver table generators, if enabled. If the interleaver load (*INTER_LOAD_SEL*) flag in TCP3D_IC_CFG2_P0 is set to a 1, then the ISM goes to the WAIT_INTL state and waits for the interleaver indices to be loaded or generated for the current code block. Otherwise, the decoder will use the previous interleaver indices and the ISM goes directly to the WAIT_MAP state.

After the turbo decoder finishes decoding the current code block and all of the output decisions (i.e. hard and soft decisions) have been generated, a REVT0/1 DMA event is sent to the EDMA3 and the ISM moves to the WAIT_RD state. The state machine waits in this state until all of the selected data has been read from the TCP3D via DMA. The hard decisions in the Output/Decision memory are always read out. If the soft output read flag (*SOFT_OUT_FLAG_EN*) in TCP3D_IC_CFG2_P0 is set to a 1, the state machine will also wait for the contents of the valid Soft Output memories to be read. If the output status register read flag (*OUT_FLAG_EN*) in TCP3D_IC_CFG2_P0 is set to a 1, the state machine will wait for the 3 output status registers (TCP3D_OUT_STS0_P0, TCP3D_OUT_STS1_P0, **and** TCP3D_OUT_STS2_P0) to be read.

When all the data has been read, the ISM branches to the END_CLR state to prepare the TCP3D for a new decode before going back to the WAIT_TRIG state to wait for the next code block to be loaded.

### 3.5.2 Stopping Criteria

The minimum and maximum number of iterations that the decoder will perform before placing the decoded data in the output memory and sending the REVT0/1 signals is governed by the corresponding parameters in the input configuration registers. The decoding may be terminated before the maximum number of iterations is reached due to the SNR or LTE CRC stopping criteria.

#### 3.5.2.1 Max and Min Iteration Parameters

The maximum and minimum number of decoding iterations for the TCP3D is specified in TCP3D_IC_CFG3_P0 and TCP3D_IC_CFG3_P1. The minimum number of iterations is specified in the *MIN_ITR* field and has a range of 0-15 iterations. It must be less than or equal to the maximum number of iterations, or an error is flagged in the *MAX_MIN_IT_ERR* field of TCP3D_EINT_STAT_P0 or TCP3D_EINT_STAT_P1. The SNR stopping logic and the CRC logic are disabled until the number of iterations has reached *MIN_ITR*.

The maximum number of iterations is specified in the *MAX_ITR* field and has a range of 0-15 iterations. If the SNR or LTE CRC stopping criteria are enabled, the decoding can stop before *MAX_ITR* if either stopping criterion is satisfied.

Since the hard decisions are only generated on the non-interleaved half iteration, the maximum number of iterations executed by the TCP3D for all modes except LTE is *MAX_ITR* – 0.5. For LTE, the CRC is calculated from the hard decisions during the interleaved half iteration. As such, the maximum number of iterations executed by the TCP3D for LTE is *MAX_ITR*.

#### 3.5.2.1.1 Iteration Parameter Special Cases

**MAX_ITR = 0**

If the TCP3D Interface State Machine does not halt due to a *MAX_MIN_IT_ERR* error (a *MAX_MIN_IT_ERR* will not occur if *MIN_ITR* = 0 and *MAX_ITR*=0), the TCP3D will stop after one iteration. It is the same as if MAX_ITR=1.

**MIN_ITR = 0**

Since the SNR stopping logic and the CRC logic are disabled until the minimum number of iterations has reached *MIN_ITR*, a value of 0 results in all of these functions being enabled on the first iteration.

**MAX_ITR < MIN_ITR**

If the TCP3D Interface State Machine does not halt due to a *MAX_MIN_IT_ERR* error, the max condition will be satisfied before the min condition, and the SNR stopping logic and the CRC logic will never get enabled

**WiMAX and 3GPP with Soft Outputs Enabled**

Since the Parity 1 Soft Output memory is not written until the interleaved half iteration, reading this memory after only the first non-interleaved half iteration will result in invalid values. To guarantee that this does not occur, set MIN_ITR > 1 and MAX_ITR > 1 for WiMAX and 3GPP applications that use soft outputs. For LTE applications, it is acceptable to set MIN_ITR = 1 and MAX_ITR = 1 since the MAP decoder will always perform the interleaved half-iteration before stopping.

### 3.5.2.2 SNR Stopping Criterion

The SNR stopping criterion is based on measuring the SNR of the extrinsics and comparing it to a specified threshold $T_{SNR}$ . Normally, the turbo decoder executes a fixed number of iterations, but with the SNR stopping criterion enabled, it may terminate earlier, thus saving both time and power.

The turbo decoder calculates the SNR of the extrinsics every non-interleaved half-iteration. If the value is above $T_{SNR}$ , the turbo decoder terminates further iterations and calculates the final decisions. If the value never exceeds $T_{SNR}$ , the turbo decoder keeps iterating until the specified maximum number of iterations is reached, and then it calculates the final decisions.

**Note—**The SNR and LTE CRC stopping criteria are never enabled at the same time. However, the SNR reporting feature described below can be used with the LTE CRC stopping criterion.

The threshold value is in the range from 0 to 20 dB, and it is specified in the *SNR_VAL* field of TCP3D_IC_CFG2_P0 and TCP3D_IC_CFG2_P1. The SNR stopping criterion is enabled by setting the *STOP_SEL* field to 0x2 or 0x3 in TCP3D_IC_CFG2_P0 and TCP3D_IC_CFG2_P1. Internal results of the SNR calculations are reported in the *SNR_M1* field of TCP3D_OUT_STS0_P0 and TCP3D_OUT_STS0_P1 and the *SNR_M2* and *SNR_EXCEED* fields of TCP3D_OUT_STS1_P0 and TCP3D_OUT_STS1_P1.

The TCP3D also supports a mode where the SNR function is active, but it is not used as a stopping criterion. In this mode, the internal results of the SNR calculations are reported in the *SNR_M1*, *SNR_M2,* and *SNR_EXCEED* fields for debugging or monitoring. This mode is enabled by setting the *STOP_SEL* field to 0x0 or 0x1 and the *SNR_REP* bit to 0x1 in TCP3D_IC_CFG2_P0 and TCP3D_IC_CFG2_P1. The TCP3D will support applications where SNR reporting and LTE CRC stopping criterion are both enabled.

#### 3.5.2.3 LTE CRC Stopping Criterion

All LTE blocks contain a 24-bit CRC value at the end. The TCP3D will check this CRC and can use the result as a stopping criterion. The LTE data received by the TCP3D uses two different polynomials to generate the CRC as described below.

LTE transport blocks have a 24-bit CRC appended to the block. If the resultant block size is less than or equal to the maximum LTE block size of 6144 bits, the transport block will not be segmented into code blocks, and the TCP3D must use the following transport block CRC polynomial defined as:

$$1 + x^{1} + x^{3} + x^{4} + x^{5} + x^{6} + x^{7} + x^{10} + x^{11} + x^{14} + x^{17} + x^{18} + x^{23} + x^{24}$$

Transport blocks greater than 6144 bits are segmented into smaller code blocks, and a 24-bit CRC is appended to each code block using a different polynomial. In this case, the TCP3D uses the following code block polynomial defined as:

$$1 + x^{1} + x^{5} + x^{6} + x^{23} + x^{24}$$

The polynomial used by the TCP3D is selected via the *CRC_SEL* bit in TCP3D_IC_CFG2_P0 or TCP3D_IC_CFG2_P1. If this bit is 0, the code block CRC is selected. If this bit is 1, the transport block CRC is selected.

The TCP3D will use the LTE CRC match as a stopping criterion when the *STOP_SEL* field is set to 0x1 in TCP3D_IC_CFG2_P0 or TCP3D_IC_CFG2_P1.

The outputs of the hard decision function are used as inputs to the LTE CRC function. Since the TCP3D processes up to 4 separate sliding windows in parallel, the hard decisions are not generated as one sequential data steam, and the CRC cannot be generated in “real time” as the hard decisions are generated. Instead, the CRC is generated and checked during the interleaved half iteration using the hard decisions stored in the Output Decision 0 memory during the previous non-interleaved half iteration. Since the CRC generator calculates the CRC 8-bits at a time, the CRC will be generated and checked prior to the end of the interleaved half iteration. The CRC is checked by computing a 24-bit CRC over the entire code block including the appended CRC. A CRC result of all zeros at the end of this process indicates a CRC match.

Regardless of whether the CRC is used as a stopping criterion for LTE, the CRC generator and checking logic is always running in LTE mode, and the result of the CRC check on the final hard decisions is always reported via the *LTE_CRC_CHK* bit in TCP3D_OUT_STS1_P0. There is also a dynamic version of *LTE_CRC_CHK* available in TCP3D_STS_P0 that is updated at the end of each CRC calculation, regardless of whether the TCP3D is stopping or not. The CRC calculation results in the TCP3D requiring an additional half iteration to decode LTE blocks as compared to WiMAX or 3GPP blocks.

The LTE CRC function also supports a feature where the CRC has to match on consecutive iterations for the TCP3D to stop. The number of consecutive iterations is programmable via the *CRC_ITER_PASS* field in TCP3D_IC_CFG2_P0 or TCP3D_IC_CFG2_P1.

For future flexibility, the *LTE_CRC_ISEL* bit in TCP3D_MODE can be used to select the initial value (0x00_0000 or 0xFF_FFFF) for the internal CRC registers. Currently, this bit should always be set to 0 to use 0x00_0000 as the initial value.

**Note—NOTE:** The SNR and LTE CRC stopping criteria cannot be enabled at the same time.

### 3.5.3 Channel Quality Indicator

The TCP3D provides Channel Quality Indicator (CQI) logic to monitor the quality (i.e. noise characteristics) of the channel over which the encoded data travels. The CQI logic compares the computed hard decision bits to the sign (positive/negative) of the corresponding systematic soft bits from the systematic memories. For each iteration of the decoder, it will compute the total of all of the mismatches between the hard decision bits and the systematic soft bits. At the end of the decode, the number of mismatches from the last iteration is stored in the *CNT_CQI* field of TCP3D_OUT_STS2_P0. Every mismatch is assumed to be a case where the turbo decoder corrected a systematic soft bit that was corrupted by noise. So, the channel quality is inversely proportional to the number of mismatches.

A systematic soft bit value with all 6 bits equal to 0 is an ambiguous value and will be excluded from the CQI logic described above. Instead, it will be counted separately and reported in the *CNT_CQI_ZERO* field of TCP3D_OUT_STS2_P0.

### 3.5.4 Error Reporting (Error Interrupt)

The TCP3D can generate an interrupt to the DSP if it detects an error. To support double-buffered and split modes the TCP3D provides separate error interrupts for Process 0 and Process 1. This results in the TCP3D driving 2 interrupt outputs. For single-buffered mode only the Process 0 interrupt is active. The TCP3D interrupts are described in more detail below.

#### 3.5.4.1 TCP3D Error Interrupt

The TCP3D will generate an error interrupt for a process when it senses a programming error, which includes setting a control parameter in a register to an invalid value; bad length for DMA transfers; or accesses to the TCP3D memories while it is actively decoding a frame. All of the error conditions are shown in Table 4-14.

The effect of each error condition on the internal operation of the TCP3D is shown in Table 4-15. In many cases the TCP3D can be configured to stop decoding and the DSP must intervene. In other cases, the source of the error can cause the decoder to become corrupted. The DSP is expected to clear error; re-program the external DMA controller; and then re-start the TCP3D via the TCP3D_EXE_P0 or TCP3D_EXE_P1 registers (i.e. issue TCP3D CLEAR command followed by an ENABLE command). An error recovery scheme is detailed in Section 3.5.4.5.

The TCP3D aggregates all of the error conditions for a process into a single, active high, level handshake interrupt that is an output of the TCP3D. It will remain high until the DSP clears the source of the interrupt in the TCP3D. This interrupt output is connected to an Interrupt Distributor Component in the SOC, which conditions the interrupt and delivers it to the target DSP.

There are a number of registers that are implemented to satisfy the requirements of PDR 3.5 and the Highlander Interrupt Architecture. Table 3-6 provides a list of all of the registers associated with the TCP3D error interrupt. The complete description of these registers for the TCP3D error interrupt are found in Table 4-14 through Table 4-25.

**Table 3-6 TCP3D Error Interrupt Registers**

| Register Type | Description |
|---|---|
| TCP3D Error Interrupt Raw Status and Set | This register contains the interrupt status bits for each error condition before any enable processing. The interrupt status bits are set to 1 by an error condition or writing a 1 (Used for debug to allow software to set an interrupt status bit). Writing a 0 has no effect. They are cleared when a 1 is written to the associated bit in the TCP3D Error Interrupt Clear register. |
| TCP3D Error Interrupt Clear | Write-only shadow register that mirrors the interrupt status bits in the TCP3D Error Interrupt Raw Status and Set register. Writing a 1 causes the associated interrupt status bit in the TCP3D Error Interrupt Raw Status and Set register to be cleared to a 0. Used by software to clear an interrupt condition after it has been serviced. Writing a 0 has no effect. Reading this register will return the value in the TCP3D Error Interrupt Raw Status and Set register. |
| TCP3D Error Interrupt Enabled Status | Read-only register that is the logical AND of the TCP3D Error Interrupt Raw Status and Set register and the TCP3D Error Interrupt Enable and Set register. This is only an address and not a physical register. |
| TCP3D Error Interrupt Enable and Set | This register contains the enables for each of the interrupt status bits in the TCP3D Error Interrupt Raw Status and Set register. An interrupt status bit can only cause an interrupt if it is enabled.<br>Interrupt enable bits are set to a 1 by writing a 1, but writing a 0 leaves the value un-changed. Interrupt enable bits are cleared by writing to the TCP3D Interrupt Error Interrupt Enable Clear registers described below. |
| TCP3D Error Interrupt Enable Clear | Write-only shadow register that mirrors the interrupt enable bits in the TCP3D Error Interrupt Enable and Set register. Writing a 1 causes the associated interrupt enable bit in the TCP3D Error Interrupt Enable and Set register to be cleared to a 0 and disables the associated interrupt status bit. Writing a 0 has no effect. Reading this register will return the value in the TCP3D Error Interrupt Enable and Set register. |
| TCP3D End of Interrupt | This register supports the End of Interrupt (EOI) interface between the TCP3D and the external Interrupt Distributor Component. It is a common register that is used for servicing both of the TCP3D interrupts. This register is written by software to acknowledge the interrupt has been cleared so another interrupt of the same type can be generated. This register is written after either of the 2 TCP3D interrupts has been cleared via the associated Interrupt Clear register. |
| **End of Table 3-6** | |

Here are some additional rules that all of the TCP3D interrupts follow:

- If an interrupt condition occurs and the corresponding interrupt raw status bit is already set to a 1, the occurrence of the second interrupt condition will not be saved.
- If the DSP attempts to clear an interrupt raw status bit at the same time it is being set again, the raw status bit will remain set to a 1.

#### 3.5.4.2 TCP3D Error Interrupt Usage

The following example illustrates how the TCP3D Error Interrupt registers shown in Table 3-6 are used (Process 0 assumed):

1. Only allow the TCP3D “block length” error condition to cause an interrupt by writing a 0x2 to the TCP3D Error Interrupt Enable and Set register.
2. A code block length error occurs in the TCP3D when the software programs an invalid value into the *BLK_LN* field in the TCP3D_IC_CFG0 register for the current frame. Since this error condition is enabled, the TCP3D Process 0 error interrupt output (*TCP3D_int_intr[0]*) is asserted.
3. The DSP identifies the source of the interrupt by reading a 0x2 from the TCP3D Error Interrupt Enabled Status register.
4. After servicing the interrupt, the DSP clears the source of the interrupt by writing a 0x2 to the TCP3D Error Interrupt Cle**ar** register. If there are no new error conditions, *TCP3D_int_intr[0]* is negated. Otherwise, it remains asserted.
5. The DSP then writes a value into the *EOI_VECTOR* field the TCP3D End of Interrupt register to acknowledge that it has cleared the TCP3D error interrupt condition. The TCP3D will then generate a transaction on the EOI interface to the external Interrupt Distributor Component. If the TCP3D error interrupt output is still asserted due to another pending error condition, the Interrupt Distributor Component will generate another interrupt to the DSP. The write to the TCP3D End of Interrupt register has no effect on the internal interrupt processing of the TCP3D.

#### 3.5.4.3 Emulation Suspend Support

During emulation suspend mode interrupts are handled as follows.

- The generation of new TCP3D external interrupts is suppressed. This refers to operational interrupts and interrupts that are set via writes to TCP3D_EINT_SET_P0 or TCP3D_EINT_SET_P1. The interrupt raw status bits will still capture any operational interrupt conditions or those created by writes to TCP3D_EINT_SET_P0 or TCP3D_EINT_SET_P1.
- External interrupts that were awaiting service when emulation mode occurred are held at the active high level but can be cleared in the normal manner.
- Suppressed interrupts will be generated when the TCP3D exits emulation mode.

#### 3.5.4.4 Clock Management Support

When the system asserts the CBA clock management request signal, *TCP3D_cs_clkstop_req* (high), the TCP3D will clear any active interrupt conditions by doing the following prior to asserting *TCP3D_cs_clkstop_ack* (high):

- De-assert both external error interrupt signals (*TCP3D_int_intr[1:0]*)
- Clear all interrupt raw status bits in TCP3D_EINT_STAT_P0 and TCP3D_EINT_STAT_P1

Once the clock is started again, the TCP3D will be ready to report any new interrupt conditions since the state of the interrupt enables is not cleared when the clock to the TCP3D is stopped.

#### 3.5.4.5 Error Recovery

This section presents a sample error recovery scheme if the TCP3D *Interface State Machine* halts due one of the errors described in Table 4-15. If the TCP3D is not in SPLIT mode (see SPLIT mode exceptions below), it stops decoding, the error must be cleared, and the TCP3D and EDMA3 controller must be re-started.

The system can detect that the TCP3D has halted via either an interrupt from the TCP3D or, if interrupts are not enabled, a software timeout due to the lack of an expected interrupt from the EDMA3 controller to the DSP indicating that a code block or a group of code blocks have been decoded. The process of servicing an error interrupt, clearing it, and writing TCP3D_EOI is described in Section 3.5.4.2. The next step is to re-start the TCP3D and the EDMA controller, as described below.

- The TCP3D is re-started by issuing a TCP3D CLEAR command to TCP3D_EXE_P0, waiting a minimum of 20 TCP3D clocks, and then writing an ENABLE command to the same register. This puts the TCP3D back into a state where it can accept and process new code blocks. For double-buffer applications, any code blocks that were waiting to be decoded when the TCP3D halted are lost.
- If the TCP3D is in the double-buffer mode, data for the other process may have been in-transit when the *Interface State Machine* halted due to the error. To prevent issuing a TCP3D CLEAR command while data is in still in-transit and then removing it when data is still in-intransit, it is suggested that the DSP not issue the TCP3D CLEAR command until the in-transit data is completely in the TCP3D. This can be detected by waiting for *PROC_PEND* in TCP3D_STS_P0 or TCP3D_STS_P1 to be set to a “1” before issuing the TCP3D CLEAR command.
- Once the EDMA3 controller has been re-programmed, the DSP kicks off the transfer of more code blocks to the TCP3D by creating a manually-triggered transfer request in the EDMA3 controller. The EDMA3 controller can be re-programmed to either re-start from where the TCP3D halted, if that can be identified, or just to skip any old code blocks that were never processed and send in new code blocks.

**SPLIT Mode Exceptions**

In SPLIT mode, an error will only cause one of the processes to halt. The process without the error will operate normally.

### 3.5.5 Throughput and Decoding Performance

This section describes the throughput and the decoding performance of the TCP3D. This is intended as a reference and to aid in the tradeoff and selection of the sliding window 0 (SW0) length input parameter.

#### 3.5.5.1 Throughput

The decoding throughput, B, (bits per sec) for LTE and WiMAX is computed by the following formula. Note the dependence on the length of sliding window 0 ($L_{SW0}$).

$$B = \frac{N_{MAP} \times 2 \times F_{clock}}{(2\mathrm{N}_{\mathrm{iter}} - 1)\left(1 + \frac{2N_{MAP} \times L_{SW0}}{K}\right)}$$

*where:*

$K$ - code block length
$N_{MAP}$ - Number of MAP decoders
$F_{clock}$ - Clock frequency in Hz
$N_{iter}$ - Number of turbo iterations
$L_{SW0}$ - Length of the first sliding window SW

Note that in the last iteration the TCP3D decoder stops execution after the first half of the iteration since it does not compute hard decisions in the second half of the iteration.

The throughput curves for the LTE and WiMAX standard are shown in Figure 3-41 and Figure 3-42, respectively. The throughput increases with the increase of the code block length for a fixed length of SW0. This is due to relative decrease of the first beta computations to the overall computation. The first beta computations are not done in parallel with the alpha computations. Also, for the fixed code block length, the throughput increases with the decrease of the SW0 length. However, as the SW0 length decreases, the number of SWs increases and the BER performance deteriorates, but also the amount of memory to save beta states, (belief propagation memory) increases. Because of this, some curves in the figures end when the belief propagation memory becomes full.

In the case of WiMAX 27-byte code block length, the interleaver does hot have the contention free property. Because of that the processing time of the interleaved pass is doubled. Therefore the throughput is reduced by 1/3.

The throughput curves for the HSUPA+ in split mode are calculated according to the following equations. Note once again the dependence on the length of sliding window 0 ($L_{SW0}$). The processing time of one MAP decoder is given by:

$$T_{MAP} = \frac{K \times \left(N_{iter} \times 1 + (N_{iter} - 1) \times 2\right)}{2F_{TCP3clk}} \times \left(1 + \frac{2L_{SW0}}{K}\right)$$

The DMA setup and transfer time is calculated as

$$T_{DMA\_OVHD} = \frac{108}{F_{CPUclk}}$$

$$T_{DMA\_TRANSFER} = \frac{ceil\left(\frac{3K}{16}\right) + ceil\left(\frac{K}{128}\right)}{F_{DMAclk}}$$

The throughput for two MAP decoders is calculated as:

$$B_{SPLIT} = 2 \times \frac{K}{T_{DMA\_OVHD} + T_{DMA\_TRANSFER} + T_{MAP}}$$

*where:*

$F_{DMAclk} = F_{CPUclk}$ /3, and $F_{TCP3Dclk} = F_{CPUclk}$ /2.

The throughput curves for HSUPA+ are shown in Figure 3-40.

**Figure 3-40 Decoding Throughput for HSUPA+**

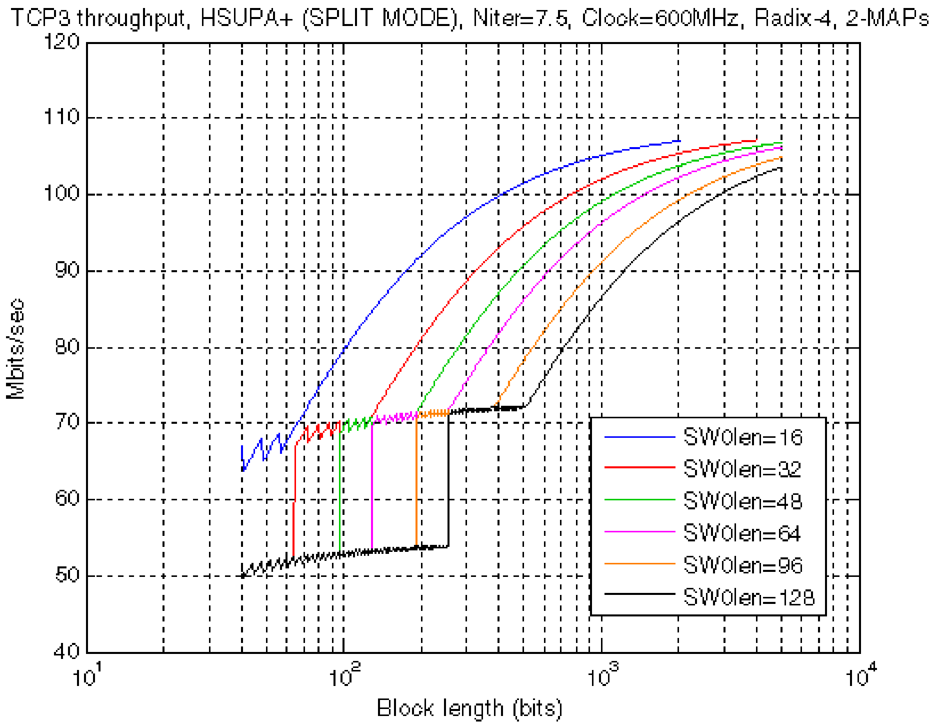


**Figure 3-41 Decoding Throughput for LTE**

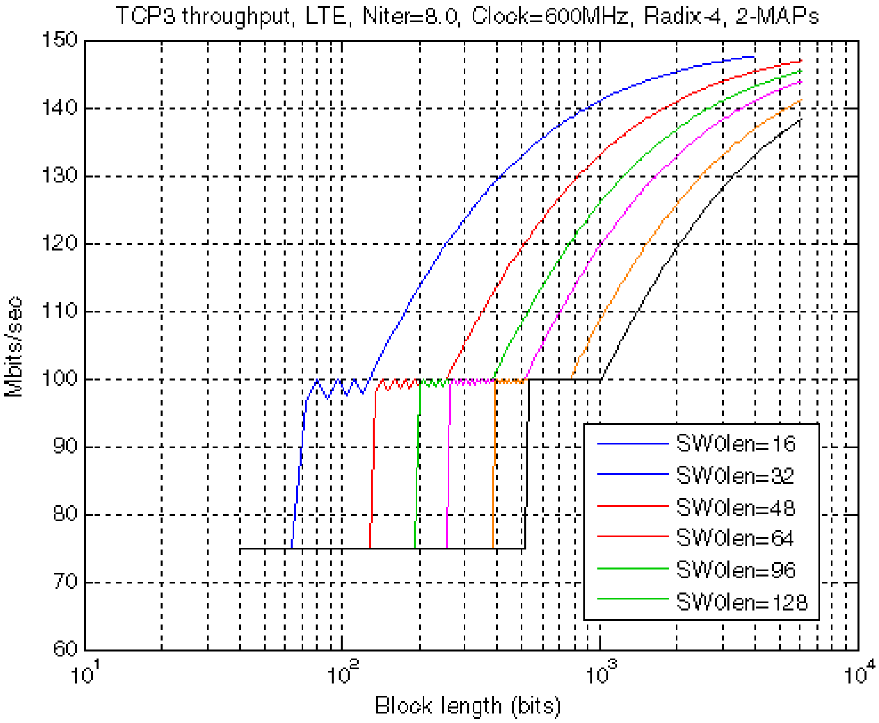
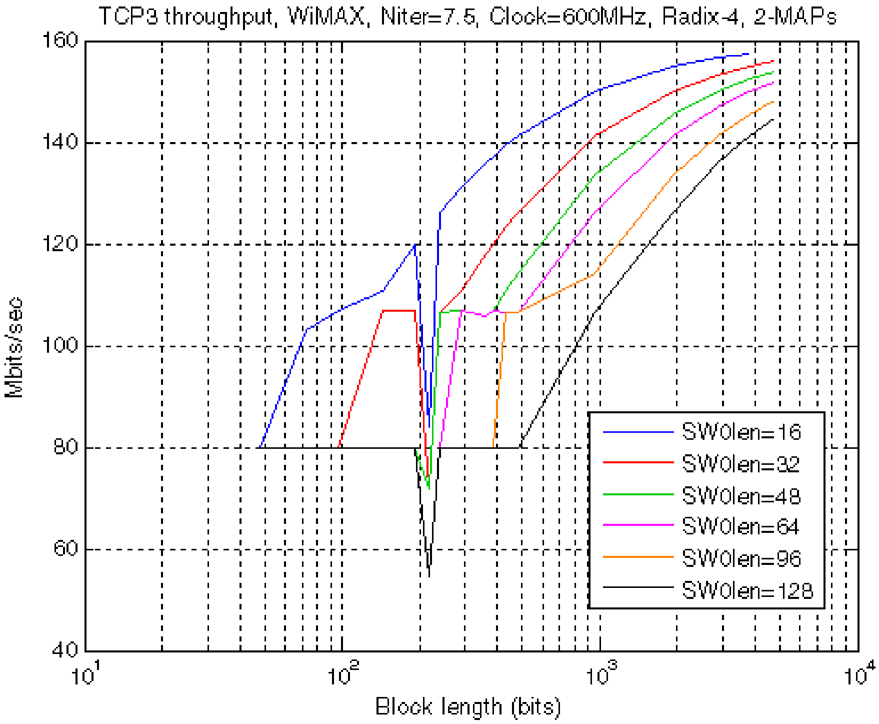


**Figure 3-42 Decoding Throughput for WiMAX**



### 3.5.5.2 Decoding Performance

This section describes the decoding performance of the TCP3D. Because the LTE and 3GPP modes have similar performance, the 3GPP performance is not shown. The first two sections give the performance without rate matching and the third section gives an example of LTE performance with rate matching (the effective code rate is varied from 1/3 up to ~0.97). The figures in the first two sections are plots of the Signal-to-Noise Ratio (SNR) required to hit a specific Frame Error Rate (FER) target vs. code block size. The figure in the third section plots SNR vs. effective code rate. The impact of the length of sliding window 0 (SW0) is displayed in all the figures. Moving from a SW0 length of 16 to 32 gives a very small performance increase. The performance increase by moving beyond a length of 32 is almost negligible.

#### 3.5.5.2.1 LTE Without Rate Matching

- Block sizes: 40 – 6144
- Max-Log-MAP
- Extrinsic Scale = 0.75
- 8 (7½) iterations
- SW0 size = 16, 32, 48, 64, 96, 128 bits
- Code rate K/(3K+12)
- AWGN channel

**Figure 3-43 LTE Decoding Performance for FER=$10^{-1}$**

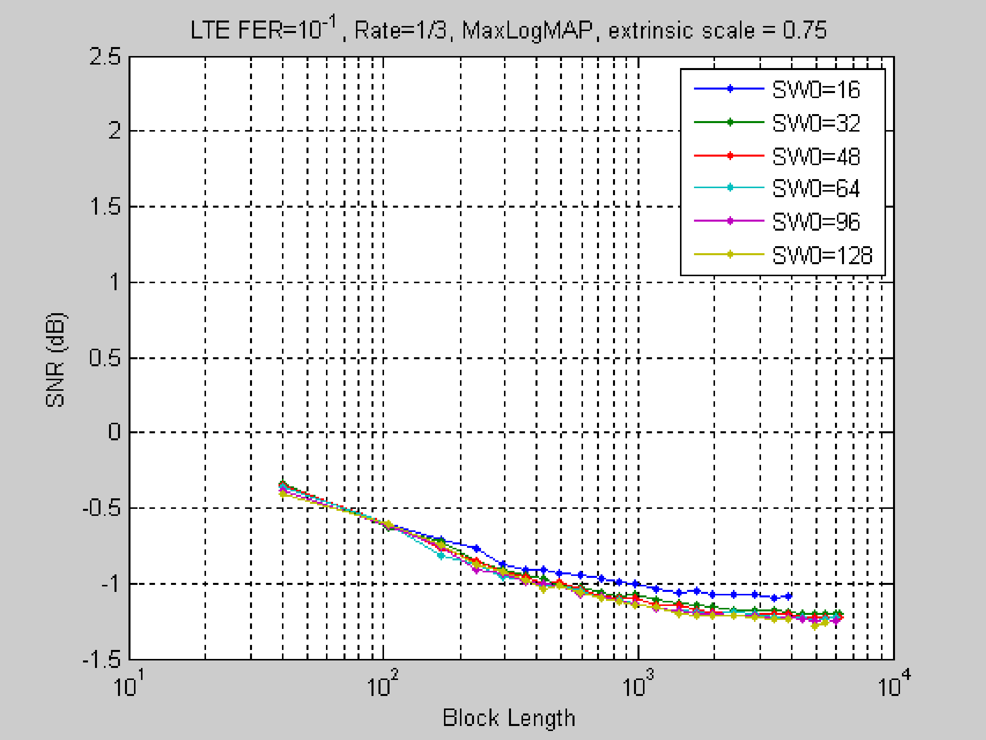


**Figure 3-44 LTE Decoding Performance for FER=$10^{-2}$**

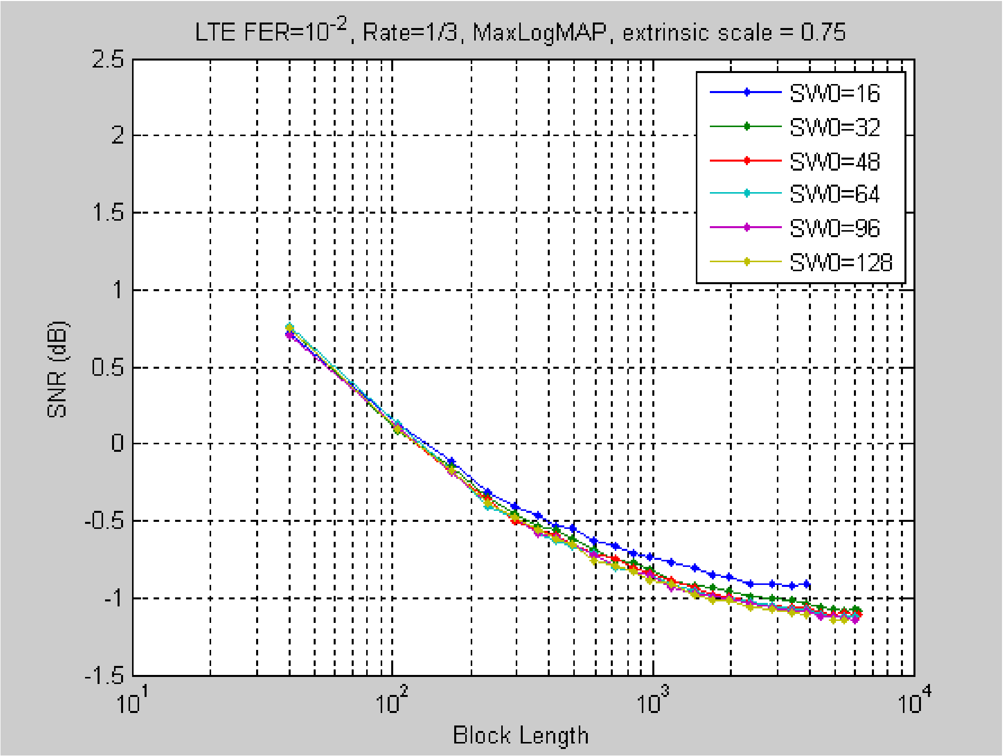


**Figure 3-45 LTE Decoding Performance for FER=$10^{-3}$**

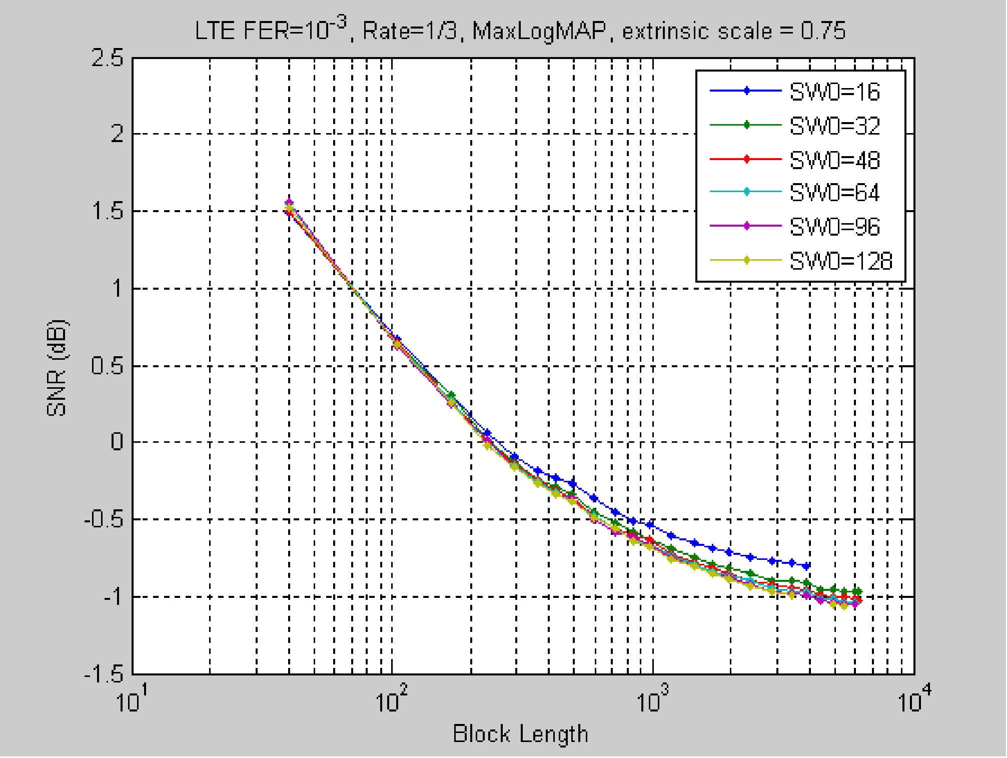
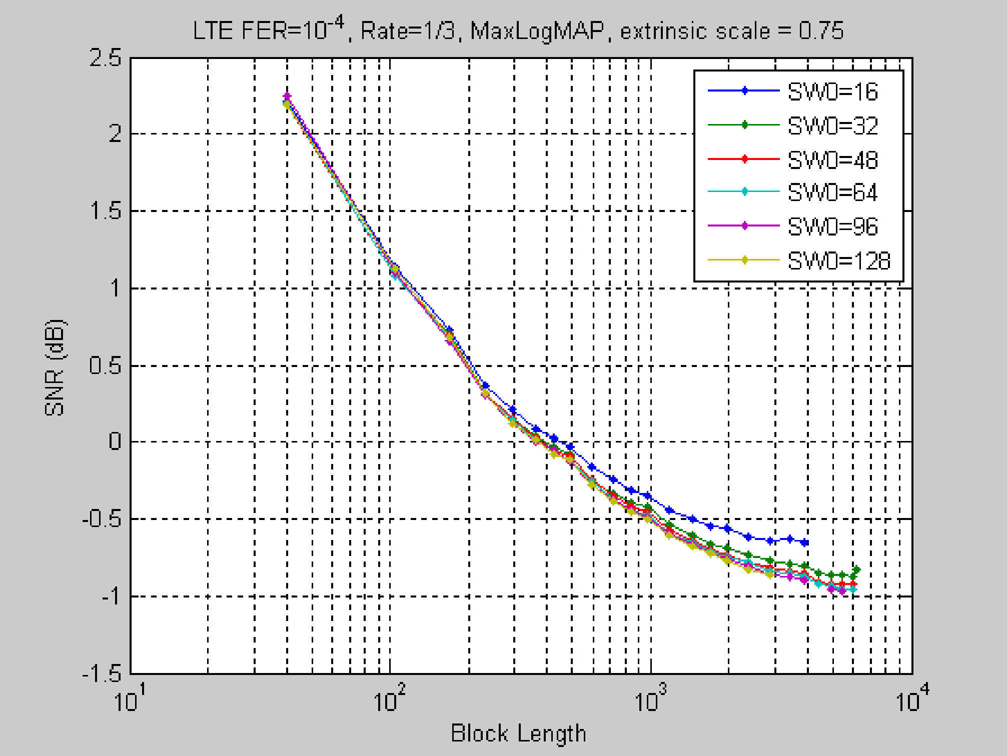


Figure 3-46 LTE Decoding Performance for FER=$10^{-4}$



#### 3.5.5.2.2 WiMAX Without Rate Matching

- Block sizes: 48 – 4800
- Max-Log-MAP
- Extrinsic Scale = 0.75
- 8 (7½) iterations
- SW0 size = 16, 32, 48, 64, 96, 128 bits
- Code rate = 1/3
- AWGN channel

**Figure 3-47 WiMAX Decoding Performance for FER=10-1**

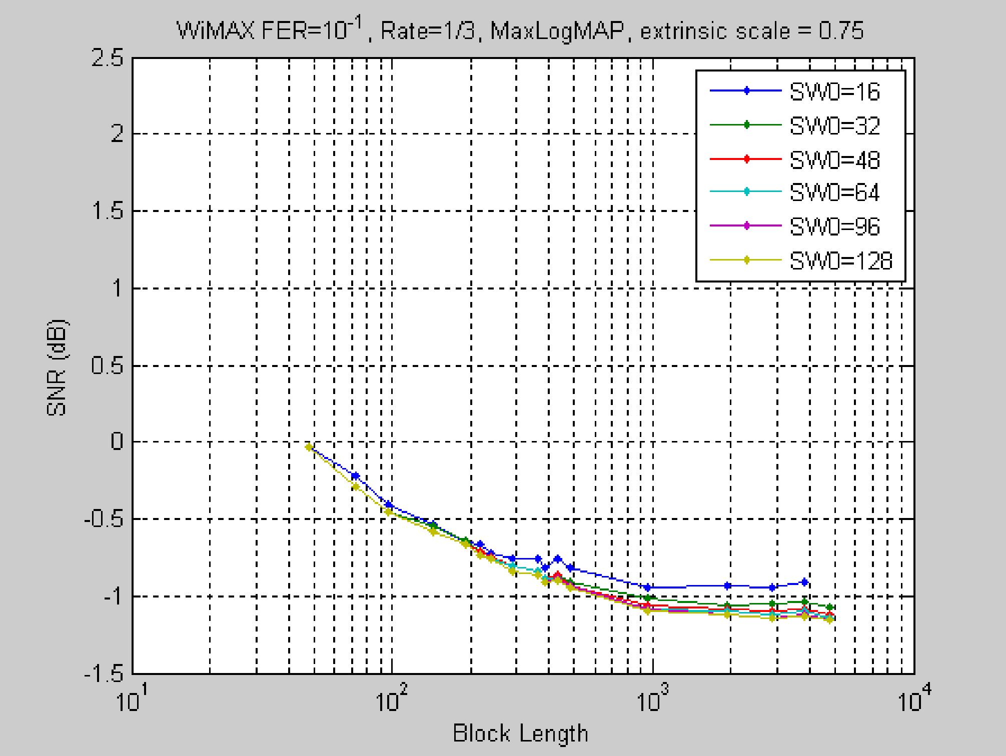


**Figure 3-48 WiMAX Decoding Performance for FER=10-2**

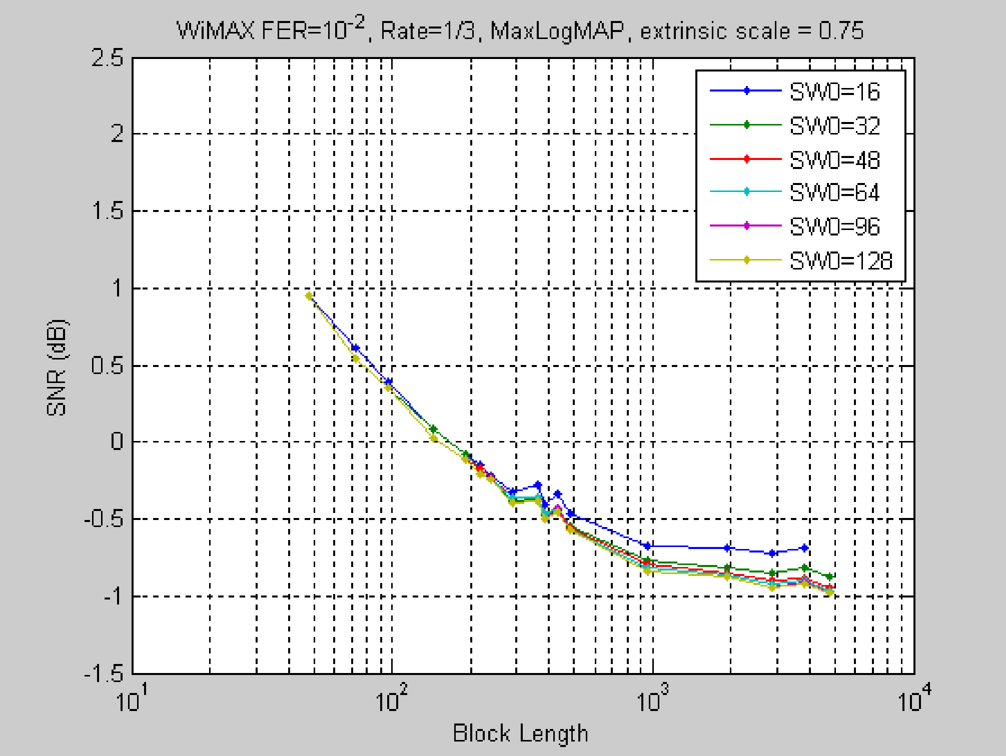


Figure 3-49 WiMAX Decoding Performance for FER=10-3

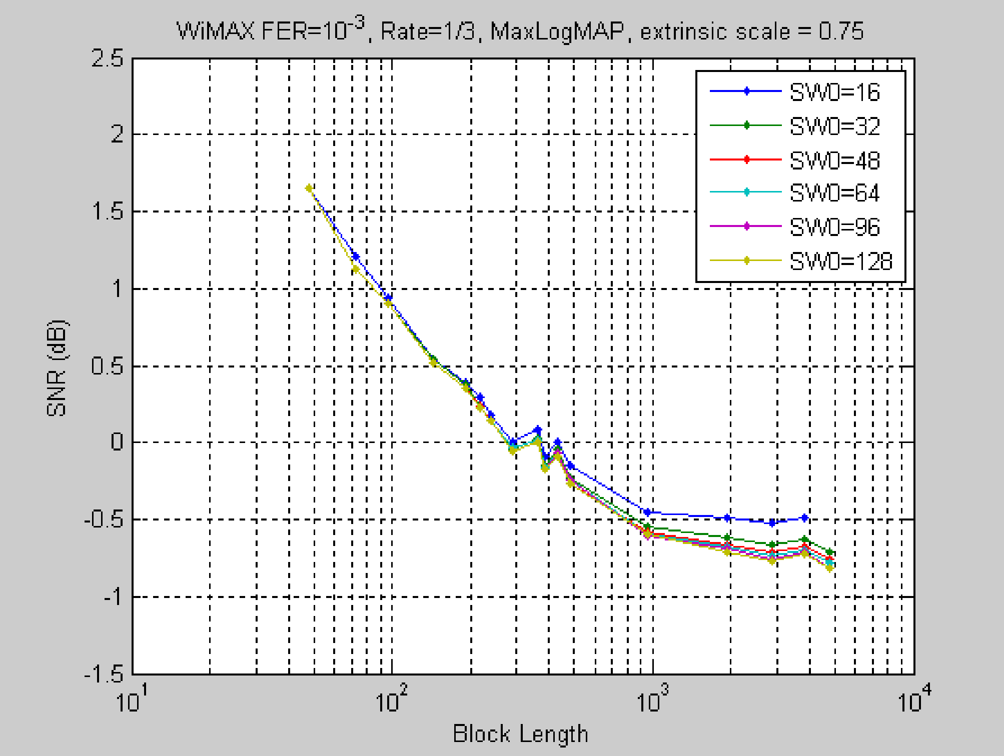
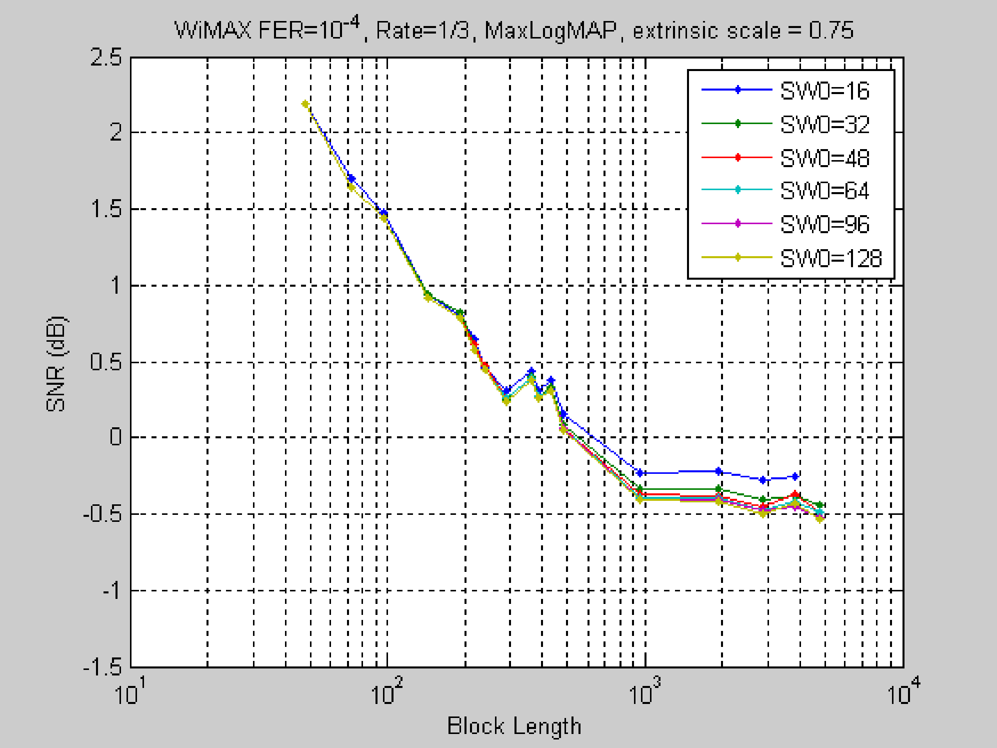


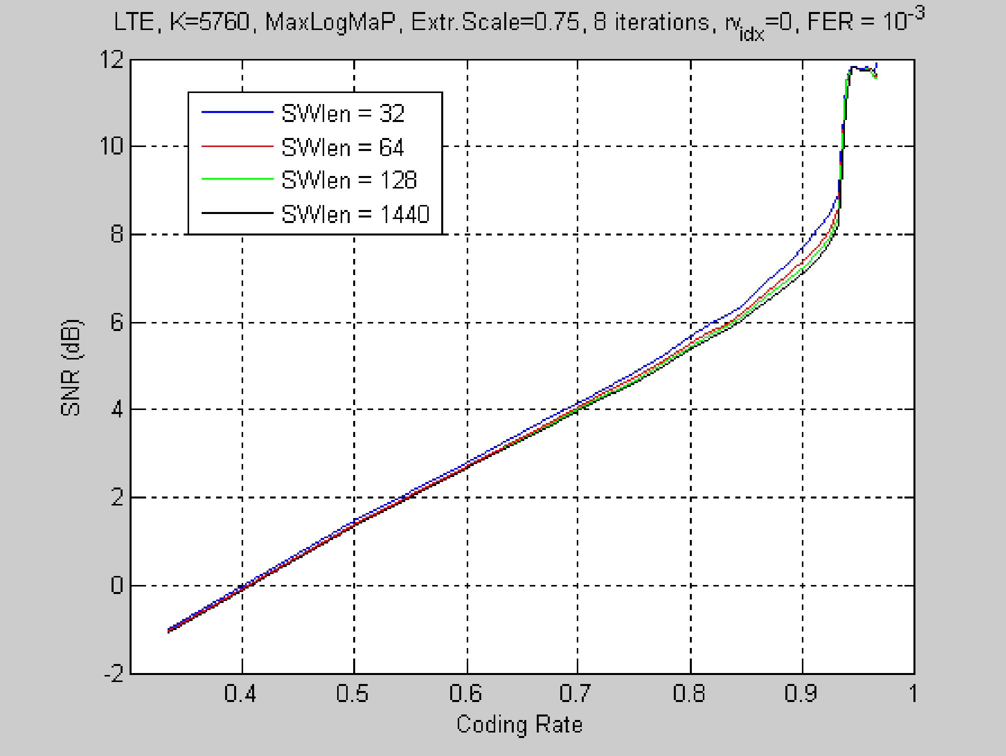
Figure 3-50 WiMAX Decoding Performance for FER=10-4



#### 3.5.5.2.3 LTE Performance with Rate Matching

- LTE, block size K=5760
- Max-Log-MAP
- Extrinsic scaling = 0.75
- SW0 sizes: 32, 64, 128
- Code Rate varied from 1/3 up to ~0.97

**Figure 3-51 LTE Decoding Performance with Rate Matching for FER=10-3**

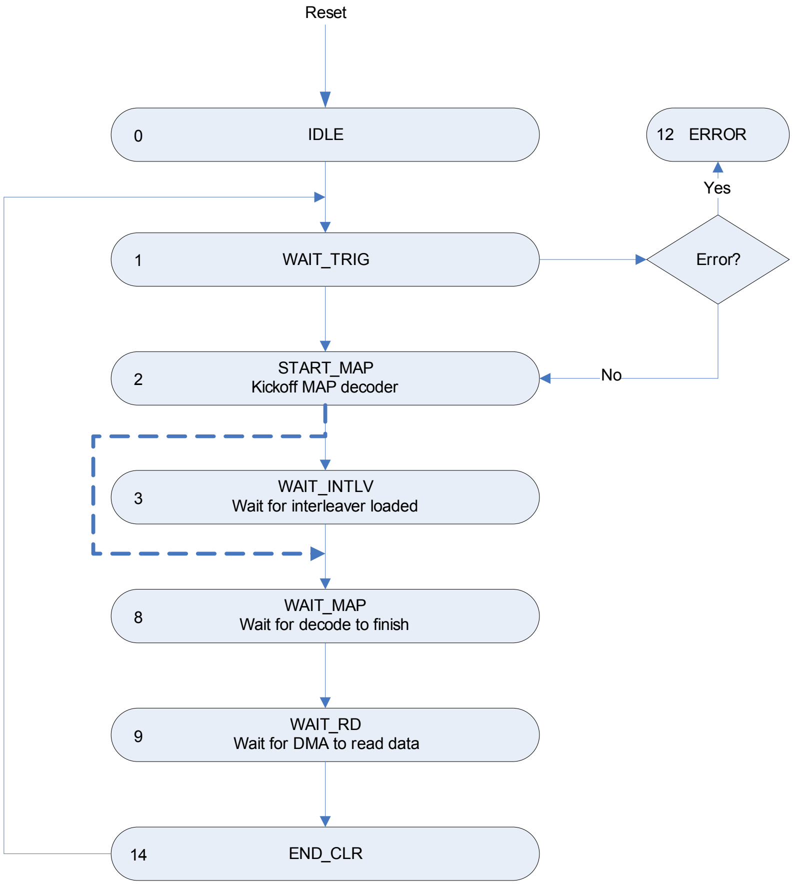


## 3.6 Debug and Emulation Support

### 3.6.1 Debug Mode Support

The TCP3D has a debug mode that allows the user to step through the Interface State Machine (ISM) described in Section 3.5.1. Debug mode is not supported in double-buffer mode. The state diagram for debug mode operation is shown in Figure 3-52.

**Figure 3-52 TCP3D Interface State Machine - Debug Mode**



After a reset or clear, the ISM is in the IDLE state. Writing the following to TCP3D_EXE_P0 will cause the TCP3D to perform the following actions:

4 = Debug_4: Wait in the DEBUG state after all input configuration registers, input data and interleaver indices have been transferred to the TCP3D via DMA.

5 = Debug_5: Execute one full MAP decode iteration and then wait in the DEBUG state.

6 = Debug_6: Execute remaining MAP decode iterations and complete normally. The ISM then loops back to WAIT_TRIG.

**Note—** In split mode, the two *Interface State Machine*s have independent execution registers so they do not need to both be in debug mode at the same time.

The following text describes each command in more detail. The pertinent input/output memories and registers may be examined after each command.

The Debug_4 command causes the state machine to follow its normal sequence through WAIT_TRIG (where it waits for all the inputs to be loaded via DMA), state 2 (which is NOP and does nothing in debug mode), and WAIT_INTLV (optionally). It then continues to DEBUG before stopping to allow the DSP to examine the contents of the memories and input configuration registers that were transferred via DMA. The DSP waits for the *Interface State Machine* to reach the DEBUG state by polling the *TCP_IF_STATE* field in TCP3D_STS_P0.

The Debug_5 command causes the state machine to sequence from DEBUG to WAIT_MAP and then back to DEBUG after performing a single MAP decode iteration. If the Debug_5 command is issued while the state machine is in IDLE or WAIT_TRIG, it essentially bypasses the Debug_4 command. Instead of stopping the first time it reaches DEBUG, it will continue to WAIT_MAP and after one iteration is complete it will branch back to DEBUG and stop. Successive writes of the Debug_5 command will allow successive single-stepping of MAP decoder iterations, but the DSP must wait for the *Interface State Machine* to reach the DEBUG state after each command by polling the *TCP_IF_STATE* field in TCP3D_STS_P0.

The Debug_6 command is used to complete the decoding of the code block normally, presumably after a number of Debug_5 commands or a Debug_4 command. This command causes the state machine to sequence from DEBUG to WAIT_MAP. However, this time it remains in the WAIT_MAP state until the current code block is completely decoded. The state machine then sequences through WAIT_RD (where it waits until all the selected output data and results have been read) and END_CLR before returning to WAIT_TRIG, where it is ready for another code block to decode. The DSP waits for the *Interface State Machine* to reach the WAIT_TRIG state by polling the *TCP_IF_STATE* field in TCP3D_STS_P0.

Table 3-7 summarizes the operation of the *Interface State Machine* in debug mode.

**Table 3-7 TCP3D Debug State Transitions**

| TCP3D_EXE_P0 Value | Interface SM Starting State | Interface SM Ending State | Description |
|---|---|---|---|
| 4 | IDLE | DEBUG | Debug – initialization |
| 5 | DEBUG | DEBUG | Debug – single iteration |
| 6 | DEBUG | WAIT_TRIG | Debug – finish decode |

When the TCP3D is halted in the DEBUG state, the internal counters used for keeping track of the amount of code block input data (i.e. systematic, parity 0 and parity 1) that has been transferred to the TCP3D for auto trigger mode, interleaver table entries loaded, and status flag generation are disabled.   Also, the internal counters that keep track of the number of bytes read from the TCP3D for the hard decisions, soft outputs, and output status registers are disabled. As such, reads of the TCP3D output data while it is halted in debug mode will not cause the logic to believe that any of the output data has been read by the EDMA controller.

## 3.6.2 Emulation Mode Support

### 3.6.2.1 Emulation Overview

The TCP3D emulation interface consists of 2 input signals (*TCP3D_emu_dbgsusp* and *tcp3_emu_dbgsusp_rt*) and the TCP3D Emulation Control register (Section 4.3.4). Also, the *EMU_HALT* bit in the TCP3D Status register is set to a 1 when the TCP3D internal operation has come to a halt in response to an emulation suspend request from the system.

Since the TCP3D is a coprocessor, it can be safely halted and restarted while it is decoding without any loss of data. When halted in emulation mode, the TCP3D will still respond to accesses on both of its VBUS interfaces. As an example, if the TCP3D is halted while the DMA controller is sending a new code block of data, it can continue to receive and store the data, although it will not process it.

During emulation mode it is expected that the DSP will be accessing the TCP3D memories via the 128-bit VBUS_DMA interface. Since the DSP cannot access data as 128-bits in emulation mode, it will use 32-bit accesses to the RAMs which are supported by the VBUS_DMA interface. Big endian data swapping is also supported in emulation mode.

During emulation mode, the internal counters used for keeping track of the amount of code block input data (i.e. systematic, parity 0 and parity 1) that has been transferred to the TCP3D for auto trigger mode, interleaver table entries loaded, and status flag generation are still active. As such, accesses to the TCP3D while it is halted in emulation mode can cause the auto-trigger logic to trigger and the error status bits in TCP3D_EINT_STAT_P0 and TCP3D_EINT_STAT_P1 to indicate that an overflow has occurred (i.e. *DMA_OSYS_ERR*, *DMA_OPAR0_ERR*, *DMA_OPAR1_ERR* and *DMA_OINTLV_ERR*).

During emulation mode, the internal counters that keep track of the number of bytes read from the TCP3D for the hard decisions, soft outputs, and output status registers are disabled so that these reads create no undesirable side effects when emulation mode is over. As such, reads of the TCP3D output data while in emulation mode will not cause the logic to believe that any of the output data has been read by the EDMA controller when it exits emulation mode.

**Note—** Because the TCP3D does not control the transfer of the input data into it, there is no guarantee that the EDMA controller has completed all of its transfers from system memory to the TCP3D when the *EMU_HALT* bit is set. This can be detected via the *PROC_PEND* in **T**CP3D_STS_P0 or TCP3D_STS_P1.

### 3.6.2.2 Halting TCP3D in Emulation Mode

In the descriptions below, unless otherwise noted, all of the register references are to the Process 0 registers that are used for single-buffered mode, double-buffered mode, and split mode process 0.

Halting the TCP3D for emulation mode requires halting the *TCP3D Interface State Machine* (Section 3.5.1). The manner in which the TCP3D internal operation is halted depends on the configuration of the TCP3D_EMU register, as described below. When the TCP3D is halted, the *EMU_HALT* bit in TCP3D_STS_P0 is set to a 1. Figure 3-53 is a copy of Figure 3-39 with the states that can be halted in during emulation mode highlighted. Figure 3-54 is a copy of Figure 3-52 with the states that can be halted in during emulation and debug modes highlighted.

In split mode, both decoders are halted in emulation mode, but since they are totally independent, they will not halt at exactly the same time. Therefore, the DSP should check the *EMU_HALT* bit in both TCP3D_STS_P0 and TCP3D_STS_P1 before accessing the TCP3D.

### Hard Stop

When the *SOFT* bit in TCP3D_EMU_P0 is 0, a “hard” stop is performed when the TCP3D is halted in emulation mode. In this case, the TCP3D is required to halt as soon as it can. As such, there is no guarantee that the TCP3D will complete any decodes in progress before halting. If the TCP3D is waiting for the next code block to arrive, it is halted immediately in IDLE or WAIT_TRIG of the *Interface State Machine*. If the TCP3D is actively decoding a block, it is halted at the end of the current MAP decode iteration in WAIT_MAP of the *TCP3D Interface State Machine* (Figure 3-53). The TCP3D will not issue any new EDMA events (REVT0, REVT1) while in emulation mode and configured for a “hard” stop once it is halted (There can be a race between emulation mode and an EDMA event before the TCP3D halts);

The TCP3D will restart from the halt state when the emulation suspend signal is negated and resume its normal operation (e.g. any decodes in progress will successfully complete; any blocks waiting to be decoded will be processed).

When the *Interface State Machine* is in debug mode (Section 3.6.1) and the *EXE_CMD* in TCP3D_EXE_P0 is Debug_5, a “hard” stop will not halt in WAIT_MAP, as described above (see Figure 3-54). Instead, it will branch back to DEBUG after the current MAP decode iteration has finished. This allows the DSP to retain debug control over the MAP decoding process. Other than this exception, the emulation halt states remain the same for debug mode. The debug commands can still cause the state machine to stop in DEBUG, but the state machine is not actually halted (i.e. DSP can control the MAP decoding process via debug commands).

**Note—Note:** Issuing a DEBUG_6 command while in emulation mode will cause the MAP decoder to complete the decoding of the current block and the *Interface State Machine* will halt in WAIT_MAP.

### Soft Stop

When the *SOFT* bit in TCP3D_EMU is 1, a “soft” stop is performed when the TCP3D is halted in emulation mode. In this case, the TCP3D is allowed to halt gracefully. Any decodes in progress are allowed to complete and the results are DMAed to system memory. The TCP3D will halt in WAIT_TRIG of the *Interface State Machine* when it is active when the emulation suspend request occurs. If the TCP3D is waiting for the next code block to arrive, it is halted immediately in IDLE or WAIT_TRIG of the *Interface State Machine*. The TCP3D will restart from the halt state when the emulation suspend signal is negated and resume its normal operation (e.g. any new blocks received or blocks waiting to be decoded will be processed).

When the TCP3D *Interface State Machine* is in debug mode, the “soft” stop emulation halt states remain the same (see Figure 3-54). However, the debug commands can cause the state machine to stop in DEBUG, but the state machine is not actually halted (i.e. DSP can control the MAP decoding process via debug commands).

### Error Interrupt Suppression

TCP3D interrupts are suppressed while in emulation mode for either a “hard” or “soft” stop. Any pending interrupts will be generated when the TCP3D exits emulation mode (see Section 3.5.4.3).

**Figure 3-53 TCP3D Interface State Machine – Normal Mode with Emulation Halt State**

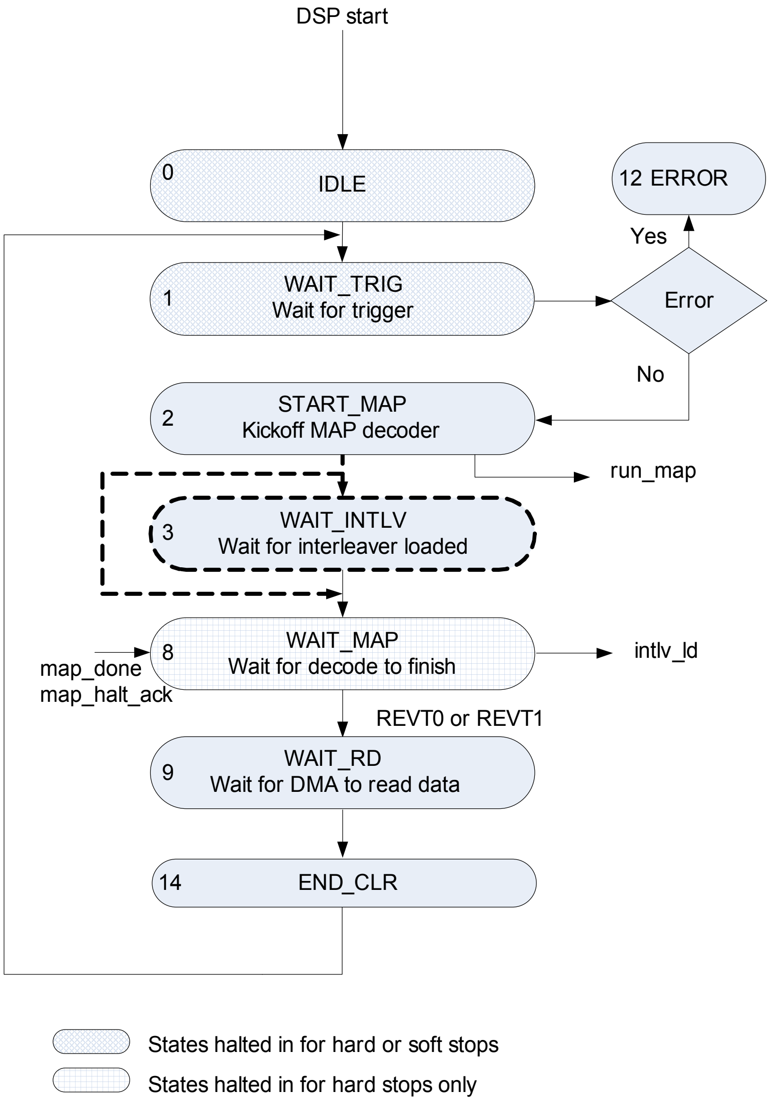
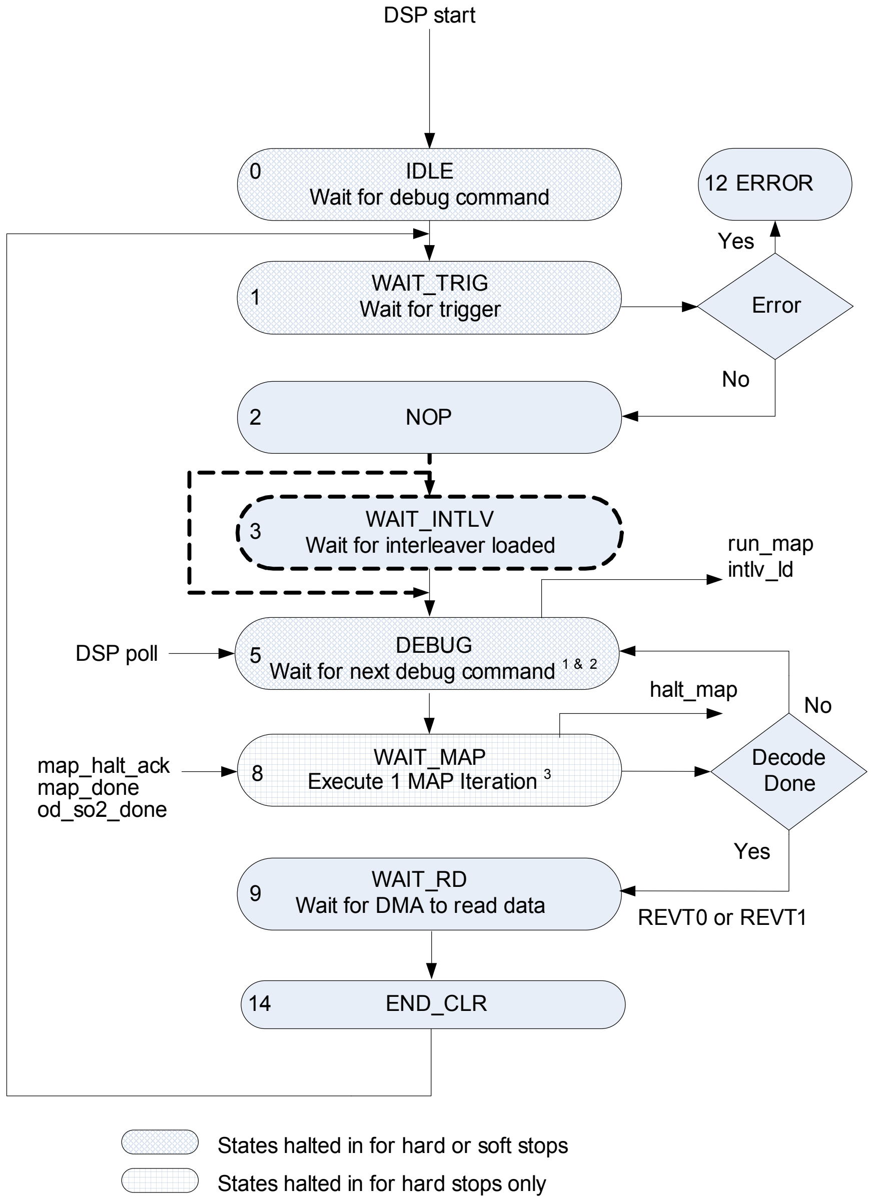

DSP start
0 IDLE
12 ERROR
Yes
1 WAIT_TRIG Wait for trigger
Error
No
2 START_MAP Kickoff MAP decoder
run_map
3 WAIT_INTLV Wait for interleaver loaded
map_done
map_halt_ack
8 WAIT_MAP Wait for decode to finish
intlv_ld
REVT0 or REVT1
9 WAIT_RD Wait for DMA to read data
14 END_CLR
States halted in for hard or soft stops
States halted in for hard stops only

**Figure 3-54 Interface State Machine – Debug Mode with Emulation Halt State**


DSP start
0 IDLE Wait for debug command
12 ERROR
Yes
1 WAIT_TRIG Wait for trigger
Error
No
2 NOP
3 WAIT_INTLV Wait for interleaver loaded
run_map
intlv_ld
DSP poll
5 DEBUG Wait for next debug command 1 & 2
halt_map
No
map_halt_ack
map_done
od_so2_done
8 WAIT_MAP Execute 1 MAP Iteration 3
Decode Done
Yes
9 WAIT_RD Wait for DMA to read data
REVT0 or REVT1
14 END_CLR
States halted in for hard or soft stops
States halted in for hard stops only

Notes

1. For LTE, on the last iteration, **map_halt_ack** occurs prior to **map_done**, so the transition from DEBUG to WAIT occurs when **map_done** is asserted without requiring that an additional DEBUG_5 command be issued.

2. If the command is DEBUG_4 or DEBUG_5, the state machine halts in DEBUG state until MAP decode is finished. If the initial command is DEBUG_5, the state machine does not halt in DEBUG state for the first iteration. Instead, it passes through DEBUG into WAIT_MAP.

3. If the command is DEBUG_6 or MAP decode is finished, the state machine halts in WAIT_MAP.

#### 3.6.2.3 Debug Transaction Considerations (EMUDBG)

Certain features of software debuggers, like the “watch windows” found in Code Composer Studio, may result in accesses to TCP3D RAMs while the TCP3D is actively decoding a block. These accesses are considered “in progress” accesses as described in Section 3.6.3and as such, features like “watch windows” are not supported since the data read from the TCP3D would not be correct. Also, reads of the output/decision and soft output RAMs and output status registers can corrupt the operation of the TCP3D as discussed in Section 3.6.3.

The mechanism used to distinguish debugger transactions on the VBUSP is the *emudbg* signal. It is not supported by the TCP3D.

### 3.6.3 Memory Access Restrictions

The systematic, parity 0, parity 1 and interleaver memories are written by the system when a new code block of data is loaded into the TCP3D. However, once the trigger for the process associated with these RAMs is set, any subsequent accesses are considered “in progress” accesses that will be internally ignored by the TCP3D. The interleaver RAM is a special case since it can be loaded after the trigger is set. For it, an “in progress” access is any access after the *TCP3D Interface State Machine* leaves WAIT_INTLV. These “in progress” accesses are flagged by setting the *IPROG_ACC_ERR* bit in TCP3D_EINT_STAT_P0 or TCP3D_EINT_STAT_P1.

The output/decision and soft output RAMs are normally read by the system when the decoding is complete and after the TCP3D has issued a read event to the EDMA3 controller. Any accesses before the receive event is issued are considered “in progress” accesses and may potentially corrupt the operation of the TCP3D. These accesses will cause the *IPROG_ACC_ERR* bit in TCP3D_EINT_STAT_P0 or TCP3D_EINT_STAT_P1 to be set. Accesses to RAM are allowed when the TCP3D is halted in either debug or emulation mode, and the *IPROG_ACC_ERR* bit will not be set in these modes.

For all “in progress” accesses, the VBUS_DMA read or write transaction is acknowledged, but the access is blocked internally. For writes, the write request is ignored, and for reads, all 0s are returned as the read data.

**Note—** *In progress* reads of the output/decision and soft output RAMs may corrupt the operation of the TCP3D. These reads will cause the internal counters that keep track of the number of bytes read from the TCP3D for the hard decisions and soft outputs to increment. This will cause the TCP3D to prematurely believe that all of the output data has been read by the EDMA controller, and it will start decoding the next block too soon. In this case, there is a chance that the data in these RAMs could be overwritten before it is read by the EDMA3. This issue also exists for “in progress” reads of the output status registers which are not reported by the *IPROG_ACC_ERR*.

Chapter 4

# Registers

The beginning of this chapter should include a brief description of the peripheral registers and a table listing each of the registers with a pointer to the information for each one.

## 4.1 Registers and Memories

The range of defined addresses for the VBUS_CFG and VBUS_DMA interfaces is shown below. Within each range are many unsupported address locations. Each interface is placed at a unique base address in the global address map.

**Table 4-1 VBUS Interface Address Ranges**

| VBUS Interface | Defined Address Range |
|---|---|
| VBUS CFG | 0x0000 – 0x0250 |
| VBUS DMA | 0x0_0000 – 0x7_1FFF |
| **End of Table 4-1** | |

**Note—**To support the double-buffered and split modes, some registers are duplicated so that each process has its own register. These are designated with the suffix P0 for process 0 and P1 for process 1.

### 4.1.1 Memory Maps

Table 4-2 through Table 4-5 list the main control and status registers, the control and status registers for Process 0 and process 1, and the memory-mapped RAM registers.

**Table 4-2 Control and Status Registers (VBUS_CFG)**

| VBUS_CFG Offset Address | RD / WR | Register Name | Function |
|---|---|---|---|
| 0x0000 | R | TCP3D_PID | TCP3D Peripheral Identification Register |
| 0x0004 | R/W | TCP3D_MODE | TCP3D Mode Register |
| 0x0008 | R/W | TCP3D_END | TCP3D Endianess Register |
| 0x000C | R/W | TCP3D_EMU | TCP3D Emulation Register |
| 0x0010 | R/W | TCP3D_EOI | TCP3D End of Interrupt Register |
| 0x0014 | R/W | TCP3D_SOFT_RESET | TCP3D Soft Reset Register |
| **Process 0 Registers** | | | |
| 0x0100 | R/W | TCP3D_EXE_P0 | TCP3D Process 0 Execution Register |
| 0x0104 | R | TCP3D_STS_P0 | TCP3D Process 0 Status Register |
| 0x0108 | W | TCP3D_EV_SET_P0 | TCP3D Process 0 Event Set Register |
| 0x0140 | R/W | TCP3D_EINT_STAT_P0 | TCP3D Process 0 Error Interrupt Raw Status and Set Register |
| 0x0144 | W | TCP3D_EINT_CLR_P0 | TCP3D Process 0 Error Interrupt Clear Register |
| 0x0148 | R | TCP3D_EINT_EN_STAT_P0 | TCP3D Process 0 Error Interrupt Enabled Status Register |
| 0x014C | R/W | TCP3D_EINT_EN_P0 | TCP3D Process 0 Error Interrupt Enable and Set Register |
| 0x0150 | W | TCP3D_EINT_EN_CLR_P0 | TCP3D Process 0 Error Interrupt Enable Clear Register |
| **Process 1 Registers** | | | |
| 0x0200 | R/W | TCP3D_EXE_P1 | TCP3D Process 1 Execution Register |
| 0x0204 | R | TCP3D_STS_P1 | TCP3D Process 1 Status Register |
| 0x0208 | W | TCP3D_EV_SET_P1 | TCP3D Process 1 Event Set Register |
| 0x0240 | R/W | TCP3D_EINT_STAT_P1 | TCP3D Process 1 Error Interrupt Raw Status and Set Register |
| 0x0244 | W | TCP3D_EINT_CLR_P1 | TCP3D Process 1 Error Interrupt Clear Register |
| 0x0248 | R | TCP3D_EINT_EN_STAT_P1 | TCP3D Process 1 Error Interrupt Enabled Status Register |
| 0x024C | R/W | TCP3D_EINT_EN_P1 | TCP3D Process 1 Error Interrupt Enable and Set Register |
| 0x0250 | W | TCP3D_EINT_EN_CLR_P1 | TCP3D Process 1 Error Interrupt Enable Clear Register |
| **End of Table 4-2** | | | |

**Table 4-3 Process 0 Control and Status Registers (VBUS_DMA)**

| VBUS_DMA Offset Address | RD / WR | Register Name | Function |
|---|---|---|---|
| 0x0_0000 | R/W | TCP3D_IC_CFG0_P0 | TCP3D Process 0 Input Configuration Register 0 |
| 0x0_0004 | R/W | TCP3D_IC_CFG1_P0 | TCP3D Process 0 Input Configuration Register 1 |
| 0x0_0008 | R/W | TCP3D_IC_CFG2_P0 | TCP3D Process 0 Input Configuration Register 2 |
| 0x0_000C | R/W | TCP3D_IC_CFG3_P0 | TCP3D Process 0 Input Configuration Register 3 |
| 0x0_0010 | R/W | TCP3D_IC_CFG4_P0 | TCP3D Process 0 Input Configuration Register 4 |
| 0x0_0014 | R/W | TCP3D_IC_CFG5_P0 | TCP3D Process 0 Input Configuration Register 5 |
| 0x0_0018 | R/W | TCP3D_IC_CFG6_P0 | TCP3D Process 0 Input Configuration Register 6 |
| 0x0_001C | R/W | TCP3D_IC_CFG7_P0 | TCP3D Process 0 Input Configuration Register 7 |
| 0x0_0020 | R/W | TCP3D_IC_CFG8_P0 | TCP3D Process 0 Input Configuration Register 8 |
| 0x0_0024 | R/W | TCP3D_IC_CFG9_P0 | TCP3D Process 0 Input Configuration Register 9 |
| 0x0_0028 | R/W | TCP3D_IC_CFG10_P0 | TCP3D Process 0 Input Configuration Register 10 |
| 0x0_002C | R/W | TCP3D_IC_CFG11_P0 | TCP3D Process 0 Input Configuration Register 11 |
| 0x0_0030 | R/W | TCP3D_IC_CFG12_P0 | TCP3D Process 0 Input Configuration Register 12 |
| 0x0_0034 | R/W | TCP3D_IC_CFG13_P0 | TCP3D Process 0 Input Configuration Register 13 |
| 0x0_0038 | R/W | TCP3D_IC_CFG14_P0 | TCP3D Process 0 Input Configuration Register 14 |
| 0x0_003C | W | TCP3D_TRIG_P0 | TCP3D Process 0 Trigger Register |
| 0x0_0100 | R | TCP3D_OUT_STS0_P0 | TCP3D Process 0 Output Status Register 0 |
| 0x0_0104 | R | TCP3D_OUT_ STS1_P0 | TCP3D Process 0 Output Status Register 1 |
| 0x0_0108 | R | TCP3D_OUT_ STS2_P0 | TCP3D Process 0Output Status Register 2 |
| **End of Table 4-3** | | | |

Table 4-4 Process 1 Control and Status Registers (VBUS_DMA)

| VBUS_DMA Offset Address | RD / WR | Register Name | Function |
|---|---|---|---|
| 0x0_0200 | R/W | TCP3D_IC_CFG0_P1 | TCP3D Process 1 Input Configuration Register 0 |
| 0x0_0204 | R/W | TCP3D_IC_CFG1_P1 | TCP3D Process 1 Input Configuration Register 1 |
| 0x0_0208 | R/W | TCP3D_IC_CFG2_P1 | TCP3D Process 1 Input Configuration Register 2 |
| 0x0_020C | R/W | TCP3D_IC_CFG3_P1 | TCP3D Process 1 Input Configuration Register 3 |
| 0x0_0210 | R/W | TCP3D_IC_CFG4_P1 | TCP3D Process 1 Input Configuration Register 4 |
| 0x0_0214 | R/W | TCP3D_IC_CFG5_P1 | TCP3D Process 1 Input Configuration Register 5 |
| 0x0_0218 | R/W | TCP3D_IC_CFG6_P1 | TCP3D Process 1 Input Configuration Register 6 |
| 0x0_021C | R/W | TCP3D_IC_CFG7_P1 | TCP3D Process 1 Input Configuration Register 7 |
| 0x0_0220 | R/W | TCP3D_IC_CFG8_P1 | TCP3D Process 1 Input Configuration Register 8 |
| 0x0_0224 | R/W | TCP3D_IC_CFG9_P1 | TCP3D Process 1 Input Configuration Register 9 |
| 0x0_0228 | R/W | TCP3D_IC_CFG10_P1 | TCP3D Process 1 Input Configuration Register 10 |
| 0x0_022C | R/W | TCP3D_IC_CFG11_P1 | TCP3D Process 1 Input Configuration Register 11 |
| 0x0_0230 | R/W | TCP3D_IC_CFG12_P1 | TCP3D Process 1 Input Configuration Register 12 |
| 0x0_0234 | R/W | TCP3D_IC_CFG13_P1 | TCP3D Process 1 Input Configuration Register 13 |
| 0x0_0238 | R/W | TCP3D_IC_CFG14_P1 | TCP3D Process 1 Input Configuration Register 14 |
| 0x0_023C | W | TCP3D_TRIG_P1 | TCP3D Process 1 Trigger Register |
| 0x0_0300 | R | TCP3D_OUT_STS0_P1 | TCP3D Process 1 Output Status Register 0 |
| 0x0_0304 | R | TCP3D_OUT_ STS1_P1 | TCP3D Process 1 Output Status Register 1 |
| 0x0_0308 | R | TCP3D_OUT_ STS2_P1 | TCP3D Process 1 Output Status Register 2 |
| **End of Table 4-4** | | | |

Table 4-5 Memory Mapped RAM (VBUS_DMA)

| VBUS_DMA Offset Address | RD / WR | Name | Address Range | Length (bytes) |
|---|---|---|---|---|
| 0x1_0000 | R/W | Systematic Data Memory 0 | 0x1_0000 - 0x1_1FFF | 8,192 |
| 0x1_2000 | R/W | Parity 0 Data Memory 0 | 0x1_2000 - 0x1_3FFF | 8,192 |
| 0x1_4000 | R/W | Parity 1 Data Memory 0 | 0x1_4000 - 0x1_5FFF | 8,192 |
| 0x1_6000 | R/W | Systematic Data Memory 1 | 0x1_6000 - 0x1_7FFF | 8,192 |
| 0x1_8000 | R/W | Parity 0 Data Memory 1 | 0x1_8000 - 0x1_9FFF | 8,192 |
| 0x1_A000 | R/W | Parity 1 Data Memory 1 | 0x1_A000 - 0x1_BFFF | 8,192 |
| 0x2_0000 | R/W | Interleaver Memory 0 | 0x2_0000 - 0x2_3FFF | 16,384 |
| 0x2_4000 | R/W | Interleaver Memory 1 | 0x2_4000 - 0x2_7FFF | 16,384 |
| 0x3_0000 | RO | Output/Decision Memory 0 | 0x3_0000 - 0x3_03FF | 1,024 |
| 0x3_1000 | RO | Output/Decision Memory 1 | 0x3_1000 - 0x3_13FF | 1,024 |
| 0x8_0000 | RO | Systematic Soft Output Memory | 0x8_0000 - 0x8_1FFF | 8,192 |
| 0x8_2000 | RO | Parity 0 Soft Output Memory[1] | 0x8_2000 - 0x8_3FFF | 8,192 |
| 0x8_4000 | RO | Parity 1[2] Soft Output Memory | 0x8_4000 - 0x8_5FFF | 8,192 |
| **End of Table 4-5** | | | | |

[1] This memory stores the Process 1 systematic soft outputs for SPLIT mode.

[2] This memory is not used in SPLIT mode.

## 4.2 Constant Registers (VBUS CFG Bus)

The only register with a constant, read-only value is the TCP3D_PID register.

### 4.2.1 TCP3D Peripheral ID Register (TCP3D_PID)

Figure 4-1 shows the TCP3D Peripheral ID register layout and Table 4-6 describes the bits.

**Figure 4-1 Peripheral ID Register**

| 31 30 | 29 28 | 27 16 | 15 11 | 10 8 | 7 6 | 5 0 |
|---|---|---|---|---|---|---|
| SCHEME | Reserved | FUNCTION | RTL | MAJOR | CUSTOM | MINOR |
| R-X | R-X | R-X | R-X | R-X | R-X | R-X |

Legend: R = Read only; -x, value is indeterminate — see the device-specific data manual

**Table 4-6 Peripheral ID Register Field Descriptions**

| Bit | Name | Value | Description |
|---|---|---|---|
| 31-30 | SCHEME | 01 | Current scheme |
| 29-28 | Reserved | 00 | Reserved |
| 27-16 | FUNCTION | 0x804 | Function code assigned to TCP3D |
| 15-11 | RTL | RLS | RTL Version (R) code |
| 10-8 | MAJOR | RLS | Major revision (X) code |
| 7-6 | CUSTOM | RLS | Custom version code |
| 5-0 | MINOR | RLS | Minor revision (Y) code |
| **End of Table 4-6** | | | |

The peripheral identification register (TCP3D_PID) is a constant register that contains the ID and ID revision number for that peripheral. The TCP3D_PID stores version information used to identify the peripheral. All bits within this register are read-only (writes have no effect) meaning that the values within this register are hard-coded with the appropriate values and do not change from their reset state. The lower 16-bits of TCP3D_PID are the revision ID of the TCP3D. Since this is a dynamic value based upon the version number of the RTL, it is not kept up to date here. Instead, please refer to the release notes for the TCP releases.

## 4.3 Control and Status Registers (VBUS_CFG Bus)

To support split mode, some registers are duplicated so that each process has its own register. These are designated with the suffix “P0” for process 0 and “P1” for process 1. The register map is the same for both and only the address changes. In the diagrams below, only one register map table is shown for both registers.

### 4.3.1 TCP3D Soft Reset Register (TCP3D_SOFT_RESET)

Figure 4-2 shows the TCP3D Soft Reset register layout and Table 4-7 describes the bits.

**Figure 4-2 TCP3D Soft Reset Register (reset value 0x0000)**

| 31 1 | 0 |
|---|---|
| Reserved | SOFT_RESET |
| R-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -n = value after reset

**Table 4-7 TCP3D Soft Reset Register Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31-1 | Reserved | Reserved (always reads 0) |
| 0 | SOFT_RESET | TCP3D soft reset control:<br>0 = Negate internal soft reset<br>1 = Assert internal soft reset |
| **End of Table 4-7** | | |

### 4.3.2 TCP3D Mode Control Register (TCP3D_MODE)

Figure 4-3 shows the TCP3D Mode Control register layout and Table 4-8 describes the bits.

**Figure 4-3 TCP3D Mode Control Register (reset value 0x0000)**

| 31 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 0 |
|---|---|---|---|---|---|---|---|
| Reserved | LTE_CRC_ISEL | AUTO_TRIG_EN | ERR_IGNORE_EN | ITG_EN | IN_MEM_DB_EN | Reserved | MODE_SEL |
| R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -n = value after reset

**Table 4-8 TCP3D Mode Control Register Field Descriptions (Part 1 of 2)**

| Bit | Name | Description |
|---|---|---|
| 31-8 | Reserved | Reserved (always reads 0) |
| 7 | LTE_CRC_ISEL | Select initial value used for LTE CRC calculation:<br>0 = Use 0x00_0000 as initial value (e.g. LTE)<br>1 = Use 0xFF_FFFF as initial value. |
| 6 | AUTO_TRIG_EN | Enable feature where TCP3D is auto triggered when all of the systematic, parity 0 and parity 1 data for the code block has sent to the TCP3D. This eliminates the requirement of triggering the TCP3D by writing to TCP3D_TRIG_P0 or TCP3D_TRIG_P1.<br>0 = Disable auto trigger mode<br>1 = Enable auto trigger mode |
| 5 | ERR_IGNORE_EN | Enable feature where TCP3D only reports errors, but does not stop when an error is detected.<br>0 = Stop TCP3D on enabled errors<br>1 = Do not stop TCP3D on enabled errors |

**Table 4-8 TCP3D Mode Control Register Field Descriptions (Part 2 of 2)**

| Bit | Name | Description |
|---|---|---|
| 4 | ITG_EN | Enable internal interleaver table generator for mode selected by *MODE_SEL:*<br>0 = Internal generation of interleaver table disabled (i.e. entered via DMA)<br>1 = Internal generation of interleaver table enabled when INTER_LOAD_SEL bit in TCP3D_IC_CFG2_P0 is set to a 1 |
| 3 | IN_MEM_DB_EN | Input memory double-buffer enable (i.e. ping pong):<br>**NOTE:** Not available with split mode.<br>0 = Double-buffering disabled<br>1 = Double-buffering enabled |
| 2 | Reserved | Reserved (always reads 0) |
| 1:0 | MODE_SEL | TCP3D mode i.e., binary/duo-binary, etc.<br>0 = 3GPP, single MAP, binary mode<br>1 = LTE, dual MAP, binary mode<br>2 = WiMAX, dual MAP, duo-binary mode<br>3 = HSUPA+, split decoder, binary mode<br>**NOTES:**<br>• Split mode only allowed if IN_MEM_DB_EN = 0.<br>• The TCP3D must be reset (either external reset or soft reset) before changing the mode of the TCP3D via MODE_SEL. |
| **End of Table 4-8** | | |

**Note—**Two MAP decoders are used at full speed for all non-contentious, WiMAX interleavers. When contentious interleaver, WiMAX code block size 27 byte is detected, the TCP3D reverts into a slower, contentious avoidance mode using two MAP decoders.

### 4.3.3 TCP3D Endianess Control Register (TCP3D_END)

Figure 4-4 shows the TCP3D Endianess Control register layout and Table 4-9 describes the bits.

**Figure 4-4 TCP3D Endianess Control Register (reset value 0x0000)**

| 31 3 | 2 | 1 0 |
|---|---|---|
| Reserved | ENDIAN_INDATA | ENDIAN_INTR |
| R-0 | R/W-0 | R/W--0 |

Legend: R = Read only; R/W = Read/Write; -n = value after reset

**Table 4-9 TCP3D Endianess Control Register Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31:2 | Reserved | Reserved (always reads 0) |
| 1 | ENDIAN_INDATA | DSP format of systematic and parity input data in big endian mode. This bit has no effect if in little endian mode.<br>DSP TCP3D<br>Mode Format Format<br>------------------- - ------------------ ----------------------<br>0 = 32-bit packed 3,2,1,0, -> 15,14.13,12,<br>7,6,5,4, 11,10,9,8<br>11,10,9,8 7,6,5,4<br>‘ 15,14,13,12 3,2,1,0 (bytes)<br>1 = 8 bit native 0,1,2,…14,15 -> 15,14,13,…2,1,0 (bytes)<br>(6 bits right justified) |
| 0 | ENDIAN_INTR | DSP format of interleaver indices in big endian mode. This bit has no effect if in little endian mode.<br>DSP TCP3D<br>Mode ‘ Format Format<br>------------------- - ------------------ ---------------------<br>0 = 32-bit packed 1,0,3,2,5,4,7,6 -> 7,6,5,4,3,2,1,0 (16-bits)<br>1 = 16-bit native 0,1,2,3,4,5,6,7 -> 7,6,5,4,3,2,1,0 (16-bits)<br>(13-bits right justified) |
| **End of Table 4-9** | | |

## 4.3.4 TCP3D Emulation Control Register (TCP3D_EMU)

Figure 4-5 shows the TCP3D Emulation Control register layout and Table 4-10 describes the bits.

**Figure 4-5 TCP3D Emulation Control Register (reset value 0x0000)**

| 31 3 | 2 | 1 | 0 |
|---|---|---|---|
| Reserved | RT_SEL | SOFT | FREERUN |
| R-0 | R/W-0 | R/W-0 | R/W- |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-10 TCP3D Emulation Control Register Bit Description**

| Bit | Name | Description |
|---|---|---|
| 31-3 | Reserved | Reserved (always reads 0) |
| 2 | RT_SEL | RT_SEL bit:<br>0 = TCP3D emulation mode is controlled only by the *TCP3D_emu_dbgsusp* CBA signal. The *TCP3D_emu_dbgsusp_rt* signal is ignored.<br>1 = TCP3D emulation mode is controlled only by the *TCP3D_emu_dbgsusp_rt* CBA signal. The *TCP3D_emu_dbgsusp* signal is ignored. |
| 1 | SOFT | SOFT bit:<br>0 = Hard stop: TCP3D halts as soon as it can in TCP3D Interface State Machine states 0, 1, 5, or 8.<br>1 = Soft stop: TCP3D halts gracefully and completes any decodes in progress. Halts in TCP3D Interface State Machine states 0, 1, or 5. |
| 0 | FREERUN | FREERUN bit:<br>0 = TCP3D responds to the emulation suspend signal it is monitoring and operates according to setting of the SOFT bit.<br>1 = TCP3D ignores emulation suspend signals and runs to completion |
| **End of Table 4-10** | | |

### 4.3.5 TCP3D Execution Control Register (TCP3D_EXE_P0/P1)

Figure 4-6 shows the TCP3D Execution Control register layout and Table 4-11 describes the bits.

**Figure 4-6 TCP3D Process 0 Execution Control Register (reset value 0x0000)**

| 31 3 | 2 0 |
|---|---|
| Reserved | EXE_CMD |
| R-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-11 TCP3D Process 0 Execution Control Register Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31-3 | Reserved | Reserved (always reads 0) |
| 2-0 | EXE_CMD | Execution commands:<br>0 = No Instruction<br>1 = ENABLE: TCP3D Normal Mode<br>4 = Debug mode: Normal initialization and wait in TCP3D Interface State Machine DEBUG and TCP3D Main State Machine state 0.<br>5 = Debug mode: Execute one MAP decode iteration and wait in TCP3D Interface State Machine DEBUG and TCP3D Main State Machine state 5.<br>6 = Debug mode: Execute remaining MAP decode iterations and complete normal ending.<br>7 = TCP3D CLEAR: TCP3D Interface State Machine goes to IF_CLEAR state and remains there until a different command is written. This command also results in all of the other state machines going to the IDLE state and clearing of other, non-state machine, control logic. Does not affect any of the memory mapped registers or the contents of the embedded RAMs.<br>**NOTE:** The TCP3D does not support debug mode when double-buffer mode is enabled |
| **End of Table 4-11** | | |

### 4.3.6 TCP3D Status Register (TCP3D_STS_P0/P1)

The TCP3D Status register is described in Table 4-12. Separate registers are provided for each process to support double-buffered and split modes. In single-buffered mode, only TCP3D_STS_P0 is used, and TCP3D_STS_P1 has no effect on the operation of the TCP3D. In double-buffered or split modes, both registers are used. In double-buffered mode, TCP3D_STS_P0 contains the status of code blocks that were loaded into the Process 0 input buffers, while TCP3D_STS_P1 contains the status of code blocks that were loaded into the Process 1 input buffers. Since there is only one *Interface State Machine* and one Main State Machine in double-buffer mode, some of the fields in these registers track exactly. These are noted in Table 4-12.

Some of the fields in this register are held at their final values when the TCP3D decoding completes for this process, and these are indicated in the table below. These fields are held until the process associated with this register receives the next trigger. All of the fields in TCP3D_STS_P0 and TCP3D_STS_P1 are cleared to 0 when a TCP3D CLEAR command is issued to TCP3D_EXE_P0 and the TCP3D is in double-buffered mode.   Otherwise, a TCP3D CLEAR command issued to TCP3D_EXE_P0 just clears TCP3D_STS_P0. In split mode, a TCP3D CLEAR command issued to TCP3D_EXE_P1 clears TCP3D_STS_P1.

Figure 4-7 shows the TCP3D Execution Control register layout and Table 4-12 describes the bits.

**Figure 4-7 TCP3D Process 0 Status Register (reset value 0x0000)**

| 31 29 | 28 | 27 24 | 23 22 |
|---|---|---|---|
| Reserved | LTE_CRC_CHK | TCP_MAIN_STATE | ACTIVE_MAP |
| R-0 | R-0 | R-0 | R-0 |

| 21 20 | 19 16 | 15 12 | 11 | 10 | 9 |
|---|---|---|---|---|---|
| SNR_EXCEED | TCP_IF_STATE | ACTIVE_ITR | Reserved | EMU_HALT | WAIT_RD_OS |
| R-0 | R-0 | R-0 | R-0 | R-0 | R-0 |

| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| WAIT_RD_HD | WAIT_RD_SO | WAIT_WR_IC | WAIT_WR_IL | WAIT_WR_P1 | WAIT_WR_P0 | WAIT_WR_SYS | PROC_BUSY | PROC_PEND |
| R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 |

Legend: R = Read only; W = Write only; -*n* = value after reset

**Table 4-12 TCP3D Process 0 Status Register Field Descriptions (Part 1 of 2)**

| Bit | Name | Description |
|---|---|---|
| 31-29 | Reserved | Reserved (always reads 0) |
| 28 | LTE_CRC_CHK | LTE CRC Check Status (Only supported for Process 0; RSRVD for process 1; Updated after each CRC calculation in LTE mode; final value held):<br>0 = LTE CRC Check failed or not in LTE mode<br>1 = LTE CRC Check passed<br>This field is only active in LTE mode. |
| 27-24 | TCP_MAIN_STATE | State of the TCP3D main state machine (P0 and P1 fields track in double-buffer mode):<br>0x0 = IDLE<br>0x1 = ZAPM<br>0x2 = MAP0X<br>0x3 = WAIT0<br>0x4 = MAP1X<br>0x5 = WAIT1<br>0x6 = DITR<br>0x7 = DONE |
| 23-22 | ACTIVE_MAP | Active MAP Status (P0 and P1 fields track in double-buffer mode):<br>00 = No active MAP<br>01 = MAP0 active<br>10 =MAP1 active |
| 21-20 | SNR_EXCEED | SNR status (Final status held)[1]:<br>00 = MAP0 failed SNR<br>01 = MAP0 passed SNR<br>This field is also active when the SNR_REP bit in TCP3D_IC_CFG2_P0 and TCP3D_IC_CFG_P1 is set to a 1. |
| 19-16 | TCP_IF_STATE | State of the TCP3D interface state machine as described in Figure 3-39 and Figure 3-52 (P0 and P1 fields track in double-buffer mode):<br>0x0 = IDLE<br>0x1 = WAIT_TRIG<br>0x2 = START_MAP<br>0x3 = WAIT_INTLV<br>0x5 = DEBUG<br>0x8 = WAIT_MAP<br>0x9 = WAIT_RD<br>0xC) = ERROR<br>0xE = END_CLR<br>0xF = IF_CLEAR |
| 15-12 | ACTIVE_ITR | Active iteration status i.e. iteration count (Final count held). VAlues range from 0 to 15. |
| 11 | Reserved | Reserved (always reads 0) |
| 10 | EMU_HALT | Indicates if the TCP3D has halted due to emulation (P0 and P1 fields track in double-buffer mode):<br>0 = Not halted due to emulation<br>1 = Halted due to emulation |
| 9 | WAIT_RD_OS | Indicates if the TCP3D is waiting for output status data to be read:<br>0 = Not waiting<br>1 = Waiting for output status registers to be read |
| 8 | WAIT_RD_HD | Indicates if the TCP3D is waiting for hard decision data to be read:<br>0 = Not waiting<br>1 = Waiting for output/decision memory to be read |
| 7 | WAIT_RD_SO | Indicates if the TCP3D is waiting for soft output data to be read. In all modes except split mode, wait for all 3 soft output memories (systematic, parity 0, parity 1) to be read. In split mode, only need to wait for systematic output memory to be read:<br>0 = Not waiting<br>1 = Waiting for soft output memory to be read |

**Table 4-12 TCP3D Process 0 Status Register Field Descriptions (Part 2 of 2)**

| Bit | Name | Description |
|---|---|---|
| 6 | WAIT_WR_IC | Indicates if the TCP3D is waiting for the input configuration registers to be loaded:<br>0 = Not waiting<br>1 = Waiting for the input configuration registers to be loaded |
| 5 | WAIT_WR_IL | Indicates if the TCP3D is waiting for interleaver indices to be loaded:<br>0 = Not waiting<br>1 = Waiting for interleaver memory to be loaded<br>**NOTE**: This bit is not set to a 1 until all of the input configuration registers are loaded. If the *INTER_LOAD_SEL* bit in TCP3D_IC_CFG2 is set to a 1, this bit remains set until all of the interleaver indices are loaded via DMA or ITG. |
| 4 | WAIT_WR_P1 | Indicates if the TCP3D is waiting for the parity 1 data to be loaded:<br>0 = Not waiting<br>1 = Waiting for the parity 1 data memory to be loaded |
| 3 | WAIT_WR_P0 | Indicates if the TCP3D is waiting for the parity 0 data to be loaded:<br>0 =Not waiting<br>1 = Waiting for the parity 0 data memory to be loaded |
| 2 | WAIT_WR_SYS | Indicates if the TCP3D is waiting for the systematic data to be loaded:<br>0 = Not waiting<br>1 = Waiting for the systematic data memory to be loaded |
| 1 | PROC_BUSY | Indicates if process is busy (i.e. executing):<br>0 = Process NOT busy<br>1 = Process IS busy |
| 0 | PROC_PEND | Indicates when a process has been triggered but has not been serviced yet. In this case this bit stays high until PROC_BUSY is set for the process.<br>0 = Process DOES NOT have a pending trigger<br>1 = Process HAS a pending trigger |
| **End of Table 4-12** | | |

[1] Bit 21 (SNR_EXCEED[1]) is always 0.

### 4.3.7 TCP3D Event Set Register (TCP3D_EV_SET_P0/P1)

The TCP3D Event Set register is described in Table 4-13. It allows the software to generate a DMA event by writing to this register when the *Interface State Machine* is in the IDLE state (see section Section 3.5.1). Separate registers are provided for each process. TCP3D_EV_SET_P0 is used to generate a REVT0 event and TCP3D_EV_SET_P1 is used to generate a REVT1 event. In split mode, writes to TCP3D_EV_SET_P1 will only generate a REVT1 event if the Process 1 *Interface State Machine* is in the IDLE state. Otherwise, the state of the Process *0 Interface State Machine* is always used for all other writes to these registers.

This is only an address and not a physical register. Reads to this register always return 0x0000_0000.

Figure 4-8 shows the TCP3D Execution Control register layout and Table 4-13 describes the bits.

**Figure 4-8 TCP3D Process 0 Event Set Register (reset value 0x0000)**

| 31 1 | 0 |
|---|---|
| RSRVD | REVT |
| W-0 | W-0 |

Legend: W = Write only; -*n* = value after reset

**Table 4-13 TCP3D Process 0 Event Set Register Field Descriptions**

| Bit | Name | Value | Description |
|---|---|---|---|
| 31-1 | RSRVD | N/A | Reserved |
| 0 | REVT | 1 | Generate REVT event to DMA controller |
| **End of Table 4-13** | | | |

## 4.4 TCP3D Error Interrupt Registers (VBUS_CFG Bus)

To support double-buffered and split modes, the TCP3D provides separate TCP3D error interrupts for each process. In single-buffered mode, only Process 0 will generate an error interrupt. As such, two sets of TCP3D error interrupt registers are described below.

### 4.4.1 TCP3D Error Interrupt Raw Status/Set Register (TCP3D_EINT_STAT_P0/P1)

The TCP3D Error Interrupt Raw Status and Set register contains the interrupt status bit for each error condition before any enable processing. It is described in Table 4-14. The interrupt status bits are set to 1 by an error condition or writing a 1 (used for debug to allow software to set an interrupt status bit). Writing a 0 has no effect. They are cleared when a 1 is written to the associated bit in the TCP3D Error Interrupt Clear register.

Table 4-15 provides an explanation of the effect of each enabled error condition on the internal operation of the TCP3D. With the exception of BLK_LN_ERR and SW0_LN_ERR, as described in Table 4-14, the TCP3D Interface State Machine will only halt in the ERROR state when the *ERR_IGNORE_EN* bit in TCP3D_MODE is cleared to a 0.

**Note—** Ignoring errors detected when a trigger occurs (i.e. either by setting *ERR_IGNORE_EN* to a 1 or disabling a specific error condition) can result in other error conditions being reported once the decode starts. As an example, ignoring a *DMA_UCFG_ERR* error could result in an *INTLV_CONT_ERR* error. Also, there is no guarantee that the results of the decode are correct if errors are ignored.

Figure 4-9 shows the TCP3D Error Interrupt Raw Status/Set register layout and Table 4-14 describes the bits.

**Figure 4-9 TCP3D Process 0 Error Interrupt Raw Status and Set Register (reset value 0x0000)**

| 31 29 | 28 | 27 | 26 | 25 | 24 | 23 |
|---|---|---|---|---|---|---|
| Reserved | VD_WADDR_ERR | VD_RADDR_ERR | VC_WADDR_ERR | VC_RADDR_ERR | DMA_OINTLV_ERR | DMA_OPAR1_ERR |
| R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | | R/W-0 |

| 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|
| DMA_OPAR0_ERR | DMA_OSYS_ERR | Reserved | DMA_UPAR1_ERR | DMA_UPAR0_ERR | DMA_USYS_ERR | DMA_UCFG_ERR |
| R/W-0 | R/W-0 | R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |

| 15 | 14 | 13 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|
| TRIG_ERR | MAX_MIN_IT_ERR | Reserved | IPROG_ACC_ERR | ICFG_ACC_ERR | IDAT_FMT_ERR | SNR_VAL_ERR |
| R/W-0 | R/W-0 | R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | SW0_LN_ERR | END_REG_WR_ERR | SPLIT_TCP_ERR | MODE_REG_WR_ERR | INTLV_CONT_ERR | BLK_LN_ERR | Reserved |
| R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-14 TCP3D Process 0 Error Interrupt Raw Status and Set Register Bit Description (Part 1 of 3)**

| Bit | Name | Description[1] |
|---|---|---|
| 31-29 | Reserved | Reserved (always reads 0) |
| 28 | VD_WADDR_ERR[2] | VBUS_DMA Undefined Write Address Access:<br>0 = No error<br>1 = A write access to an undefined memory location was detected. |
| 27 | VD_RADDR_ERR[2] | VBUS_DMA Undefined Read Address Access:<br>0 = No error<br>1 = A read access to an undefined memory location was detected. |
| 26 | VC_WADDR_ERR[2] | VBUS_CFG Undefined Write Address Access:<br>0 = No error<br>1 = A write access to an undefined memory location was detected. |
| 25 | VC_RADDR_ERR[2] | VBUS_CFG Undefined Read Address Access:<br>0 = No error<br>1 = A read access to an undefined memory location was detected. |
| 24 | DMA_OINTLV_ERR[3] | Interleaver memory DMA transfer overflow error:<br>0 = No error<br>1 = DMA transfer to interleaver memory is greater than the programmed code block size[4]when the TCP3D Interface State Machine transitions to the WAIT_MAP state.<br>**Note**: This error is reported only if the interleaver overflow occurs prior to the trigger. An overflow that occurs after the trigger is reported as an *IPROG_ACC_ERR*. |
| 23 | DMA_OPAR1_ERR[3] | Parity 1 data memory DMA transfer overflow error:<br>0 = No error<br>1 = DMA transfer to parity 1 data memory is greater than the programmed code block size[4]when process triggered. |
| 22 | DMA_OPAR0_ERR[3] | Parity 0 data memory DMA transfer overflow error:<br>0 = No error<br>1 = DMA transfer to parity 0 data memory is greater than the programmed code block size[4] when process triggered. |

**Table 4-14 TCP3D Process 0 Error Interrupt Raw Status and Set Register Bit Description (Part 2 of 3)**

| Bit | Name | Description[1] |
|---|---|---|
| 21 | DMA_OSYS_ERR[4] | Systematic data memory DMA transfer overflow error:<br>0 = No error<br>1 = DMA transfer to systematic data memory is greater than the programmed code block size[4] when process triggered. |
| 20 | RSRVD | Reserved (always reads 0) |
| 19 | DMA_UPAR1_ERR | Parity 1 data memory DMA transfer underflow error (only valid if auto trigger mode disabled):<br>0 = No error<br>1 = DMA transfer to parity 1 data memory is less than the programmed code block size[4] when process triggered. |
| 18 | DMA_UPAR0_ERR | Parity 0 data memory DMA transfer underflow error (only valid if auto trigger mode disabled):<br>0 = No error<br>1 = DMA transfer to parity 0 data memory is less than the programmed code block size[4] when process triggered. |
| 17 | DMA_USYS_ERR | Systematic data memory DMA transfer underflow error (only valid if auto trigger mode disabled):<br>0 = No error<br>1 = DMA transfer to systematic data memory is less than the programmed code block size[4] when process triggered. |
| 16 | DMA_UCFG_ERR | Input configuration register DMA transfer underflow error (only valid if auto trigger mode disabled):<br>0 = No error<br>1 = DMA transfer to the input configuration registers does not write to all of the registers. |
| 15 | TRIG_ERR[5] | Trigger received for a process that is either busy or has a pending trigger (i.e. *PROC_BUSY* or *PROC_PEND* set):<br>0 = No error<br>1 = Trigger error |
| 14 | MAX_MIN_IT_ERR | Max/min iteration error:<br>0 = No error<br>1 = MIN_IT > MAX_IT |
| 13-12 | Reserved | Reserved (always reads 0) |
| 11 | IPROG_ACC_ERR | TCP3D *in progress* memory access error:<br>0 = No error.<br>1 = TCP3D *in progress* memory access error:<br>The TCP3D memories were accessed at the wrong time. This error is disabled when the TCP3D is in halted in debug or emulation modes. See Section 3.6.3 for a more detailed explanation. |
| 10 | ICFG_ACC_ERR | Spurious TCP3D input configuration access error:<br>0 = No error.<br>1 = TCP3D input configuration access error:<br>The input configuration registers were written after the decoder was triggered[5]. |
| 9 | IDAT_FMT_ERR | Input data format error:<br>0 = No error.<br>1 = Input data format error:<br>The systematic, parity 0 or parity 1 data sent to the TCP3D has a non-zero value in the 2 MSBs of any byte field. These 2 bits are not used internally in the TCP3D and indicate a data error. |
| 8 | SNR_VAL_ERR | SNR threshold value error:<br>0 = No error<br>1 = SNR threshold value > 20dB |
| 7 | RSRVD | Reserved (always reads 0) |
| 6 | SW0_LN_ERR[4] | SW0 Length Error:<br>0 = No error<br>1 = SW0 length less than SW1 length. SW0 length represented by SW0_LN_SEL is less than SW1_LN. |
| 5 | END_REG_WR_ERR | Endian register write error:<br>0 = No error<br>1 = The endian register was written when the TCP3D Interface state machine was not in the IDLE state[6] |
| 4 | SPLIT_TCP_ERR | Split TCP3D programming error:<br>0 = No error<br>1 = TCP3D configured for split mode and double-buffering. TCP3D_MODE has the following settings in this case:<br>» *MODE_SEL* = 0x3<br>» *IN_MEM_DB_EN* = 0x1 |

### Table 4-14 TCP3D Process 0 Error Interrupt Raw Status and Set Register Bit Description (Part 3 of 3)

| Bit | Name | Description[1] |
|---|---|---|
| 3 | MODE_REG_WR_ERR | Mode register write error:<br>0 = No error<br>1 = The mode register was written when the TCP3D Interface state machine was not in the IDLE state[6]. |
| 2 | INTLV_CONT_ERR | Interleaver contention error:<br>0 = No error<br>1 = Interleaver memory contention is detected for non-contentious configurations of LTE and WiMAX (except 27 byte). |
| 1 | BLK_LN_ERR[7] | Block length error:<br>0 = No error<br>1 = Block length less than minimum. code block size supported (BLK_LN <39) |
| 0 | Reserved | Reserved (always reads 0) |
| **End of Table 4-14** | | |

[1] Writing a 1 to any non-reserved bit in this register will set it to a 1. Writing a 0 will not change the value of the bit.

[2] In double-buffer and SPLIT modes, undefined VBUS access errors result in this error bit being set in both TCP3D_EINT_STAT_P0 and TCP3D_EINT_STAT_P1 registers. For single buffer mode, only the bit in TCP3D_EINT_STAT_P0 will be set.

[3] When the TCP3D is halted in emulation mode, the internal counters that control these error bits are still enabled and writes to the TCP3D can cause these bits to be asserted. When the TCP3D is stopped due to debug mode, these same internal counters are disabled and writes to the TCP3D will not cause these bits to be asserted.

[4] Extended code block size is used for 3GPP code block sizes not divisible by 4 as described in Section 3.3.1 For systematic and parity data the code block size is double extended. For interleaver data, the code block size is only single extended. Additionally, the overflow logic rounds up the code block size it uses to detect overflow to a multiple of 4 for all modes.

[5] Any prior errors due to input configuration errors that had been reported when the original trigger occurred and have been already cleared will be reported again when this spurious trigger occurs. These include BLK_LN_ERR, SW0_LN_ERR, SNR_VAL_ERR, and MAX_MIN_IT_ERR.

[6] In SPLIT mode both interface state machines can be active at the same time. So, a write to the mode or endian registers when both TCP3D interface state machines are not in the IDLE state will result in this error bit being set in both the TCP3D_EINT_STAT_P0 and TCP3D_EINT_STAT_P1 registers. If the bits from both registers are enabled to generate an interrupt, both interrupts from the TCP3D will be asserted at the same time. In double-buffer mode, this error bit will also be set in both the TCP3D_EINT_STAT_P0 and TCP3D_EINT_STAT_P1 registers since only one interface state machine is active and the same state information is used in the generation of both error bits.

[7] These are special cases. See further description in Table 4-14.

**Table 4-15 TCP3D Internal Error Actions**

| Bit | Name | TCP3D Internal Error Action |
|---|---|---|
| 28 | VD_WADDR_ERR | No effect; Access is acknowledged and ignored |
| 27 | VD_RADDR_ERR | No effect; Access is acknowledged and ignored |
| 26 | VC_WADDR_ERR | No effect; Access is acknowledged and ignored |
| 25 | VC_RADDR_ERR | No effect; Access is acknowledged and ignored |
| 24 | DMA_OINTLV_ERR | TCP3D Interface State Machine halts in ERROR state |
| 23 | DMA_OPAR1_ERR | TCP3D Interface State Machine halts in ERROR state |
| 22 | DMA_OPAR0_ERR | TCP3D Interface State Machine halts in ERROR state |
| 21 | DMA_OSYS_ERR | TCP3D Interface State Machine halts in ERROR state |
| 19 | DMA_UPAR1_ERR | TCP3D Interface State Machine halts in ERROR state |
| 18 | DMA_UPAR0_ERR | TCP3D Interface State Machine halts in ERROR state |
| 17 | DMA_USYS_ERR | TCP3D Interface State Machine halts in ERROR state |
| 16 | DMA_UCFG_ERR | TCP3D Interface State Machine halts in ERROR state |
| 15 | TRIG_ERR | Trigger ignored by TCP3D Interface State Machine |
| 14 | MAX_MIN_IT_ERR | TCP3D Interface State Machine halts in ERROR state |
| 11 | IPROG_ACC_ERR | Access is ignored; No effect if access is to systematic, parity 0, parity 1 or interleaver memories. Accesses to output/decision and soft output memories may corrupt TCP3D (see Section 3.6.3) |
| 10 | ICFG_ACC_ERR | Internal operation may be corrupted |
| 9 | IDAT_FMT_ERR | TCP3D Interface State Machine halts in ERROR state |
| 8 | SNR_VAL_ERR | TCP3D Interface State Machine halts in ERROR state |
| 6 | SW0_LN_ERR | TCP3D Interface State Machine halts in ERROR state<br>**NOTE**: This error condition will always cause the Interface State Machine to halt in the ERROR state regardless of whether the ERR_IGNORE_EN bit in TCP3D_MODE is set to a 1 or this condition is enabled to cause an interrupt since it can cause unpredictable behavior. |
| 5 | END_REG_WR_ERR | Internal operation may be corrupted |
| 4 | SPLIT_TCP_ERR | TCP3D Interface State Machine halts in ERROR state |
| 3 | MODE_REG_WR_ERR | Internal operation may be corrupted |
| 2 | INTLV_CONT_ERR | Turbo decoding corrupted |
| 1 | BLK_LN_ERR | TCP3D Interface State Machine halts in ERROR state.<br>**NOTE:** This error condition will always cause the Interface State Machine to halt in the ERROR state regardless of whether the ERR_IGNORE_EN bit in TCP3D_MODE is set to a 1 or this condition is enabled to cause an interrupt since it can cause the TCP3D to hang, otherwise. |
| **End of Table 4-15** | | |

In double-buffered and split modes, not all of the errors reported in TCP3D_EINT_STAT_P0 are relevant for TCP3D_EINT_STAT_P1. Table 4-16 shows which errors are reported in TCP3D_EINT_STAT_P1 for these modes. An entry of Active indicates an error that is reported, while errors that are not reported have an entry of Reserved. For any Reserved bit, the associated entries in the various TCP3D error interrupt registers described below would also be Reserved.

**Table 4-16 TCP3D_EINT_STAT_P1 Reported Errors in Double-Buffered and Split Modes**

| Bit | Name | Double Buffered Mode | Split Mode |
|---|---|---|---|
| 28 | VD_WADDR_ERR | Active | Active |
| 27 | VD_RADDR_ERR | Active | Active |
| 26 | VC_WADDR_ERR | Active | Active |
| 25 | VC_RADDR_ERR | Active | Active |
| 24 | DMA_OINTLV_ERR | Active | Active |
| 23 | DMA_OPAR1_ERR | Active | Active |
| 22 | DMA_OPAR0_ERR | Active | Active |
| 21 | DMA_OSYS_ERR | Active | Active |
| 19 | DMA_UPAR1_ERR | Active | Active |
| 18 | DMA_UPAR0_ERR | Active | Active |
| 17 | DMA_USYS_ERR | Active | Active |
| 16 | DMA_UCFG_ERR | Active | Active |
| 15 | TRIG_ERR | Active | Active |
| 14 | MAX_MIN_IT_ERR | Active | Active |
| 11 | IPROG_ACC_ERR | Active | Active |
| 10 | ICFG_ACC_ERR | Active | Active |
| 9 | IDAT_FMT_ERR | Active | Active |
| 8 | SNR_VAL_ERR | Active | Active |
| 6 | SW0_LN_ERR | Active | Active |
| 5 | END_REG_WR_ERR | Active | Active |
| 4 | SPLIT_TCP_ERR | Active | Active |
| 3 | MODE_REG_WR_ERR | Active | Active |
| 2 | INTLV_CONT_ERR[1] | Active | RSRVD |
| 1 | BLK_LN_ERR | Active | Active |
| **End of Table 4-16** | | | |

[1] INTLV_CONT_ERR also reserved for P0 in split mode since 3GPP interleaver is not contention free.

### 4.4.2 TCP3D Error Interrupt Clear Register (TCP3D_EINT_CLR_P0/P1)

The TCP3D Error Interrupt Clear register is a write-only shadow register that mirrors the interrupt status bits in the TCP3D Error Interrupt Raw Status and Set register. Writing a 1 here causes the associated interrupt status bit in the TCP3D Error Interrupt Raw Status and Set register to be cleared to a 0. Used by software to clear an interrupt condition after it has been serviced. Writing a 0 has no effect. Reading this register will return the value in the TCP3D Error Interrupt Raw Status and Set register. This is only an address and not a physical register. It is described in Table 4-17.

Figure 4-10 shows the TCP3D Error Interrupt Clear register layout and Table 4-17 describes the bits.

**Figure 4-10 TCP3D Process 0 Error Interrupt Clear Register (reset value 0x0000)**

| 31 29 | 28 | 27 | 26 | 25 | 24 | 23 |
|---|---|---|---|---|---|---|
| Reserved | VD_WADDR_CLR | VD_RADDR_CLR | VC_WADDR_CLR | VC_RADDR_CLR | DMA_OINTLV_CLR | DMA_OPAR1_CLR |
| R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | | R/W-0 |

| 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|
| DMA_OPAR0_CLR | DMA_OSYS_CLR | Reserved | DMA_UPAR1_CLR | DMA_UPAR0_CLR | DMA_USYS_CLR | DMA_UCFG_CLR |
| R/W-0 | R/W-0 | R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |

| 15 | 14 | 13 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|
| TRIG_CLR | MAX_MIN_IT_CLR | Reserved | IPROG_ACC_CLR | ICFG_ACC_CLR | IDAT_FMT_CLR | SNR_VAL_CLR |
| R/W-0 | R/W-0 | R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | SW0_LN_CLR | END_REG_WR_CLR | SPLIT_TCP_CLR | MODE_REG_WR_CLR | INTLV_CONT_CLR | BLK_LN_CLR | Reserved |
| R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-17 TCP3D Process 0 Error Interrupt Clear Register Field Descriptions (Part 1 of 2)**

| Bit | Name | Description |
|---|---|---|
| 31-29 | Reserved | Reserved |
| 28 | VD_WADDR_CLR | Clear VBUS_DMA undefined write address access error |
| 27 | VD_RADDR_CLR | Clear VBUS_DMA undefined read address access error |
| 26 | VC_WADDR_CLR | Clear VBUS_CFG undefined write address access error |
| 25 | VC_RADDR_CLR | Clear VBUS_CFG undefined read address access error |
| 24 | DMA_OINTLV_CLR | Clear interleaver memory DMA transfer overflow error interrupt status bit |
| 23 | DMA_OPAR1_CLR | Clear parity 1 data memory DMA transfer overflow error interrupt status bit |
| 22 | DMA_OPAR0_CLR | Clear parity 0 data memory DMA transfer overflow error interrupt status bit |
| 21 | DMA_OSYS_CLR | Clear systematic data memory DMA transfer overflow error interrupt status bit |
| 20 | Reserved | Reserved |
| 19 | DMA_UPAR1_CLR | Clear parity 1 data memory DMA transfer underflow error interrupt status bit |
| 18 | DMA_UPAR0_CLR | Clear parity 0 data memory DMA transfer underflow error interrupt status bit |
| 17 | DMA_USYS_CLR | Clear systematic data memory DMA transfer underflow error interrupt status bit |
| 16 | DMA_UCFG_CLR | Clear input configuration register DMA transfer underflow error interrupt status bit |
| 15 | TRIG_CLR | Clear trigger error interrupt status bit |
| 14 | MAX_MIN_IT_CLR | Clear max/min iteration error interrupt status bit |
| 13-12 | Reserved | Reserved |
| 11 | IPROG_ACC_CLR | Clear TCP3D *in progress* memory access error interrupt status bit |
| 10 | ICFG_ACC_CLR | Clear spurious TCP3D input configuration access error interrupt status bit |
| 9 | IDAT_FMT_CLR | Clear input data format error interrupt status bit |
| 8 | SNR_VAL_CLR | Clear SNR threshold value error interrupt status bit |
| 7 | Reserved | Reserved |
| 6 | SW0_LN_CLR | Clear SW0 length error interrupt status bit |
| 5 | END_REG_WR_CLR | Clear endian register write error interrupt status bit |

**Table 4-17 TCP3D Process 0 Error Interrupt Clear Register Field Descriptions (Part 2 of 2)**

| Bit | Name | Description |
|---|---|---|
| 4 | SPLIT_TCP_CLR | Clear split TCP3D programming error interrupt status bit |
| 3 | MODE_REG_WR_CLR | Clear mode register write error interrupt status bit |
| 2 | INTLV_CONT_CLR | Clear interleaver contention error interrupt status bit |
| 1 | BLK_LN_CLR | Clear code block length error interrupt status bit |
| 0 | Reserved | Reserved |
| **End of Table 4-17** | | |

### 4.4.3 TCP3D Error Interrupt Enabled Status Register (TCP3D_EINT_EN_STAT_P0/P1)

The TCP3D Error Interrupt Enabled Status register is a read-only register that is the logical AND of the TCP3D Error Interrupt Raw Status and Set register and the TCP3D Error Interrupt Enable and Set register. This is only an address and not a physical register. It is described in Table 4-18.

Figure 4-11 shows the Error Interrupt Enabled Status register layout and Table 4-18 describes the bits.

**Figure 4-11 TCP3D Process 0 Error Interrupt Enabled Status Register (reset value 0x0000)**

| 31 29 | 28 | 27 | 26 | 25 | 24 | 23 |
|---|---|---|---|---|---|---|
| Reserved | VD_WADDR_ERR | VD_RADDR_ERR | VC_WADDR_ERR | VC_RADDR_ERR | DMA_OINTLV_ERR | DMA_OPAR1_ERR |
| R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | | R/W-0 |

| 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|
| DMA_OPAR0_ERR | DMA_OSYS_ERR | Reserved | DMA_UPAR1_ERR | DMA_UPAR0_ERR | DMA_USYS_ERR | DMA_UCFG_ERR |
| R/W-0 | R/W-0 | R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |

| 15 | 14 | 13 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|
| TRIG_ERR | MAX_MIN_IT_ERR | Reserved | IPROG_ACC_ERR | ICFG_ACC_ERR | IDAT_FMT_ERR | SNR_VAL_ERR |
| R/W-0 | R/W-0 | R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | SW0_LN_ERR | END_REG_WR_ERR | SPLIT_TCP_ERR | MODE_REG_WR_ERR | INTLV_CONT_ERR | BLK_LN_ERR | Reserved |
| R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-18 TCP3D Process 0 Error Interrupt Enabled Status Register Field Descriptions (Part 1 of 3)**

| Bit | Name | Description |
|---|---|---|
| 31-29 | Reserved | Reserved (always reads 0) |
| 28 | VD_WADDR_ERR[1] | VBUS_DMA Undefined Write Address Access:<br>0 = No error<br>1 = A write access to an undefined memory location was detected. |
| 27 | VD_RADDR_ERR[1] | VBUS_DMA Undefined Read Address Access:<br>0 = No error<br>1 = A read access to an undefined memory location was detected. |

**Table 4-18 TCP3D Process 0 Error Interrupt Enabled Status Register Field Descriptions (Part 2 of 3)**

| Bit | Name | Description |
|---|---|---|
| 26 | VC_WADDR_ERR[1] | VBUS_CFG Undefined Write Address Access:<br>0 = No error<br>1 = A write access to an undefined memory location was detected. |
| 25 | VC_RADDR_ERR[1] | VBUS_CFG Undefined Read Address Access:<br>0 = No error<br>1 = A read access to an undefined memory location was detected. |
| 24 | DMA_OINTLV_ERR | Interleaver memory DMA transfer overflow error:<br>0 = No error<br>1 = DMA transfer to interleaver memory is greater than the programmed code block size[2] when the TCP3D Interface State Machine transitions to the WAIT_MAP state. |
| 23 | DMA_OPAR1_ERR | Parity 1 data memory DMA transfer overflow error:<br>0 = No error<br>1 = DMA transfer to parity 1 data memory is greater than the programmed code block size [2] when process triggered. |
| 22 | DMA_OPAR0_ERR | Parity 0 data memory DMA transfer overflow error:<br>0 = No error<br>1 = DMA transfer to parity 0 data memory is greater than the programmed code block size[2] when process triggered. |
| 21 | DMA_OSYS_ERR | Systematic data memory DMA transfer overflow error:<br>0 = No error<br>1 = DMA transfer to systematic data memory is greater than the programmed code block size[2] when process triggered. |
| 20 | Reserved | Reserved (always reads 0) |
| 19 | DMA_UPAR1_ERR | Parity 1 data memory DMA transfer underflow error (only valid if auto trigger mode disabled):<br>0 = No error<br>1 = DMA transfer to parity 1 data memory is less than the programmed code block size[2] when process triggered. |
| 18 | DMA_UPAR0_ERR | Parity 0 data memory DMA transfer underflow error (only valid if auto trigger mode disabled):<br>0 = No error<br>1 = DMA transfer to parity 0 data memory is less than the programmed code block size[2] when process triggered. |
| 17 | DMA_USYS_ERR | Systematic data memory DMA transfer underflow error (only valid if auto trigger mode disabled):<br>0 = No error<br>1 = DMA transfer to systematic data memory is less than the programmed code block size[2] when process triggered. |
| 16 | DMA_UCFG_ERR | Input configuration register DMA transfer underflow error (only valid if auto trigger mode disabled):<br>0 = No error<br>1 - DMA transfer to the input configuration registers does not write to all of the registers. |
| 15 | TRIG_ERR | Trigger received for a process that is either busy or has a pending trigger (i.e. *PROC_BUSY* or *PROC_PEND* set):<br>0 = No error<br>1 = Trigger error[3] |
| 14 | MAX_MIN_IT_ERR | Max/min iteration error:<br>0 = No error<br>1 = MIN_IT > MAX_IT |
| 13-12 | Reserved | Reserved (always reads 0) |
| 11 | IPROG_ACC_ERR | TCP3D “in progress” memory access error:<br>0 = No error.<br>1 = TCP3D “in progress” memory access error:<br>The embedded RAMs in the TCP3D were accessed at the wrong time. |
| 10 | ICFG_ACC_ERR | Spurious TCP3D input configuration access error:<br>0 = No error.<br>1 = TCP3D input configuration access error:<br>The input configuration registers were written after the decoder was triggered[4]. |

**Table 4-18 TCP3D Process 0 Error Interrupt Enabled Status Register Field Descriptions (Part 3 of 3)**

| Bit | Name | Description |
|---|---|---|
| 9 | IDAT_FMT_ERR | Input data format error:<br>0 = No error.<br>1 = Input data format error:<br>The systematic, parity 0 or parity 1 data sent to the TCP3D has a non-zero value in the 2 MSBs of any byte field. These 2 bits are not used internally in the TCP3D and indicate a data error. |
| 8 | SNR_VAL_ERR | SNR threshold value error:<br>0 = No error<br>1 = SNR threshold value > 20dB |
| 7 | Reserved | Reserved (always reads 0) |
| 6 | SW0_LN_ERR | SW0 Length Error:<br>0 = No error<br>1 = SW0 length less than SW1 length. SW0 length represented by SW0_LN_SEL is less than SW1_LN. |
| 5 | END_REG_WR_ERR | Endian register write error:<br>0 = No error<br>1 = The endian register was written when the TCP3D Interface state machine was not in the IDLE state. |
| 4 | SPLIT_TCP_ERR | Split TCP3D programming error<br>0 = No error<br>1 = TCP3D configured for split mode and double-buffering. TCP3D_MODE has the following settings in this case:<br>*MODE_SEL* = 0x3<br>*IN_MEM_DB_EN* = 0x1 |
| 3 | MODE_REG_WR_ERR | Mode register write error:<br>0 = No error<br>1 = The mode register was written when the TCP3D Interface state machine was not in the IDLE state. |
| 2 | INTLV_CONT_ERR | Interleaver contention error:<br>0 = No error<br>1 = Interleaver memory contention is detected for non-contentious configurations of LTE and WiMAX (except 27 byte). |
| 1 | BLK_LN_ERR | Block length error:<br>0 = No error<br>1 = Block length less than min code block size supported (BLK_LN-1 <39) |
| 0 | Reserved | Reserved (always reads 0) |
| **End of Table 4-18** | | |

1. In double-buffer and SPLIT modes, undefined VBUS access errors result in this error bit being set in both TCP3D_EINT_STAT_P0 and TCP3D_EINT_STAT_P1 registers. For single buffer mode, only the bit in TCP3D_EINT_STAT_P0 will be set.
2. Extended code block size is used for 3GPP code block sizes not divisible by 4 as described in Section 3.3.1. For systematic and parity data the code block size is double extended. For interleaver data, the code block size is only single extended. Additionally, the overflow logic rounds up the code block size it uses to detect overflow to a multiple of 4 for all modes.
3. Any prior errors due to input configuration errors that had been reported when the original trigger occurred and have been already cleared will be reported again when this spurious trigger occurs. These include BLK_LN_ERR, SW0_LN_ERR, SNR_VAL_ERR, and MAX_MIN_IT_ERR.
4. The ICFG_ACC_ERR bit is also set when TRIG_ERR is set, since the decoding for ICFG_ACC_ERR also includes the **TCP3D_TRIG_P0** or **TCP3D_TRIG_P1** registers and writes to these registers cause TRIG_ERR.

### 4.4.4 TCP3D Error Interrupt Enable and Set Register (TCP3D_EINT_EN_P0/P1)

The TCP3D Error Interrupt Enable and Set register contains the enables for each of the interrupt status bits in the TCP3D Error Interrupt Raw Status and Set register. An interrupt status bit can only cause an interrupt if it is enabled. It is described in Table 4-19.

Interrupt enable bits are set to a 1 by writing a 1, but writing a 0 leaves the value un-changed. Interrupt enable bits are cleared by writing to the TCP3D Interrupt Error Interrupt Enable Clear registers described below.

Figure 4-12 shows the TCP3D Process 0 Error Interrupt Enable and Set Register layout and Table 4-19 describes the bits.

**Figure 4-12 TCP3D Process 0 Error Interrupt Enable and Set Register (reset value 0x0000)**

| 31 29 | 28 | 27 | 26 | 25 | 24 | 23 |
|---|---|---|---|---|---|---|
| Reserved | VD_WADDR_EN | VD_RADDR_EN | VC_WADDR_EN | VC_RADDR_EN | DMA_OINTLV_EN | DMA_OPAR1_EN |
| R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |  | R/W-0 |

| 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|
| DMA_OPAR0_EN | DMA_OSYS_EN | Reserved | DMA_UPAR1_EN | DMA_UPAR0_EN | DMA_USYS_EN | DMA_UCFG_EN |
| R/W-0 | R/W-0 | R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |

| 15 | 14 | 13 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|
| TRIG_EN | MAX_MIN_IT_EN | Reserved | IPROG_ACC_EN | ICFG_ACC_EN | IDAT_FMT_EN | SNR_VAL_EN |
| R/W-0 | R/W-0 | R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | SW0_LN_EN | END_REG_WR_EN | SPLIT_TCP_EN | MODE_REG_WR_EN | INTLV_CONT_EN | BLK_LN_EN | Reserved |
| R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-19 TCP3D Process 0 Error Interrupt Enable and Set Register Field Descriptions (Part 1 of 3)**

| Bit | Name | Description[1] |
|---|---|---|
| 31-29 | Reserved | Reserved (always reads 0) |
| 28 | VD_WADDR_EN | VBUS_DMA Undefined Write Address Access:<br>0 = Disabled<br>1 = Enabled |
| 27 | VD_RADDR_EN | VBUS_DMA Undefined Read Address Access:<br>0 = Disabled<br>1 = Enabled |
| 26 | VC_WADDR_EN | VBUS_CFG Undefined Write Address Access:<br>0 = Disabled<br>1 = Enabled |
| 25 | VC_RADDR_EN | VBUS_CFG Undefined Read Address Access:<br>0 = Disabled<br>1 = Enabled |
| 24 | DMA_OINTLV_EN | Interleaver memory DMA transfer overflow error:<br>0 = Disabled<br>1 = Enabled |
| 23 | DMA_OPAR1_EN | Parity 1 data memory DMA transfer overflow error:<br>0 = Disabled<br>1 = Enabled |
| 22 | DMA_OPAR0_EN | Parity 0 data memory DMA transfer overflow error:<br>0 = Disabled<br>1 = Enabled |

**Table 4-19 TCP3D Process 0 Error Interrupt Enable and Set Register Field Descriptions (Part 2 of 3)**

| Bit | Name | Description[1] |
|---|---|---|
| 21 | DMA_OSYS_EN | Systematic data memory DMA transfer overflow error:<br>0 = Disabled<br>1 = Enabled |
| 20 | Reserved | Reserved (always reads 0) |
| 19 | DMA_UPAR1_EN | Parity 1 data memory DMA transfer underflow error:<br>0 = Disabled<br>1 = Enabled |
| 18 | DMA_UPAR0_EN | Parity 0 data memory DMA transfer underflow error:<br>0 = Disabled<br>1 = Enabled |
| 17 | DMA_USYS_EN | Systematic data memory DMA transfer underflow error:<br>0 = Disabled<br>1 = Enabled |
| 16 | DMA_UCFG_EN | Input configuration register DMA transfer underflow error:<br>0 = Disabled<br>1 = Enabled |
| 15 | TRIG_EN | Trigger error:<br>0 = Disabled<br>1 = Enabled |
| 14 | MAX_MIN_IT_EN | Max/min iteration error:<br>0 = Disabled<br>1 = Enabled |
| 13-12 | Reserved | Reserved (always reads 0) |
| 11 | IPROG_ACC_EN | TCP3D *in progress* memory access error:<br>0 = Disabled<br>1 = Enabled |
| 10 | ICFG_ACC_EN | Spurious TCP3D input configuration access error:<br>0 = Disabled<br>1 = Enabled |
| 9 | IDAT_FMT_EN | Input data format error:<br>0 = Disabled<br>1 = Enabled |
| 8 | SNR_VAL_EN | SNR threshold value error:<br>0 = Disabled<br>1 = Enabled |
| 7 | Reserved | Reserved (always reads 0) |
| 6 | SW0_LN_EN | SW0 length error:<br>0 = Disabled<br>1 = Enabled |
| 5 | END_REG_WR_EN | Endian register write error:<br>0 = Disabled<br>1 = Enabled |
| 4 | SPLIT_TCP_EN | Split TCP3D programming error:<br>0 = Disabled<br>1 = Enabled |
| 3 | MODE_REG_WR_EN | Mode register write error:<br>0 = Disabled<br>1 = Enabled |
| 2 | INTLV_CONT_EN | Interleaver contention error:<br>0 = Disabled<br>1 = Enabled |

**Table 4-19 TCP3D Process 0 Error Interrupt Enable and Set Register Field Descriptions (Part 3 of 3)**

| Bit | Name | Description[1] |
|---|---|---|
| 1 | BLK_LN_EN | Block length error:<br>0 = Disabled<br>1 = Enabled |
| 0 | Reserved | Reserved (always reads 0) |
| **End of Table 4-19** | | |

[1] Writing a 1 to any non-reserved bit in this register will set it to a 1. Writing a 0 will not change the value of the bit.

### 4.4.5 TCP3D Error Interrupt Enable Clear Register (TCP3D_EINT_EN_CLR_P0/P1)

The TCP3D Error Interrupt Enable Clear register is a write-only shadow register that mirrors the interrupt enable bits in the TCP3D Error Interrupt Enable and Set register. Writing 1 causes the associated interrupt enable in the TCP3D Error Interrupt Enable and Set register to be cleared to a 0 and disables the associated interrupt status bit. Writing a 0 has no effect. Reading this register will return the value in the TCP3D Error Interrupt Enable and Set register. This is only an address and not a physical register. It is described in Table 4-20.

Figure 4-13 shows the TCP3D Process 0 Error Interrupt Enable Clear Register layout and Table 4-20 describes the bits.

**Figure 4-13 TCP3D Process 0 Error Interrupt Enable Clear Register (reset value 0x0000)**

| 31 29 | 28 | 27 | 26 | 25 | 24 | 23 |
|---|---|---|---|---|---|---|
| Reserved | VD_WADDR_ECL | VD_RADDR_ECL | VC_WADDR_ECL | VC_RADDR_ECL | DMA_OINTLV_ECL | DMA_OPAR1_ECL |
| R-0 | W-0 | W-0 | W-0 | W-0 | W-0 | W-0 |

| 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|
| DMA_OPAR0_ECL | DMA_OSYS_ECL | Reserved | DMA_UPAR1_ECL | DMA_UPAR0_ECL | DMA_USYS_ECL | DMA_UCFG_ECL |
| W-0 | W-0 | R-0 | W-0 | W-0 | W-0 | W-0 |

| 15 | 14 | 13 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|
| TRIG_ECL | MAX_MIN_IT_ECL | Reserved | IPROG_ACC_ECL | ICFG_ACC_ECL | IDAT_FMT_ECL | SNR_VAL_ECL |
| W-0 | W-0 | R-0 | W-0 | W-0 | W-0 | W-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | SW0_LN_ECL | END_REG_WR_ECL | SPLIT_TCP_ECL | MODE_REG_WR_ECL | INTLV_CONT_ECL | BLK_LN_ECL | Reserved |
| R-0 | W-0 | W-0 | W-0 | W-0 | W-0 | W-0 | R-0 |

Legend: R = Read only; W = Write; only; -*n* = value after reset

**Table 4-20 TCP3D Process 0 Error Interrupt Enable Clear Register Field Descriptions (Part 1 of 2)**

| Bit | Name | Value | Description |
|---|---|---|---|
| 31-29 | Reserved | N/A | Reserved |
| 28 | VD_WADDR_ECL | 1 | Clear VBUS_DMA undefined write address access error interrupt enable |
| 27 | VD_RADDR_ECL | 1 | Clear VBUS_DMA undefined read address access error interrupt enable |
| 26 | VC_WADDR_ECL | 1 | Clear VBUS_CFG undefined write address access error interrupt enable |
| 25 | VC_RADDR_ECL | 1 | Clear VBUS_CFG undefined read address access error interrupt enable |

**Table 4-20 TCP3D Process 0 Error Interrupt Enable Clear Register Field Descriptions (Part 2 of 2)**

| Bit | Name | Value | Description |
|---|---|---|---|
| 24 | DMA_OINTLV_ECL | 1 | Clear interleaver memory DMA transfer overflow error interrupt enable |
| 23 | DMA_OPAR1_ECL | 1 | Clear parity 1 data memory DMA transfer overflow error interrupt enable |
| 22 | DMA_OPAR0_ECL | 1 | Clear parity 0 data memory DMA transfer overflow error interrupt enable |
| 21 | DMA_OSYS_ECL | 1 | Clear systematic data memory DMA transfer overflow error interrupt enable |
| 20 | Reserved | N/A | Reserved |
| 19 | DMA_UPAR1_ECL | 1 | Clear parity 1 data memory DMA transfer underflow error interrupt enable |
| 18 | DMA_UPAR0_ECL | 1 | Clear parity 0 data memory DMA transfer underflow error interrupt enable |
| 17 | DMA_USYS_ECL | 1 | Clear systematic data memory DMA transfer underflow error interrupt enable |
| 16 | DMA_UCFG_ECL | 1 | Clear input configuration register DMA transfer underflow error interrupt enable |
| 15 | TRIG_ECL | 1 | Clear trigger error interrupt enable |
| 14 | MAX_MIN_IT_ECL | 1 | Clear max/min iteration error interrupt enable |
| 13-12 | Reserved | N/A | Reserved |
| 11 | IPROG_ACC_ECL | 1 | Clear TCP3D “in progress” memory access error interrupt enable |
| 10 | ICFG_ACC_ECL | 1 | Clear spurious TCP3D input configuration access error interrupt enable |
| 9 | IDAT_FMT_ECL | 1 | Clear input data format value error interrupt enable |
| 8 | SNR_VAL_ECL | 1 | Clear SNR threshold value error interrupt enable |
| 7 | Reserved | N/A | Reserved |
| 6 | SW0_LN_ECL | 1 | Clear SW0 length error interrupt enable |
| 5 | END_REG_WR_ECL | 1 | Clear endian register write error interrupt enable |
| 4 | SPLIT_TCP_ECL | 1 | Clear split TCP3D programming error interrupt enable |
| 3 | MODE_REG_WR_ECL | 1 | Clear mode register write error interrupt enable |
| 2 | INTLV_CONT_ECL | 1 | Clear interleaver contention error interrupt enable |
| 1 | BLK_LN_ECL | 1 | Clear code block length error interrupt enable |
| 0 | Reserved | 0 | Reserved (always reads 0) |
| **End of Table 4-20** | | | |

## 4.5 Common Interrupt Registers (VBUS CFG Bus)

### 4.5.1 TCP3D End of Interrupt Register (TCP3D_EOI)

The TCP3D End of Interrupt register supports the End of Interrupt (EOI) interface between the TCP3D and the external Interrupt Distributor Component. It is a common register that is used for servicing both of the TCP3D interrupts. This register is written by software to acknowledge the interrupt has been cleared so another interrupt of the same type can be generated by the Interrupt Distributor Component. This register is written after a TCP3D interrupt has been cleared via the associated Interrupt Clear register. It is described in Table 4-21. Writes to this register have no effect on the internal interrupt processing of the TCP3D.

Figure 4-14 shows the TCP3D End of Interrupt register layout and Table 4-21 describes the bits.

**Figure 4-14 TCP3D End of Interrupt Register (reset value 0x0000)**

| 31 8 | 7 0 |
|---|---|
| Reserved | EOI_VECTOR |
| R-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-21 TCP3D End of Interrupt Register Bit Description**

| Bit | Name | Description |
|---|---|---|
| 31-8 | Reserved | Reserved (always reads 0) |
| 7-0 | EOI_VECTOR | End of interrupt vector value that distinguishes which interrupt is being acknowledged. This value is output on the external *eoi_vector* interface of the TCP3D.<br>0x00-0xff = Supported values |
| **End of Table 4-21** | | |

## 4.6 Input Configuration Registers (VBUS DMA Bus)

The 32-bit input configuration registers are accessible via the 128-bit VBUSP_DMA interface, which results in 4 registers being accessed together. There are two complete sets of input configuration registers. The first set is to be used in conjunction with single-buffered mode, double-buffered mode process 0 or split mode process 0. The second set is to be used in conjunction with double-buffered mode process 1 or split mode process 1. The process 0 registers are labeled with a P0 suffix, while a P1 suffix is used for process 1 registers. The address range of each set of registers is shown below:

| | | | |
|---|---|---|---|
| 0x0_0000 | to | 0x0_003C | Process 0 |
| 0x0_0200 | to | 0x0_023C | Process 1 |

The register map tables shown below are generally the same for the process 0 and process 1 registers. So, only one table is shown for both and the address of the process 1 register appears below the table. Any differences between the process 0 and process 1 maps are noted.

**Note—**In the register descriptions that follow, the bit position of the fields within each register do not reflect the bit positions of these fields within the 128-bit VBUS_DMA interface. Additionally, big endian data swapping is supported for these registers on 32-bit boundaries as described in Section 3.3.9.1. This will cause the bit positions of the fields within the 128-bit interface to be swapped relative to the little endian positions. As an example, the *NUM_SW0* field in TCP3D_IC_CFG0_P0 (Table 4-22) is located in bits [5:0] of the 128-bit interface in little endian mode or bits [101:96] for big endian mode.

### 4.6.1 TCP3D Input Configuration Register 0 (TCP3D_IC_CFG0_P0/P1)

The TCP3D_IC_CFG0 input configuration register is described in Table 4-22.

Figure 4-15 shows the TCP3D Input Configuration register 0 layout and Table 4-22 describes the bits.

**Figure 4-15 TCP3D Input Configuration Register 0 (reset value 0x0000)**

| 31 29 | 28 16 | 15 6 | 5 0 |
|---|---|---|---|
| Reserved | BLK_LN | Reserved | NUM_SW0 |
| R-0 | R/W-0 | R-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-22 TCP3D Input Configuration Register 0 Field Descriptions**

| Bit | Name | Value | Description |
|---|---|---|---|
| 31-29 | Reserved | 0 | Reserved (always reads 0) |
| 28-16 | BLK_LN | 39 - 8191 | Block size -1, i.e. 40 - 8192 (should not include tail symbols). |
| 15-6 | Reserved | 0 | Reserved (always reads 0) |
| 5-0 | NUM_SW0 | 0 - 62 | Number of SW0s per sub-block; 0 = off (value of 63 is illegal) |
| **End of Table 4-22** | | | |

**Note—**Block size is represented in terms of code bits. The programmed block length = block_size - 1. For 3GPP, block size is the actual number of code bits, NOT the extended block size. For LTE and WiMAX, block size is always incremented by 8s (e.g. 40, 48, 56 … 6144). Programming non-conforming block lengths may cause unpredictable behavior.

## 4.6.2 TCP3D Input Configuration Register 1 (TCP3D_IC_CFG1_P0/P1)

The TCP3D_IC_CFG1 input configuration register is described in Table 4-23.

Figure 4-16 shows the TCP3D Input Configuration Register 1 layout and Table 4-23 describes the bits.

**Figure 4-16 TCP3D Input Config Register 1 (reset value 0x0000)**

| 31 15 | 14 8 | 7 6 | 5 4 | 3 | 2 0 |
|---|---|---|---|---|---|
| Reserved | SW1_LN | Reserved | SW2_LN_SEL | Reserved | SW0_LN_SEL |
| R-0 | R/W-0 | R-0 | R/W-0 | R-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-23 TCP3D Input Config Register 1 Bit Definitions**

| Bit | Name | Description |
|---|---|---|
| 31-15 | Reserved | Reserved (always reads 0) |
| 14-8 | SW1_LN | SW1 length:<br>0-8 = Unsupported Values<br>9 – 127 = 10 – 128 information bits |
| 7-6 | Reserved | Reserved (always reads 0) |
| 5-4 | SW2_LN_SEL | SW2 length:<br>0 = Off<br>1 = SW2_LN = SW1_LN<br>2 = SW2_LN = SW1_LN - 2<br>3 = Unsupported Value |
| 3 | Reserved | Reserved (always reads 0) |
| 2-0 | SW0_LN_SEL | Select length of SW0:<br>0 = 16 information bits<br>1 = 32 information bits<br>2 = 48 information bits<br>3 = 64 information bits<br>4 = 96 information bits<br>5 = 128 information bits<br>6-7 = Unsupported Values |
| **End of Table 4-23** | | |

## 4.6.3 TCP3D Input Configuration Register 2 (TCP3D_IC_CFG2_P0/P1)

The TCP3D_IC_CFG2 input configuration register is described in Table 4-24.

Figure 4-17 shows the TCP3D Input Configuration Register 2 layout and Table 4-24 describes the bits.

**Figure 4-17 TCP3D Input Config Register 2 (reset value 0x0000)**

| 31 | 27 | 26 | 25 | 24 | 23 | 22 |
|---|---|---|---|---|---|---|
| Reserved | | CRC_SEL | CRC_ITER_PASS | | STOP_SEL | |
| R-0 | | R/W-0 | R/W-0 | | R-0 | |

| 21 | 20 | 16 | 15 | 12 | 11 | 8 | 7 |
|---|---|---|---|---|---|---|---|
| SNR_REP | SNR_VAL | | MAX_ITR | | MIN_ITR | | SOFT_OUT_FMT |
| R/W-0 | R/W-0 | | R/W-0 | | R/W-0 | | R/W-0 |

| 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| SOFT_OUT_ORDER_SEL | SOFT_OUT_FLAG_EN | EXT_SCALE_EN | OUT_ORDER_SEL | OUT_FLAG_EN | MAXST_EN | INTER_LOAD_SEL |
| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-24 TCP3D Input Config Register 2 Field Descriptions (Part 1 of 2)**

| Bit | Name | Description |
|---|---|---|
| 31-27 | Reserved | Reserved (always reads 0) |
| 26 | CRC_SEL | LTE CRC polynomial select:<br>0 = LTE code block CRC $(1 + x^1 + x^5 + x^6 + x^{23} + x^{24})$<br>1 = LTE transport block CRC $(1 + x^1 + x^3 + x^4 + x^5 + x^6 + x^7 + x^{10} + x^{11} + x^{14} + x^{17} + x^{18} + x^{23} + x^{24})$ |
| 25-24 | CRC_ITER_PASS | Select number of consecutive LTE CRC matches as stopping criterion:<br>00 = 1 match<br>01 = 2 matches<br>10 = 3 matches<br>11 = 4 matches |
| 23-22 | STOP_SEL | Stopping criteria select:<br>00 = Max iteration only<br>01 = Max iteration or CRC (LTE only; same as 00 if not in LTE mode)<br>10 = Max iteration or SNR<br>11 = Max iteration or SNR |
| 21 | SNR_REP | SNR Reporting Enable (Always enabled if STOP_SEL = 10 or 11 regardless of this bit):<br>0 = SNR reporting disabled<br>1 = SNR reporting enabled |
| 20-16 | SNR_VAL | SNR threshold (in dB) used as a stopping criterion. The turbo decoding will stop as soon as the decoded signal SNR > SNR threshold.<br>0-20 = SNR threshold in dB |
| 15-12 | MAX_ITR | Maximum number of iterations:<br>0 - 15 = 1 - 15 iterations[1] |
| 11-8 | MIN_ITR | Minimum number of iterations:<br>0 - 15 = 0 - 15 iterations[2] |
| 7 | SOFT_OUT_FMT | Systematic, parity 1 and parity 0 soft output bit formats:<br>0 = Systematic soft output is truncated from 9 to 8-bits (divided by 2) and parity soft outputs are truncated from 11 to 10-bits and then saturated to 8-bits for storage in RAM.<br>1 = All soft outputs are saturated to 8 -bits for storage in RAM |

**Table 4-24 TCP3D Input Config Register 2 Field Descriptions (Part 2 of 2)**

| Bit | Name | Description |
|---|---|---|
| 6 | SOFT_OUT_ORDER_SEL | Format of soft output data in big endian mode. This bit has no effect if in little endian mode.<br>**Mode** **DSP** **TCP3D**<br>**Format** **Format**<br>------------------ - ----------------- ----------------------<br>0 = 32-bit packed 3,2,1,0, –> 15,14.13,12,<br>7,6,5,4, 11,10,9,8<br>11,10,9,8 7,6,5,4<br>15,14,13,12 3,2,1,0 (bytes)<br>1 = 8 bit native 0,1,2,…14,15 –>15,14,13,…2,1,0 (bytes) |
| 5 | SOFT_OUT_FLAG_EN | Soft output bits read flag:<br>0 = Soft output bits are not read via EDMA3.<br>1 = Soft output bits read via EDMA3. In all modes except split mode, all 3 soft output memories (systematic, parity 0, parity 1) must be read. In split mode, only the systematic output memory must be read. In WiMAX and 3GPP modes the parity 1 soft outputs are only valid after the second iteration. As such, MIN_ITR > 1 and MAX_ITR > 1 if this bit is set. |
| 4 | EXT_SCALE_EN | Extrinsic scale enable:<br>0 = Extrinsic scaling disabled (extrinsic values are used unscaled)<br>1 = Extrinsic scaling enabled (extrinsic values are scaled as in Table 4-34) |
| 3 | OUT_ORDER_SEL | Bit ordering of hard decisions:<br>0 = No swapping of output bit order<br>1 = Swap output bit order within each 32-bit word of 128-bit VBUS_DMA interface |
| 2 | OUT_FLAG_EN | Output status registers read flag:<br>0 = Output status registers are not read via EDMA3.<br>1 = Output status registers read via EDMA3 |
| 1 | MAXST_EN | Max Star enable:<br>0 = Max Star disabled<br>1 = Max Star enabled |
| 0 | INTER_LOAD_SEL | Interleaver table load select:<br>0 = Interleaver table will not be sent to the TCP3D or generated by one of the Interleaver Generators<br>1 = Interleaver table will be sent to the TCP3D or generated by one of the Interleaver Generators if the *ITG_EN* bit is set in **TCP3D_MODE**. |
| **End of Table 4-24** | | |

1. Entering a value of 0 for MAX_ITR results in the MAP decoder executing one iteration. It is the same as if the value had been set to 1. See Table 4-34 for other special cases.
2. See fTable 4-34 or special cases.

### 4.6.4 TCP3D Input Configuration Register 3 (TCP3D_IC_CFG3_P0/P1)

The TCP3D_IC_CFG3 input configuration register is described in Table 4-25. The suggested values for *MAXST_THOLD* and *MAXST_VALUE* are 1 and 0.5 respectively. They are stored in the same input format as the soft bits are stored in. For example, if the soft bits are in Q2 format, the stored values of *MAXST_THOLD* and *MAXST_VALUE* are 4 and 2, respectively.

Figure 4-18 shows the TCP3D Input Configuration Register 3 layout and Table 4-25 describes the bits.

**Figure 4-18 TCP3D Input Config Register 3 (reset value 0x0000)**

| 31 22 | 21 16 | 15 6 | 5 0 |
|---|---|---|---|
| Reserved | MAXST_VALUE | Reserved | MAXST_THOLD |
| R-0 | R/W-0 | R-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-25 TCP3D Input Config Register 3 Field Descriptions**

| Bit | Name | Value | Description |
|---|---|---|---|
| 31-22 | Reserved | 0 | Reserved (always reads 0) |
| 21-16 | MAXST_VALUE | 0-63 | Max Star value |
| 15-6 | Reserved | 0 | Reserved (always reads 0) |
| 5-0 | MAXST_THOLD | 0-63 | Max Star threshold |
| **End of Table 4-25** | | | |

### 4.6.5 TCP3D Input Configuration Register 4 (TCP3D_IC_CFG4_P0/P1)

The TCP3D_IC_CFG4 input configuration register sets the lower 4 starting states for beta for the first half of the iteration. These values will be used as the starting state for beta at the start of the last sliding window in a block. *MAXST_VALUE* are 4 and 2, respectively.

Figure 4-19 shows the TCP3D Input Configuration Register 4 layout and Table 4-26 describes the bits.

**Figure 4-19 TCP3D Input Config Register 4 (reset value 0x0000)**

| 31 24 | 23 16 | 15 8 | 7 0 |
|---|---|---|---|
| BETA_ST3_MAP0 | BETA_ST2_MAP0 | BETA_ST1_MAP0 | BETA_ST0_MAP0 |
| R/W-0 | R/W-0 | R/W-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset;

**Table 4-26 TCP3D Input Config Register 4 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31-24 | BETA_ST3_MAP0 | Beta state 3 for MAP0 |
| 23-16 | BETA_ST2_MAP0 | Beta state 2 for MAP0 |
| 15-8 | BETA_ST1_MAP0 | Beta state 1 for MAP0 |
| 7-0 | BETA_ST0_MAP0 | Beta state 0 for MAP0 |
| **End of Table 4-26** | | |

For TCP3D Input Config Register 4 P1, see Table 4-26.

### 4.6.6 TCP3D Input Configuration Register 5 (TCP3D_IC_CFG5_P0/P1)

The TCP3D_IC_CFG5 input configuration register sets the upper 4 starting states for beta for the first half of the iteration. These values will be used as the starting state for beta at the start of the last sliding window in a block. *MAXST_VALUE* are 4 and 2, respectively.

Figure 4-20 shows the TCP3D Input Configuration Register 5 layout and Table 4-27 describes the bits.

**Figure 4-20 TCP3D Input Config Register 5 (reset value 0x0000)**

| 31 24 | 23 16 | 15 8 | 7 0 |
|---|---|---|---|
| BETA_ST7_MAP0 | BETA_ST6_MAP0 | BETA_ST5_MAP0 | BETA_ST4_MAP0 |
| R/W-0 | R/W-0 | R/W-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-27 TCP3D Input Config Register 5 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31-24 | BETA_ST7_MAP0 | Beta state 7 for MAP0 |
| 23-16 | BETA_ST6_MAP0 | Beta state 6 for MAP0 |

**Table 4-27 TCP3D Input Config Register 5 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 15-8 | BETA_ST5_MAP0 | Beta state 5 for MAP0 |
| 7-0 | BETA_ST4_MAP0 | Beta state 4 for MAP0 |
| **End of Table 4-27** | | |

### 4.6.7 TCP3D Input Configuration Register 6 (TCP3D_IC_CFG6_P0/P1)

The TCP3D_IC_CFG6 input configuration register sets the lower 4 starting states for beta for the second half of the iteration. These values will be used as the starting state for beta at the start of the last sliding window in a block. *MAXST_VALUE* are 4 and 2, respectively.

Figure 4-21 shows the TCP3D Input Configuration Register 6 layout and Table 4-28 describes the bits.

**Figure 4-21 TCP3D Input Config Register 6 (reset value 0x0000)**

| 31 24 | 23 16 | 15 8 | 7 0 |
|---|---|---|---|
| BETA_ST3_MAP1 | BETA_ST2_MAP1 | BETA_ST1_MAP1 | BETA_ST0_MAP14 |
| R/W-0 | R/W-0 | R/W-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-28 TCP3D Input Config Register 6 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31-24 | BETA_ST3MAP1 | Beta state 3 for MAP1 |
| 23-16 | BETA_ST2_MAP1 | Beta state 2 for MAP1 |
| 15-8 | BETA_ST1_MAP1 | Beta state 1 for MAP1 |
| 7-0 | BETA_ST0_MAP1 | Beta state 0 for MAP1 |
| **End of Table 4-28** | | |

### 4.6.8 TCP3D Input Configuration Register 7 (TCP3D_IC_CFG7_P0/P1)

The TCP3D_IC_CFG7 input configuration register sets the upper 4 starting states for beta for the second half of the iteration. These values will be used as the starting state for beta at the start of the last sliding window in a block.

Figure 4-22 shows the TCP3D Input Configuration Register 7 layout and Table 4-29 describes the bits.

**Figure 4-22 TCP3D Input Config Register 7 (reset value 0x0000)**

| 31 24 | 23 16 | 15 8 | 7 0 |
|---|---|---|---|
| BETA_ST7_MAP1 | BETA_ST6_MAP1 | BETA_ST5_MAP1 | BETA_ST4_MAP1 |
| R/W-0 | R/W-0 | R/W-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-29 TCP3D Input Config Register 7 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31-24 | BETA_ST7MAP1 | Beta state 7for MAP1 |
| 23-16 | BETA_ST6_MAP1 | Beta state 6 for MAP1 |
| 15-8 | BETA_ST5_MAP1 | Beta state 5 for MAP1 |
| 7-0 | BETA_ST4_MAP1 | Beta state 4for MAP1 |
| **End of Table 4-29** | | |

### 4.6.9 TCP3D Input Configuration Register 8 (TCP3D_IC_CFG8_P0/P1)

The TCP3D_IC_CFG8 input configuration register is described in Table 4-30. This register sets the Extrinsic scale factor. The scale factors are in Q5 format so the value for a scale factor of 1 is 0x20.

Figure 4-23 shows the TCP3D Input Configuration Register 8 layout and Table 4-30 describes the bits.

**Figure 4-23 TCP3D Input Config Register 8 (reset value 0x0000)**

| | |
|---|---|
| 31 24 | 23 0 |
| Reserved | EXT_SCALE_0_3 |
| R-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-30 TCP3D Input Config Register 8 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31-24 | Reserved | Reserved (always reads 0) |
| 23-0 | EXT_SCALE_0_3 | Extrinsic scale factor:<br>5-0 = Extrinsic scale factor 0<br>11-6 = Extrinsic scale factor 1<br>17-12 = Extrinsic scale factor 2<br>23-18 = Extrinsic scale factor 3 |
| **End of Table 4-30** | | |

### 4.6.10 TCP3D Input Configuration Register 9 (TCP3D_IC_CFG9_P0/P1)

The TCP3D_IC_CFG9 input configuration register is described in Table 4-31. This register sets the Extrinsic scale factor. The scale factors are in Q5 format so the value for a scale factor of 1 is 0x20.

Figure 4-24 shows the TCP3D Input Configuration Register 9 layout and Table 4-31 describes the bits.

**Figure 4-24 TCP3D Input Config Register 9 (reset value 0x0000)**

| | |
|---|---|
| 31 24 | 23 0 |
| Reserved | EXT_SCALE_4_7 |
| R-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-31 TCP3D Input Config Register 9 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31-24 | Reserved | Reserved (always reads 0) |
| 23-0 | EXT_SCALE_4_7 | Extrinsic scale factor:<br>5-0 = Extrinsic scale factor 4<br>11-6 = Extrinsic scale factor 5<br>17-12 = Extrinsic scale factor 6<br>23-18 = Extrinsic scale factor 7 |
| **End of Table 4-31** | | |

### 4.6.11 TCP3D Input Configuration Register 10 (TCP3D_IC_CFG10_P0/P1)

The TCP3D_IC_CFG10 input configuration register is described in Table 4-32. This register sets the Extrinsic scale factor. The scale factors are in Q5 format so the value for a scale factor of 1 is 0x20.

Figure 4-25 shows the TCP3D Input Configuration Register 10 layout and Table 4-32 describes the bits.

**Figure 4-25 TCP3D Input Config Register 10 (reset value 0x0000)**

| 31 24 | 23 0 |
|---|---|
| Reserved | EXT_SCALE_8_11 |
| R-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-32 TCP3D Input Config Register 10 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31-24 | Reserved | Reserved (always reads 0) |
| 23-0 | EXT_SCALE_8 -11 | Extrinsic scale factor:<br>5-0 = Extrinsic scale factor 8<br>11-6 = Extrinsic scale factor 9<br>17-12 = Extrinsic scale factor 10<br>23-18 = Extrinsic scale factor 11 |
| **End of Table 4-32** | | |

### 4.6.12 TCP3D Input Configuration Register 11 (TCP3D_IC_CFG11_P0/P1)

The TCP3D_IC_CFG11 input configuration register is described in Table 4-33. This register sets the Extrinsic scale factor. The scale factors are in Q5 format so the value for a scale factor of 1 is 0x20.

Figure 4-26 shows the TCP3D Input Configuration Register 11 layout and Table 4-33 describes the bits.

**Figure 4-26 TCP3D Input Config Register 11 (reset value 0x0000)**

| 31 24 | 23 0 |
|---|---|
| Reserved | EXT_SCALE_12_15 |
| R-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-33 TCP3D Input Config Register 11 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31-24 | Reserved | Reserved (always reads 0) |
| 23-0 | EXT_SCALE_12_15 | Extrinsic scale factor:<br>5-0 = Extrinsic scale factor 12<br>11-6 = Extrinsic scale factor 13<br>17-12 = Extrinsic scale factor 14<br>23-18 = Extrinsic scale factor 15 |
| **End of Table 4-33** | | |

### 4.6.12.1 Extrinsic Scale Register Usage

The 16 extrinsic scale registers are 6 bits each and have a (1,5) fixed point precision. The unsigned fixed point numbers can range from 0.0 to 1.0. For example, 0.5 is equal to 0.10000 or 0x10 and 1.0 is equal to 1.00000 or 0x20.

The 16 extrinsic scale registers are selected depending on the iteration number and active MAP as shown in Table 4-34. MAP 0 is the non-interleaved MAP decode and MAP1 is the interleaved MAP decode.

**Table 4-34 Extrinsic Scale Registers**

| Iteration Number | MAP | Extrinsic Scaling Register |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 2 |
| 1 | 1 | 3 |
| 2 | 0 | 4 |
| 2 | 1 | 5 |
| 3 | 0 | 6 |
| 3 | 1 | 7 |
| 4 | 0 | 8 |
| 4 | 1 | 9 |
| 5 | 0 | 10 |
| 5 | 1 | 11 |
| 6 | 0 | 12 |
| 6 | 1 | 13 |
| 7 | 0 | 14 |
| 7 | 1 | 15 |
| 8 | 0 | 14 |
| 8 | 1 | 15 |
| - | - | - |
| - | - | - |
| 15 | 0 | 14 |
| 15 | 1 | 15 |

### 4.6.13 TCP3D Input Configuration Register 12 (TCP3D_IC_CFG12_P0/P1)

The TCP3D_IC_CFG12 input configuration register is described in Table 4-35. This register provides 2 of the constants required by the LTE and WiMAX Interleaver Generators. The field definitions are different for each ITG.

Figure 4-27 shows the TCP3D Input Configuration Register 12 layout and Table 4-35 describes the bits.

**Figure 4-27 TCP3D Input Config Register 12 (reset value 0x0000)**

| 31 29 | 28 16 | 15 13 | 12 0 |
|---|---|---|---|
| Reserved | ITG_PARAM1 | Reserved | ITG_PARAM0 |
| R-0 | R/W-0 | R-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-35 TCP3D Input Config Register 12 Field Descriptions (Part 1 of 2)**

| Bit | Name | Description |
|---|---|---|
| 31-29 | Reserved | Reserved (always reads 0) |
| 28-16 | ITG_PARAM1 | LTE ITG: g(0)<br>WiMAX ITG: Pinc |

**Table 4-35 TCP3D Input Config Register 12 Field Descriptions (Part 2 of 2)**

| Bit | Name | Description |
|---|---|---|
| 13-15 | Reserved | Reserved (always reads 0) |
| 12-0 | ITG_PARAM0 | LTE ITG: 2*f2 mod K<br>WiMAX ITG: Not used |
| **End of Table 4-35** | | |

### 4.6.14 TCP3D Input Configuration Register 13 (TCP3D_IC_CFG13_P0/P1)

The TCP3D_IC_CFG13 input configuration register is described in Table 4-36. This register provides 2 of the constants required by the LTE and WiMAX Interleaver Generators. The field definitions are different for each ITG.

Figure 4-28 shows the TCP3D Input Configuration Register 13 layout and Table 4-36 describes the bits.

**Figure 4-28 TCP3D Input Config Register 13 (reset value 0x0000)**

| 31 29 | 28 16 | 15 13 | 12 0 |
|---|---|---|---|
| Reserved | ITG_PARAM3 | Reserved | ITG_PARAM2 |
| R-0 | R/W-0 | R-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-36 TCP3D Input Config Register 13 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31-29 | Reserved | Reserved (always reads 0) |
| 28-16 | ITG_PARAM3 | LTE ITG: $\pi(K/2)$<br>WiMAX ITG: $\pi_2(0)$ |
| 13-15 | Reserved | Reserved (always reads 0) |
| 12-0 | ITG_PARAM2 | LTE ITG: $\pi(K/4)$<br>WiMAX ITG: $\pi_1(0)$ |
| **End of Table 4-36** | | |

### 4.6.15 TCP3D Input Configuration Register 14 (TCP3D_IC_CFG14_P0/P1)

The TCP3D_IC_CFG14 input configuration register is described in Table 4-37. This register provides 1 of the constants required by the LTE and WiMAX Interleaver Generators. The field definitions are different for each ITG.

Figure 4-29 shows the TCP3D Input Configuration Register 14 layout and Table 4-37 describes the bits.

**Figure 4-29 TCP3D Input Config Register 14 (reset value 0x0000)**

| 31 13 | 12 0 |
|---|---|
| Reserved | ITG_PARAM4 |
| R-0 | R/W-0 |

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

**Table 4-37 TCP3D Input Config Register 14**

| Bit | Name | Description |
|---|---|---|
| 31-13 | Reserved | Reserved (always reads 0) |
| 12-0 | ITG_PARAM4 | LTE: $\pi(3K/4)$<br>WiMAX: $\pi_3(0)$ |
| **End of Table 4-37** | | |

## 4.7 Trigger Control Register (VBUS_DMA Bus)

### 4.7.1 TCP3D Trigger Control Register (TCP3D_TRIG_P0/P1)

The TCP3D Trigger Control register is shown in Figure 4-30. It is used to initiate the decoding process once a code block of input data has been loaded into the TCP3D. Separate registers are provided for each process to support double-buffered and split modes. In single-buffered mode, only TCP_TRIG_P0 is used, and TCP3D_TRIG_P1 has no effect on the operation of the TCP3D. In double-buffered or split modes, both registers are used.

This is a write only register and writes to it create an internal trigger signal used by the TCP3D. The status of which process (0 or 1) is currently being executed is available in the *PROC_BUSY* bits of TCP3D_STS_P0 and TCP3D_STS_P1.

Figure 4-30 shows the TCP3D Trigger Control Register layout and Table 4-38 describes the bits.

**Figure 4-30 TCP3D Process 0 Trigger Control Register (reset value 0x0000)**

| 31 1 | 0 |
|---|---|
| Reserved | TRIG |
| W-0 | W-0 |

Legend: W = Write only; -*n* = value after reset

**Table 4-38 TCP3D Process 0 Trigger Control Register Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 0 | TRIG | Trigger to start new decode<br>0 = No trigger<br>1 = Trigger new decode |
| 31-1 | Reserved | Reserved |
| **End of Table 4-38** | | |

## 4.8 Output Status Registers (VBUS DMA Bus)

The 32-bit output status registers are accessible via the 128-bit VBUS_DMA interface, which results in all 3 registers being accessed at once. There are two complete sets of output status registers. The first set is to be used in conjunction with single-buffered mode, double-buffered mode or split mode process 0. The second set is to be used in conjunction split mode process 1. The process 0 registers are labeled with a P0 suffix, while a P1 suffix is used for process 1 registers. The address range of each set of registers is shown below:

- 0x0_0100 to 0x0_0108 Process 0
- 0x0_0300 to 0x0_0308 Process 1

The register map tables shown below are generally the same for the process 0 and process 1 registers. So, only one table is shown for both and the address of the process 1 register appears below the table. Any differences between the process 0 and process 1 maps are noted.

**Note—**In the register descriptions that follow the bit position of the fields within each register do not reflect the bit positions of these fields within the 128-bit VBUS_DMA interface. Additionally, big endian data swapping is supported for these registers on 32-bit boundaries as described in Section 3.3.9.1. This will cause the bit positions of the fields within the 128-bit interface to be swapped relative to the little endian positions. As an example, the *SNR_M1* field in TCP3D_OUT_STS0_P0 (Table 4-39) is located in bits [17:0] of the 128-bit interface in little endian mode or bits [113:96] for big endian mode.

### 4.8.1 TCP3D Output Status Register 0 (TCP3D_OUT_STS0_P0/P1)

The TCP3D output status register 0 (TCP3D_OUT_STS0) is described in Table 4-39.

Figure 4-31 shows the TCP3D Trigger Control Register layout and Table 4-39 describes the bits.

**Figure 4-31 TCP3D Output Status Register 0 (reset value 0x0000)**

| 31 28 | 27 24 | 23 18 | 17 0 |
|---|---|---|---|
| Reserved | FINAL_IT | Reserved | SNR_M1 |
| R-0 | R-0 | R-0 | R-0 |

Legend: R = Read only; -*n* = value after reset

**Table 4-39 TCP3D Output Status Register 0 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31-28 | Reserved | Reserved (always reads 0) |
| 27-24 | FINAL_IT | 0 - Fh = Number of decoded iterations. |
| 23-18 | Reserved | Reserved (always reads 0) |
| 17-0 | SNR_M1 | First moment of SNR calculations[1] |
| **End of Table 4-39** | | |

[1] First moment is the sum of the extrinsics

### 4.8.2 TCP3D Output Status Register 1 (TCP3D_OUT_STS1_P0/P1)

The TCP3D output status register 1 (TCP3D_OUT_CFG1) is described in Table 4-40.

Figure 4-32 shows the TCP3D Trigger Control Register layout and Table 4-40 describes the bits.

**Figure 4-32 TCP3D Output Status Register 1 (reset value 0x0000)**

| 31 30 | 29 | 28 27 22 | 21 0 |
|---|---|---|---|
| SNR_EXCEED | LTE_CRC_CHK | Reserved | SNR_M2 |
| R-0 | R-0 | R-0 | R-0 |

Legend: R = Read only; -*n* = value after reset

**Table 4-40 TCP3D Output Status Register 1 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31-30 | SNR_EXCEED | SNR status (Final status held)[1]:<br>00 = MAP0 failed SNR<br>01 = MAP0 passed SNR<br>This field is also active when the SNR_REP bit in TCP3D_IC_CFG2_P0 and TCP3D_IC_CFG_P1 is set to a “1”. |
| 29 | LTE_CRC_CHK | LTE CRC Check Status (Only supported for Process 0; Reserved for process 1):<br>0 = LTE CRC check failed or not in LTE mode<br>1 = LTE CRC check passed<br>This field is only active in LTE mode. |
| 28-22 | RSRVD | Reserved (always reads 0) |
| 21:0 | SNR_M2 | Second moment of SNR calculations[2] |
| **End of Table 4-40** | | |

[1] Bit 21 (SNR_EXCEED[1]) is always 0.

[2] Second moment is the sum of the square of the extrinsics

### 4.8.3 TCP3D Output Status Register 2 (TCP3D_OUT_STS2_P0/P1)

The TCP3D output status register 2 (TCP3D_OUT_CFG2) is described in Table 4-41.

Figure 4-33 shows the TCP3D Trigger Control Register layout and Table 4-41 describes the bits.

**Figure 4-33 TCP3D Output Status Register 2 (reset value 0x0000)**

| 31 30 | 29 16 | 15 14 | 13 0 |
|---|---|---|---|
| Reserved | CNT_CQI_ZERO | Reserved | CNT_CQI |
| R-0 | R-0 | | R-0 |

Legend: R = Read only; -*n* = value after reset

**Table 4-41 TCP3D Output Status Register 2 Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31-30 | Reserved | Reserved (always reads 0) |
| 29-16 | CNT_CQI_ZERO | CQI logic reports the number of systematics that have a value of zero (000000), which is ambiguous (i.e. does not imply a soft bit of either 0 or 1), here. |
| 15-14 | Reserved | Reserved (always reads 0) |
| 13-0 | CNT_CQI | Channel Quality Indicator measurement. Reports the number of mismatches between systematics and hard decisions. |
| **End of Table 4-41** | | |

# IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

| **Products** | | **Applications** | |
|---|---|---|---|
| Amplifiers | amplifier.ti.com | Audio | www.ti.com/audio |
| Data Converters | dataconverter.ti.com | Automotive | www.ti.com/automotive |
| DLP® Products | www.dlp.com | Communications and Telecom | www.ti.com/communications |
| DSP | dsp.ti.com | Computers and Peripherals | www.ti.com/computers |
| Clocks and Timers | www.ti.com/clocks | Consumer Electronics | www.ti.com/consumer-apps |
| Interface | interface.ti.com | Energy | www.ti.com/energy |
| Logic | logic.ti.com | Industrial | www.ti.com/industrial |
| Power Mgmt | power.ti.com | Medical | www.ti.com/medical |
| Microcontrollers | microcontroller.ti.com | Security | www.ti.com/security |
| RFID | www.ti-rfid.com | Space, Avionics & Defense | www.ti.com/space-avionics-defense |
| RF/IF and ZigBee® Solutions | www.ti.com/lprf | Video and Imaging | www.ti.com/video |
| | | Wireless | www.ti.com/wireless-apps |

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265