Product Specification

AHA4541

311 Mbits/sec Turbo Product Code Encoder/Decoder

This product is covered under multiple patents held or licensed by Comtech AHA Corporation.

This product is covered by a Turbo Code Patent License from France Telecom - TDF - Groupe des ecoles des telecommunications.

PS4541_1207

A subsidiary of Comtech Telecommunications Corporation

1126 Alturas Drive ■ Moscow ID 83843 ■ tel: 208.892.5600 ■ fax: 208.892.5601 ■ www.aha.com

Table of Contents

List of Figures

List of Tables

1.0 INTRODUCTION

The AHA4541 is a single-chip Turbo Product Code (TPC) Forward Error Correction (FEC) Encoder/Decoder capable of 311 Mbits/sec (OC-6) data rates. This device integrates both a TPC encoder and decoder and can be operated in a full duplex mode. In addition to TPC coding, support is included for helical interleaving, synchronization mark insertion and detection, CRC computation, scrambling, and higher order modulation symbol mapping. Figure 1 shows the functional block diagram.

The channel interface supports direct connection to various modulators and demodulators. Support for an arbitrary constellation mapping is included with external logic.

The encode path accepts byte-wide data, computes and inserts a CRC, and scrambles the data before TPC encoding. After the error correction code (ECC) bits are inserted by the encoder, the data is helically interleaved, and block synchronization marks are inserted to assist the decoder. Finally, the data is mapped according to the constellation and output from the device.

The decoder accepts input symbols via the demodulated in-phase (I) and quadrature (Q) components. An internal block synchronizer searches for synchronization marks, rotating the input symbol phase as necessary. After synchronization is achieved, the data is helically deinterleaved and decoded by the TPC decoder. The output of the decoder is descrambled, and the CRC is computed to verify data integrity. Decoded data is output in a parallel, byte-wide fashion.

Each of the above described functions can be disabled for support of user-defined data formatting schemes.

Internal circuitry enables the transfer rate across all ports, generating a constant, non-burst data flow. In addition, control of an external VCO can be used to generate data clocks, greatly simplifying system clocking issues.

Achieving the maximum channel rate of 360 Mbits/sec requires a DPCLK frequency of 180 MHz and a core voltage (Vdd) of 2.0 volts nominal. The maximum frequency of CCLK and DCLK is (DPCLK/2) = 90 MHz. For a core voltage of 1.8 volts nominal, DPCLK frequency reduces to 160 MHz and the channel rate reduces to 320 Mbits/sec.

1.1 FEATURES

CHANNEL INTERFACE:

- Accepts in-phase and quadrature (I & Q) inputs, up to 8 bits each
- High data rate input bus supports direct input of soft metrics at up to 4 soft metrics of 4 bits each
- Internal support for BPSK, QPSK, 8-PSK, 16-QAM, 64-QAM, and 256-QAM modulations
- Supports additional modulation formats with external logic
- Symbol rates up to 90 MSym/sec
- Channel rates up to 360 Mbits/sec
- Automatic phase ambiguity resolution
- Automatic mapping from input symbol to decoder soft metric
- Encoder and decoder pass through modes
- Built in synchronization mark insertion and detection for sync marks up to 32 bits

TPC ENCODER/DECODER:

- Block Sizes up to 16 Kbits
- Code Rates from 1/4 to 0.98
- Buffering to support variable iterations per block
- Payload rates of at least 311 Mbits/sec for all codes of rate 0.86 and above.
- Supports enhanced Turbo Product Codes
- Corrections count and averaging for channel SNR estimation

DATA INTERFACE:

- 8-bit Parallel Data Input/Output
- Programmable CRC Insertion and Checking up to 32 bits
- Support for packet level synchronization
- Flow control to generate constant data input/output.
- Support for external VCO to generate data clocks.

MICROPROCESSOR INTERFACE:

- Selectable microprocessor interface for Intel or Motorola processors

OTHER

- 3.3V I/O, 1.8V core operation
- RoHS compliant

1.2 GLOSSARY OF TERMS

Packet –
A group of user data bits.

Block (TPC) –
A TPC error correction block, including user data bits (1 or more packets), CRC bits, and TPC error correction bits.

Sync Mark/Bits –
Bits inserted in the data stream to allow location of the beginning of a TPC block.

Frame –
An entire group of data bits, including user data, CRC, error correction, and synchronization bits.

Channel Rate -
The bit rate output from the encoder, or input to the decoder, including all frame bits.

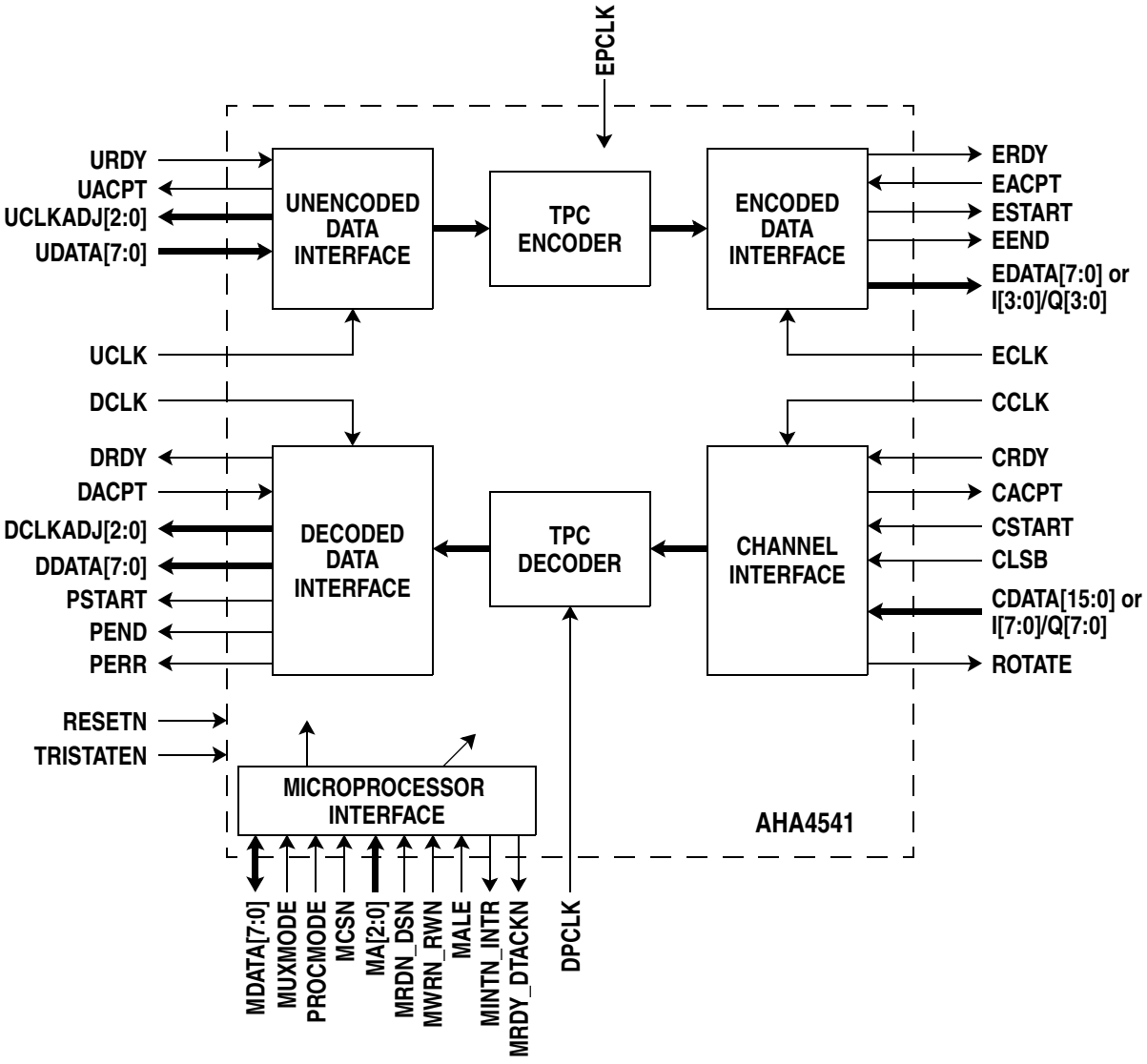
Data Rate -
The bit rate input to the encoder, or output from the decoder, including only user data bits. This is sometimes referred to as the payload data rate.

1.3 DOCUMENT CONVENTIONS

The following are formatting examples for specific document elements.

- **SIGNAL OR PINS** - Electrical connections available to the system.
- **Register Bit** - Bit(s) within a register. When the same register exists in both the encoder and decoder, an '**x**' is used to designate both register bits, as in **EBlockMode** and **DBlockMode**.
- **Interrupt** - Register bit that provides status.
- Hex values are represented with a prefix of "0x," such as Register "0x00." Binary values are represented with a prefix of "0b".

Figure 1: Functional Block Diagram



2.0 FRAME STRUCTURE

There are two frame structures supported by the AHA4541 decoder. Both contain synchronization marks, TPC error correction bits, CRC bits, and packet data. Figure 2 and Figure 3 show the frame structures supported by the device. Note that other frame structures are handled by disabling the various features within the device (frame synchronization, CRC, etc.), and handling framing in external logic.

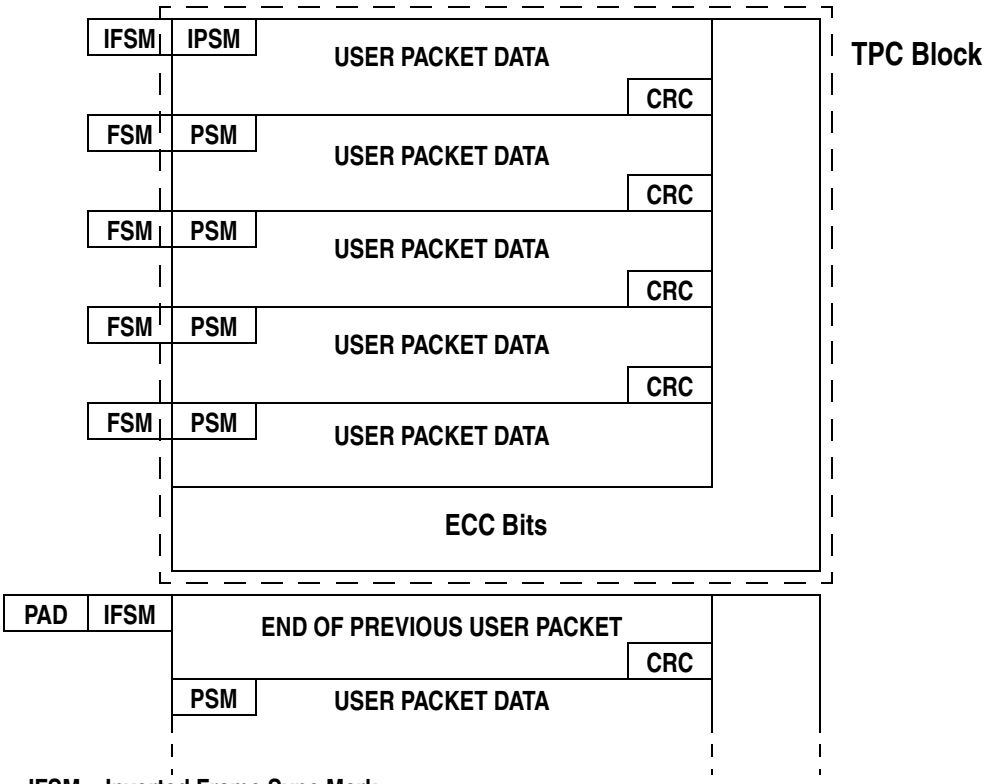
The construction of the frames will be described by how they are encoded. For each user packet input, a CRC of programmable length is computed over the data field. The resultant CRC value is appended to the data stream. If required, a packet synchronization mark is inserted into the data stream at the beginning of each packet. These packets, including sync and CRC are scrambled (randomized) by summing each bit with the output of a Pseudo Random Binary Sequence (PRBS) generator to ensure adequate bit transitions. The TPC encoder then computes ECC bits, which are inserted into the data stream at the appropriate locations. Finally, a series of block synchronization marks are added to the data stream. These marks will allow the decoder to detect the beginning of the TPC block. To decrease the required synchronization time, sync marks are placed throughout the TPC block, with inverted sync marks placed at the beginning of the block, as shown in Figure 2. Each of the above described functions can be disabled for support of user defined data formatting schemes.

Figure 2 gives a diagram of a 2-dimensional TPC frame structure with **xBlockMode** = 0. This structure (hereafter called Stream Mode) is optimum for streaming systems that do not require data packets to line up with ECC blocks. For example, broadcast video transmissions have an MPEG data packet size of 188 bytes. The ECC block size can be much larger than this packet size, giving better error correction performance. However, the CRC must be computed over each packet so that the MPEG packet error signal can be generated. Note that since the TPC data section may not be equal to an integer multiple number of data packets, a given packet may straddle two adjacent TPC blocks, as shown in the figure. The alignment of data packets is handled in the decoder by synchronizing to the packet sync marks. One drawback to this method is that extra sync marks are required to determine the beginning of a user data packet within the decoded frame (these marks are denoted as PSM in the figure). In addition, the scrambler is reset at the beginning of a group of packets. This is indicated by an inverted packet synchronization mark.

Figure 3 gives a diagram of the same TPC block with **xBlockMode** = 1. This structure (hereafter called Block Mode) is designed for block systems, such as systems that perform ARQ, or systems with frame level modulation (DMT). In these cases, a single data packet is encoded with CRC and TPC, and transmitted by itself. Using this mode, there is no need for packet synchronization marks, as the CRC and scrambler are reset at the beginning of each TPC block.

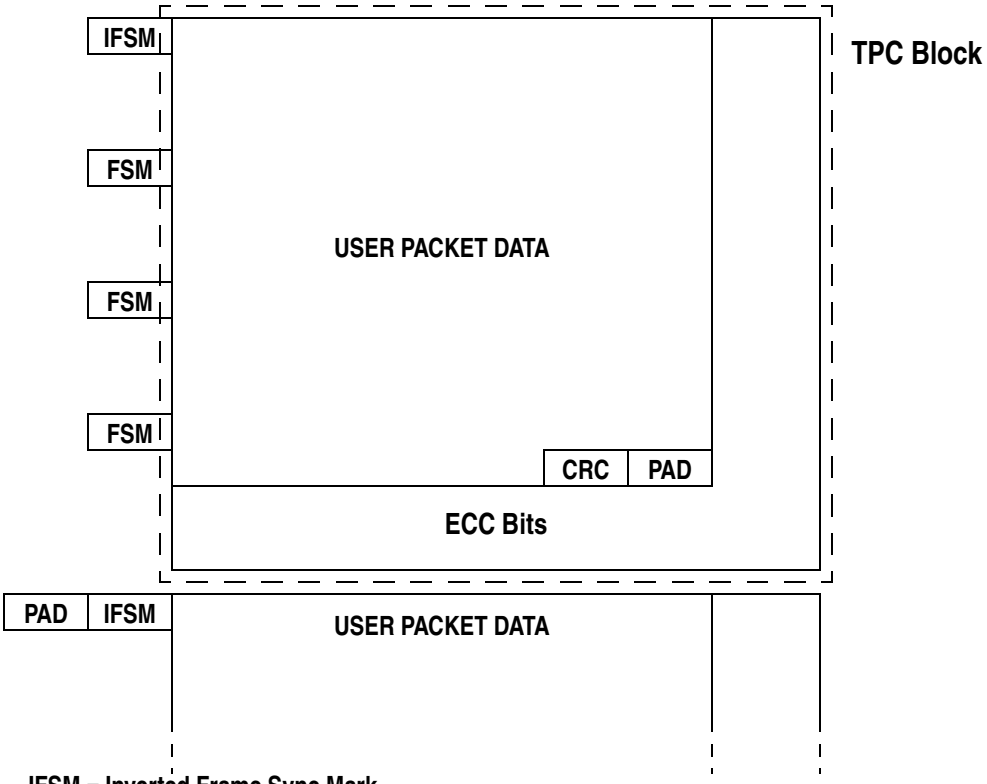
When using Block Mode, the user data section of the TPC block must be a multiple of 8 bits. However, depending on system issues, this constraint may not be met. In these cases, pad bits are inserted after the user data and CRC bits to fill out the entire uncoded TPC section. Similarly, with both frame modes, the length of the encoded TPC block may not line up with the synchronization mark configuration, requiring pad bits to be placed before the inverted sync mark of each block. All required pad bits are automatically inserted and removed by the device.

Figure 2: Frame Structure (xBlockMode = 0, Stream Mode)



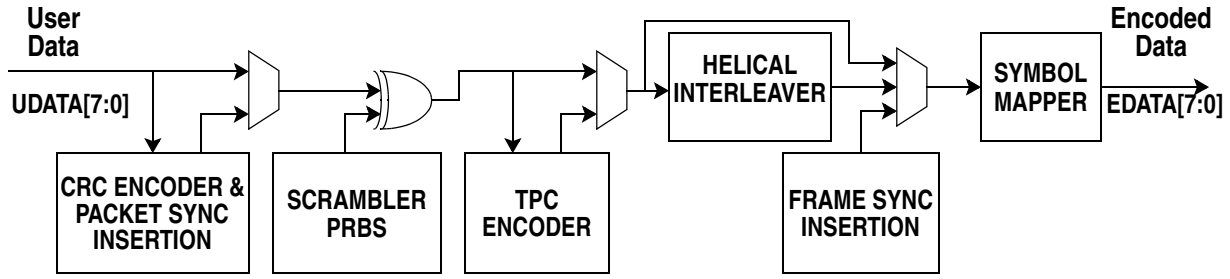
IFSM = Inverted Frame Sync Mark
FSM = Frame Sync Mark
IPSM = Inverted Packet Sync Mark
PSM = Packet Sync Mark

Figure 3: Frame Structure (xBlockMode = 1, Block Mode)



IFSM = Inverted Frame Sync Mark
FSM = Frame Sync Mark

Figure 4: Encoder Block Diagram



3.0 ENCODER

Figure 4 shows a block diagram of the encode path. Note that all of the blocks in the encode path except the helical interleaver operate by only inserting or modifying data in the stream. Therefore, the entire encode path has very low latency.

Data is input in a byte-wide fashion across the **UDATA[7:0]** bus. The first bit location in the TPC block comes in on UDATA[0]. The CRC engine computes a CRC over each packet of data which is inserted at the end of the packet. This data is scrambled by exclusive-ORing with the output of a Pseudo Random Binary Sequence (PRBS) generator. The scrambler ensures adequate bit transitions in the data stream, which are often required to allow improved DC balance and to accelerate clock recovery in the demodulator. The scrambled data is input to a TPC encoder, which computes ECC bits and inserts them at the appropriate locations in the data stream.

The helical interleaver improves burst error performance of the decoder. However, it adds a one block latency to the datapath. After interleaving, a programmable sync mark is inserted into the data stream. Finally, the bits are mapped to symbols according to the programmed constellation. The symbols are output over the **EDATA[7:0]** bus, at a rate of one symbol per clock.

The frame structure for the encoder is defined by **EBlockMode**. When cleared, the encoder operates in Stream Mode, according to Figure 2. When set, the encoder operates in Block Mode, according to Figure 3.

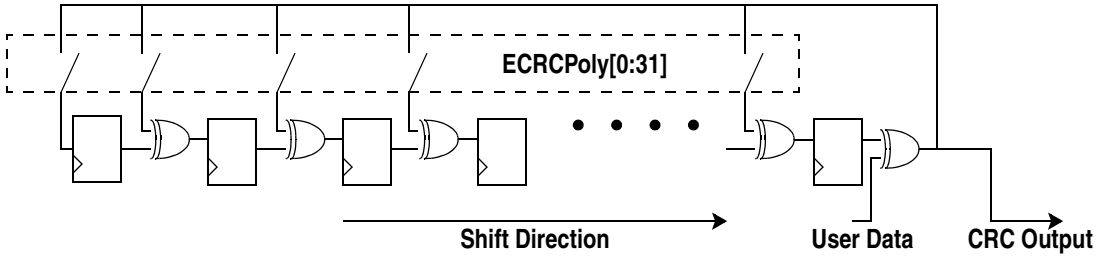
3.1 DATA INPUT AND OUTPUT

The AHA4541 supports many data flow control features, described in detail in Section 5.0. All data transfers into the encoder are synchronous to the unencoded data clock, **UCLK**. When required, handshaking is accomplished via a fully synchronous, ready/accept handshake. Data is transferred across the **UDATA[7:0]** interface on the rising edge of **UCLK** when both **URDY** and **UACPT** are asserted. Data transfers out of the encoder are synchronous to the encoded data clock, **ECLK**. Data is transferred across the **EDATA[7:0]** interface on the rising edge of clock when both **ERDY** and **EACPT** are asserted. The **ESTART** signal is asserted with the first symbol transfer of each frame. This signal is handshaked with the associated signal; i.e., it is asserted when **ERDY** is asserted with the first transfer, and deasserts the clock after the first symbol is handshaked.

3.2 CRC ENCODER

The cyclic redundancy check (CRC) encoder is a 32 bit linear feedback shift register, with a programmable polynomial. Figure 5 shows a diagram of the shift register. Each user data packet has a separate CRC encoded, with the resultant CRC bits appended to the packet. When using Stream Mode, a sync mark must exist in the data stream (**EPSyncEnable**=1) at the decoder so that packet synchronization can be achieved. This sync mark may either be inserted by the encoder, or it may be generated externally and input with the data stream. When **EBlockMode** = 1, there is one CRC computed for each TPC block.

Figure 5: CRC Encoder



If a packet synchronization mark insertion is required, the **EPSyncInsert** bit is set to a 1. If the data already contains a synchronization mark at the beginning of each packet, the CRC encoder must ignore this during encoding. In this case, the **EPSyncInsert** bit is set to a 0. In both cases, the sync mark can be 4 to 16 bits in length, as configured by **EPSyncLength**. If inserting sync marks, the value of the sync mark is written into the **EPSyncMark** register. Note that the scrambler does not scramble the packet synchronization marks.

The polynomial for the CRC encoder is written into the **ECRCPoly** register. The highest order of the polynomial (defined by **CRCLength**) is assumed to be a 1, and bit zero of the polynomial register corresponds to the 0th order of the polynomial, as shown in Figure 5. The desired size of the CRC, in bits, is written into the **ECRCSize** register. Table 1 gives a suggested list of polynomials for various length CRCs. The polynomial field gives the value to program into the **ECRCPoly** register. The detection capability gives the probability that an incorrect block will not be detected and flagged as incorrect by the CRC decoder.

Table 1: Recommended CRC Polynomials

CRC	*SIZE (bits)*	*POLY Program Value (hex)**	*DETECTION CAPABILITY***
4	4	1f	0.9375
8	8	1d5	0.99609
12	12	180f	0.999756
ANSI	16	18005	0.999985
CCITT	16	11021	0.999985
SDLC	16	1a097	0.999985
24	24	1805101	0.9999999404
32A	32	1404098e2	0.99999999953
32B	32	104c11db7	0.99999999977

Notes:

* *The leading '1' in these values is assumed by the device, and does not need to be written to the register.*

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

The shift register is reset at the beginning of each user packet. Data from the packet (excluding the sync mark, if present) is shifted into the circuit, until the number of bytes programmed into **EPSize** is reached. Note that the value written into **EPSize** does not include the packet sync mark or CRC bits. After the entire packet is shifted into the CRC encoder, the data input and feedback of the CRC shifter are disabled, and the contents of the CRC registers are shifted out and inserted into the data stream.

When using Block Mode, the sum of the packet data (**EPSize** x 8) and the CRC bits (**ECRCSize**) should equal the data size of the TPC packet. This can accomplished via shortening the TPC block. If this is not possible, the AHA4541 will automatically pad any bits remaining in the block with zeros. For example, if the packet size plus CRC bits is 300 bytes (2400 bits), and the closest achievable TPC data size is 2402 bits, the device will pad zeros in the remaining 2 bit location. Note that these 0 pad bits are inserted before scrambling so as to not affect the required bit transitions.

3.3 SCRAMBLER

The scrambler is built with a 16 bit pseudo-random binary sequence generator with programmable polynomial, length, and initialization seed. Figure 6 shows the configuration of the scrambler. The shift register is clocked once for each bit. As shown in the figure, the output of the shift register is exclusive-ORed with the data to be scrambled.

Figure 6: Scrambler

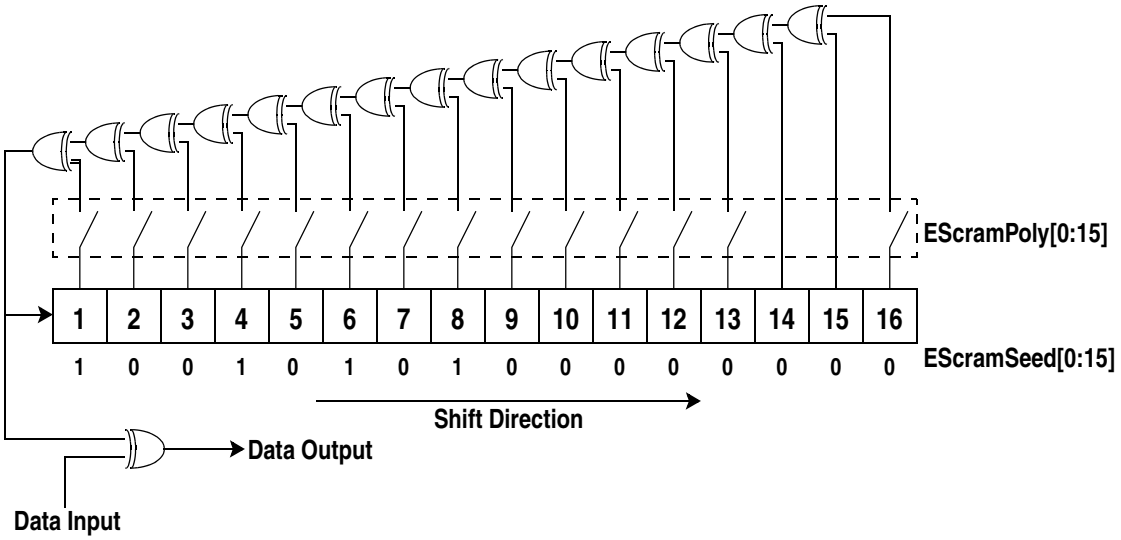


Figure 6 shows an example configuration for the scrambler that follows the DVB standard. The generator polynomial for this sequence is:

$$1 + X^{14} + X^{15}$$

This sequence is programmed into the **EScramPoly** register as 0b0110000000000000. The seed for the shift register that is shown in the diagram is programmed into **EScramSeed** as 0b0000000010101001. Every time the scrambler is reset, it is initialized with this seed value.

Note that the length of the generator is directly determined by the polynomial configuration. The sequence shown here is generated by a 15 stage shift register. Since the highest order of the polynomial is 15, the 16th bit XOR feedback will be disabled, and only 15 bits of the shift register are used. Also note that the 0th order of the polynomial is assumed to be 1.

The scrambler is reset at the beginning of every **EScramPer** packets, where a value of 0 represents 16. The packet synchronization marks at the beginning of every **EScramPer** packets are inverted to allow the descrambler to reset at the same point in the data stream as the scrambler. The first bit out of the PRBS is applied to the first data bit after the inverted packet synchronization mark. This inversion does not occur when **EScramPer** is set to 1, since the scrambler is reset at every sync mark. When using Block Mode or when the scrambler is disabled, **EScramPer** should by set to 1. Note that the packet synchronization marks are not scrambled by the scrambler. However, when **EPSyncInsert** = 0, the PRBS shift register continues to shift over the synchronization marks. When **EPSyncInsert** = 1, the PRBS does not shift over the synchronization marks.

3.4 TPC ENCODER

The TPC encoder supports two or three dimensional codes, with constituent code lengths of up to 128 bits, and overall block size up to 16 K bits. The encoder supports both extended-Hamming and parity only constituent codes, and supports enhanced TPCs. See Section 4.3 *TPC Decoder* for a description of supported codes and shortening configurations.

3.5 MODULATION SYMBOL INTERLEAVING

Modulation symbol interleaving is used to spread the less confident bits in a modulation symbol across all axes of the code so that these less confident bits do not line up in one row, column, or z-column. When using 2D or 3D codes without helical interleaving, **EModRowRotate** should be set if the shortened X-axis length is a multiple of the symbol size. When using 2D codes with helical interleaving, **EModRowRotate** should be set if the shortened Y-axis length (including the hyper row, if enabled) is a multiple of the symbol size. When using 3D codes with helical interleaving, **EModRowRotate** should be set if the shortened Z-axis length (including the hyper plane, if enabled) is a multiple of the symbol size.

When using 3D codes without helical interleaving, **EModPlaneRotate** should be set if the product of the shortened X-axis length and the Y-axis length is a multiple of the symbol size. When using 3D codes with helical interleaving, **EModPlaneRotate** should be set if the product of the shortened Z-axis length (including the hyper plane, if enable) and the shortened X-axis length is a multiple of the symbol size.

3.6 HELICAL INTERLEAVER

Helical interleaving transmits data in a helical fashion. When the channel introduces a burst of errors, the helical deinterleaver in the decoder will spread these errors across all axes of the code. The use of helical interleaving greatly increases the burst error correcting capability of the code.

The helical interleaver is enabled by setting **EHelical** to a 1. When helical interleaving is enabled, the **xShortX**, **xShortB**, and **xShortR** values must be set to 0. For 3-dimensional codes, **xShortY** must also be set to 0. This constrains the shortening resolution to one row for 2-dimensional codes, and one plane for 3-dimensional codes.

Helical interleaving is applied along a diagonal path through the encoded block. Data is output along diagonal lines from the upper left to lower right corner (for a 2D code). The first diagonal output starts with the bit row 1, column 1 followed by the diagonal starting at row 1, column 2. For 3D codes, instead of reading diagonally through the 2D array, interleaving reads diagonally through a cube of data.

The example below shows how interleaving is applied for a 2D (64,57)x(64,57) code.

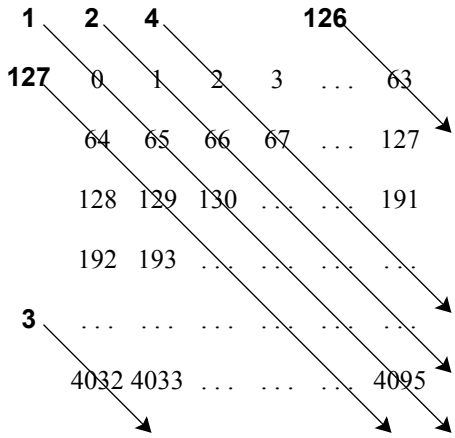
Figure 7: Input Block

0	1	2	3	. . .	63
64	65	66	67	. . .	127
128	129	130	. . .	. . .	191
192	193	. . .	. . .	. . .	. . .
. . .	. . .	. . .	. . .	. . .	. . .
4032	4033	. . .	. . .	. . .	4095

Note: The number reflects the bit order, including generated ECC bits.

The encoded, interleaved data output is taken along diagonal lines starting with bit 0 as shown below. The order of the interleaving is noted for each diagonal line.

Figure 8: 2D Interleaving



For the (64,57)x(64,57) block, the data output from the AHA4541 is: 0, 65, 130, ..., 4095, 1, 66, ..., 4031, 4032, 2, 67, ..., ..., 63, 64, ..., 4094 for a total of 4096 bits output. The Astro OC-3 operating as a decoder deinterleaves the block to restore it to its original order.

Figure 9: Encoded/Interleaved Data Output

0	65	130	. . .	4030	4095
1	66	131	. . .	4031	4032
2	67	132	. . .	3968	4033
3	68	. . .	. . .	. . .	. . .
. . .	. . .	. . .	. . .	. . .	. . .
63	64	129	. . .	4029	4094

Data bits are output from the encoder in row order from left to right. 3D interleaving/ deinterleaving is done by reading/writing cells diagonally through the x, y, and z dimensions.

3.7 FRAME SYNC MARK INSERTION

In order for the decoder to acquire block synchronization, a programmable sync mark must be inserted into the data stream before transmission over the channel. The AHA4541 supports synchronization marks up to 32 bits in length. Synchronization marks are placed at the beginning of each TPC block. In addition, sync marks may be placed throughout the block, with inverted sync marks placed at the beginning of each TPC block. This accelerates the synchronization process when using large TPC block sizes.

The sync mark length is written into the **EFSyncLength** register. A value of 0 disables sync mark insertion. The sync period is the number of bits from the start of a sync mark to the start of the next sync mark. This is written into **EFSyncPer**.

Sync marks can be placed throughout a TPC block, with inverted sync marks placed at the beginning of the TPC block. When using this feature, the total number of sync periods containing one inverted sync mark is written into **EFSyncFreq**. If only one sync mark is to be used per block, **EFSyncFreq** is written to 1. In this case, the sync mark at the beginning of the block is a non-inverted mark.

EFSyncPer × **EFSyncFreq** should be set equal to or greater than the TPC block size. If it is greater than the TPC block size, the encoder will automatically pad the remaining locations with zeros. Note that these pad bits are not scrambled, and therefore the number of pad bits should be kept to a minimum.

3.8 SYMBOL MAPPER

Symbol data is handshaked out of the device over the **EDATA[7:0]** bus. The format for this output bus is fully configurable using an internal lookup table. Data out of the encoder is grouped into symbols of size **ESymSize** bits. If the symbol map is disabled, each symbol is output to the **EDATA[ESymSize-1**:**0]** bus.

The symbol mapper is enabled by setting **ESymMapEnable**. The symbol map lookup table is a 256 x 8 bit lookup table. Each location in the table can be programmed to a user defined value. After encoding and grouping into symbols, a lookup is made into the symbol mapper table. The value stored at the addressed location is then output to the **EDATA[7:0]** bus. For example, when using 16 QAM, **EDATA[7:4]** can be configured as the I component, and **EDATA[3:0]** can be configured as the Q component.

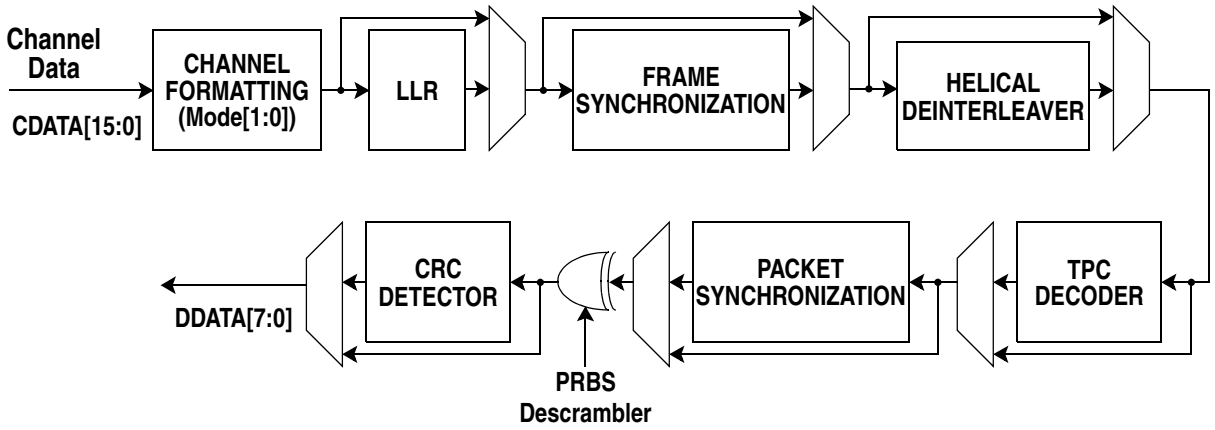
When using a QAM modulator that has its own internal symbol mapper that does not match what the AHA4541 decoder expects, the symbol map table can be used to translate between what the modulator produces and what the AHA4541 expects. It can also be used to 'pre-distort' the signal constellations for non-linear channels.

The symbol map table is programmed by writing the value for each location into the **ESymMap** register. The address written is determined by an internal pointer that is reset to the beginning of the table at power-up, or by writing the **ResetMapPtr** bit to a 1. The pointer increments for each write to the **ESymMap** register.

4.0 DECODER

The decode path of the AHA4541 includes a counterpart for each encode module as shown in Figure 10. The encoder and decoder are isolated paths. This allows full duplex operation, where the encoder and decoder are operating with different frame structures, code types, and data rates.

Figure 10: Decoder Block Diagram



4.1 DATA INPUT AND OUTPUT

The AHA4541 supports many data flow control features, described in detail in Section 5.0. The decoder channel input bus is synchronous to **CCLK**. Data is transferred across the **CDATA[15:0]** bus on the rising edge of **CCLK** when both **CRDY** and **CACPT** are asserted. The decoded data output bus is synchronous to **DCLK**. Data is transferred across the **DDATA[7:0]** bus on the rising edge of **DCLK** when both **DRDY** and **DACPT** are asserted.

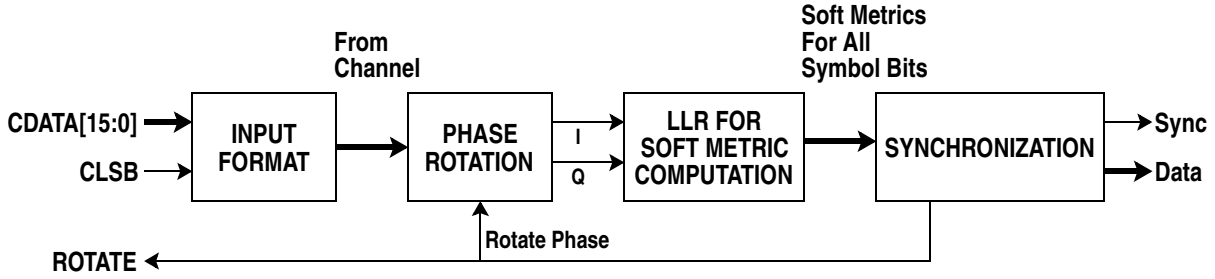
4.2 CHANNEL INTERFACE

The channel interface formats the channel data for decoding by the Turbo Product Code decoder. For best decoder performance, soft (confidence) information from the channel is necessary. When using BPSK/QPSK, this information comes directly from the in-phase (I) or quadrature (Q) component of the received symbol. However, when using higher-order modulations, the soft metrics for each bit in the constellation must be computed. This is accomplished using the Log-Likelihood Ratio (LLR). In addition to soft metric generation, the TPC decoder must know the location of the first bit of a TPC block. This is accomplished in the channel interface by searching through the input bit stream for the predefined synchronization marks.

The channel interface is designed to connect directly to the in-phase and quadrature (I & Q) outputs of a demodulator for internal soft metric computation. These inputs must be digitized, either with the use of a digital demodulator, or by an external Analog to Digital (A/D) Convertor. Alternately, metric computation can be done externally, in which case the internal computation is bypassed.

The channel interface is broken up into four major functions: channel input formatting, input symbol rotation, soft metric computation, and synchronization. Figure 11 shows a block diagram of the channel interface.

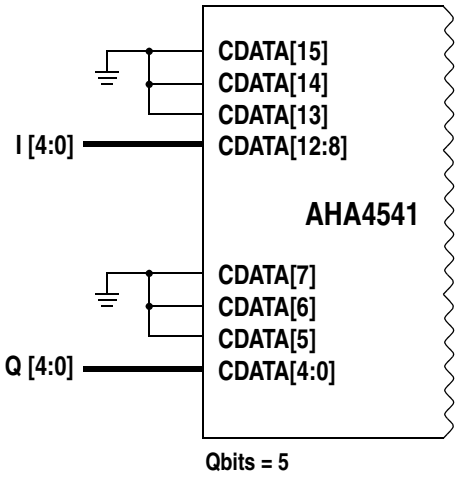
Figure 11: Channel Interface Block Diagram



4.2.1 CHANNEL INPUT FORMATTING

There are four modes of input formatting, defined by **ChannelFormat[1:0]**. Mode 0 (**ChannelFormat** = 0b00) is used when the modulation format is 8-PSK, 16-QAM, 64-QAM, or 256-QAM, and the internal LLR generation logic is enabled. The data input is expected to be one in-phase (I) and one quadrature (Q) channel. The number format is determined by **QMode**. The **DSymSize** register must be set to according to the chosen modulation. The number of bits input for each channel (I or Q) is written into **QBit.** The following diagram shows the wiring for mode 0.

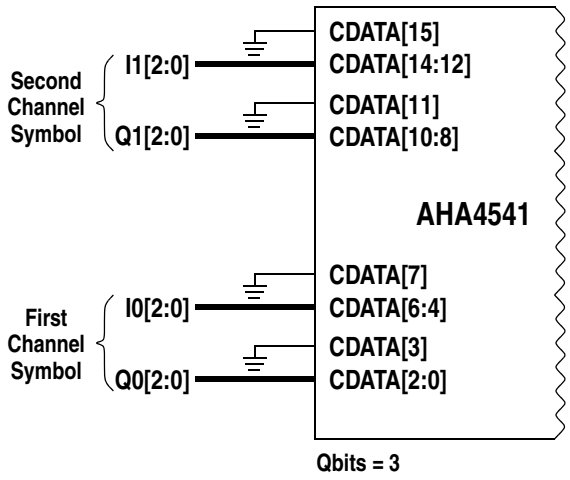
Figure 12: Mode 0 or Mode 1 Input Data Wiring



Mode 1 (**ChannelFormat** = 0b01) is identical to mode 0, but is used for BPSK or QPSK modulation. The wiring is shown in Figure 12 for the case where Qbits is set to 5. If higher data rates are required, mode 3 should be used for binary (BPSK) modulation formats, and mode 2 for QPSK modulation formats.

Mode 2 (**ChannelFormat** = 0b10) is used for high data rate QPSK systems. The data input is expected to be the in-phase (I) and quadrature (Q) components of two channel symbols. The number format is determined by **QMode**. The **DSymSize** register must be set to 0b011. The number of bits input for each channel is written into **QBit**, with a maximum of 4 bits per channel. Figure 13 shows the wiring for mode 2.

Figure 13: Mode 2 Input Data Wiring



Mode 3 (**ChannelFormat** = 0b11) is used for higher order modulations with external soft metric computation and symbol rotation. The number format and bits are configured by **QMode** and **QBits**, respectively. In this mode, the maximum number of soft bits per metric is 4. The number of data bits per constellation symbol is written into the **DSymSize** register. For modulations up to 4 bits per symbol, all metrics for the symbol are input with one transfer. The wiring for this case is shown in Figure 14 with 4 bits per symbol.

Figure 14: Mode 3 Input Data Wiring Up to 4 Bits Per Symbol

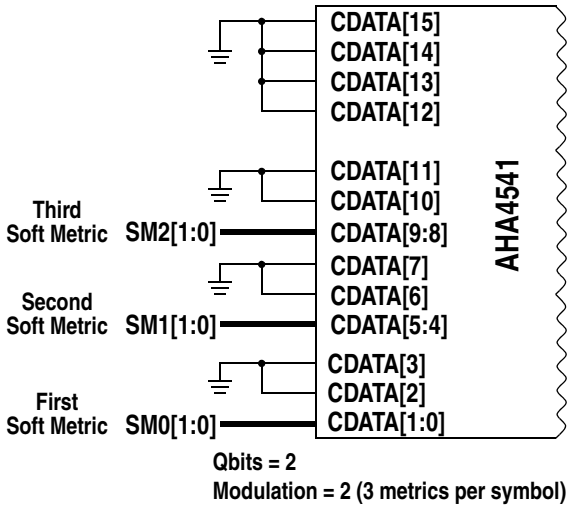
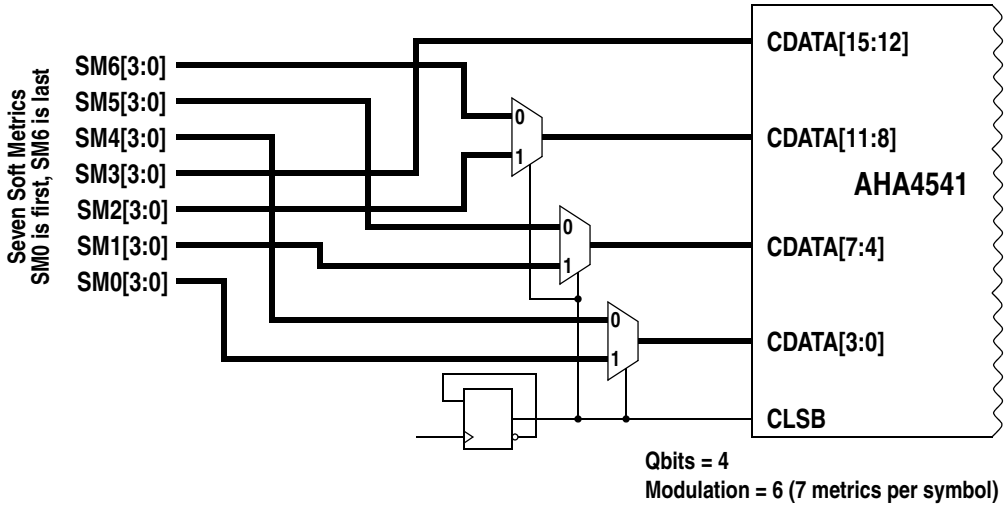


Figure 15: Mode 3 Input Data Wiring Greater than 4 Bits Per Symbol



For modulations greater than 4 bits per symbol, the metrics for each symbol are input to the device in two consecutive transfers. The first transfer loads the least significant metrics for the symbol, while the second transfer loads the most significant metrics for the symbol. To eliminate any ambiguity between LSB versus USB transfers, the **CLSB** signal must be asserted when the data on the **CDATA[15:0]** is the least significant metrics of the symbol. When using 6 or 8 bits per symbol, half of the metrics in the symbol are transferred with the LSB, and half are transferred with the MSB. When using 5 bits per symbol, 3 metrics are transferred with the LSB, while 2 metrics are transferred with the MSB. Similarly, when using 7 bits per symbol, 4 metrics are transferred with the LSB, and 3 metrics are transferred with the MSB. This is shown in Figure 15.

4.2.2 INPUT SYMBOL ROTATION

Phase Shift Keying modulation formats result in a phase ambiguity at the output of the demodulator. BPSK demodulators will produce either the correct phase output, or a 180 degree (inverted) output. QPSK and QAM demodulators will output 0, 90, 180, or 270 degree phase rotations, and 8-PSK demodulators will output rotations of 45 degree increments.

The input symbols must be rotated to the correct phase before decoding. The AHA4541 uses the following algorithm to determine phase rotation:

1) Attempt synchronization with 0 degree rotation.
2) If synchronization is detected with this phase rotation, immediately begin decoding.
3) Wait for **FSyncTime** sync mark periods without detecting synchronization, then rotate the phase by one step (90 degrees for QPSK, 45 degrees for 8-PSK).
4) Repeat steps 2 & 3 until synchronization is achieved.

After synchronization occurs, the current phase rotation of the incoming stream can be read from the **Rotation** register bits.

The **RotateEnable** bit determines if rotation is handled internally. It must be set to zero when **ChannelFormat** = 0b11. When the LLR computation is handled externally, the **ROTATE** signal is asserted for one **CCLK** cycle each time the synchronization block reaches **FSyncTime** without detecting synchronization. This can be used by external logic to rotate the phase. Note that the synchronizer can be configured to automatically sync to an inverted bit stream, regardless of the **RotateEnable** setting.

4.2.3 SOFT METRIC COMPUTATION

Data out of the TPC encoder is grouped into m bits, where m is the number of bits per symbol. This group of bits is mapped to a symbol and transmitted over the channel. Figure 16 shows the constellation map for QPSK symbols, and Figure 17 shows the constellation map for 8-PSK symbols. Figures 19, 20 and 21 show the constellation map for 16, 64 and 256-QAM respectively. When grouping bits before mapping, the first bit of a block becomes the LSB of the constellation symbol, while the m'th bit becomes the MSB of the constellation symbol.

After derotation of the constellation, the device converts the input symbol I & Q to a soft metric for each bit in the symbol. For BPSK and QPSK, no conversion is necessary, as the input values map directly to one or two soft metrics, respectively. When using 8-PSK, 16-QAM, 64-QAM, or 256-QAM, the Log-Likelihood Ratio is computed, generating 3, 4, 6 or 8 soft metrics, respectively.

For 8-PSK, if the demodulator does not align the incoming signals at 22.5 degrees and 67.5 degrees in each quadrant, according to Figure 17, but instead aligns the symbols with the axis (0 degrees and 45 degrees, according to Figure 18), then the **PreRotate** control bit must be set. This causes the device to first rotate the incoming symbol by $\Pi/8$ (22.5 degrees) before processing.

Figure 16: QPSK Symbol Definition

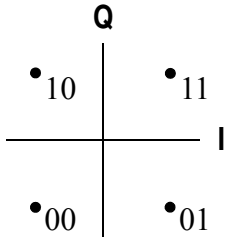


Figure 17: 8-PSK Symbol Definition (PreRotate=0)

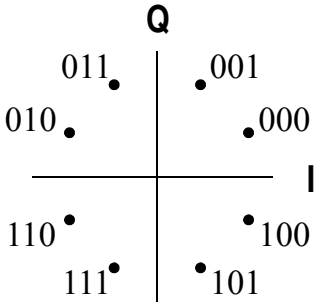


Figure 18: 8-PSK Symbol Definition (PreRotate=1)

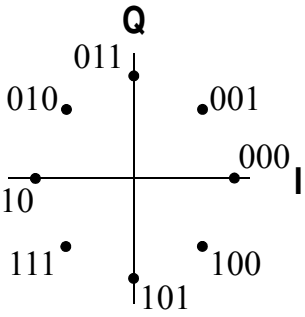


Figure 19: 16 QAM Symbol Definition

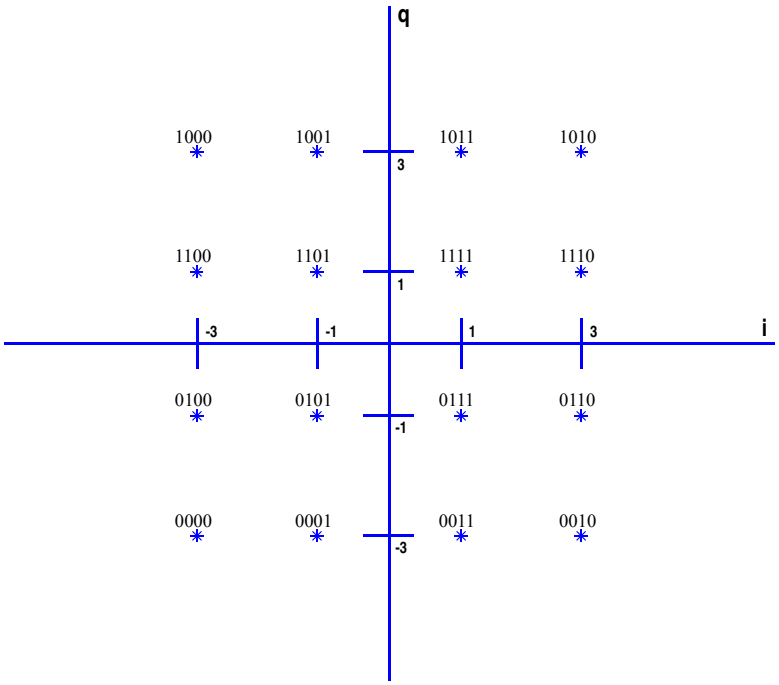


Figure 20: 64 QAM Symbol Definition

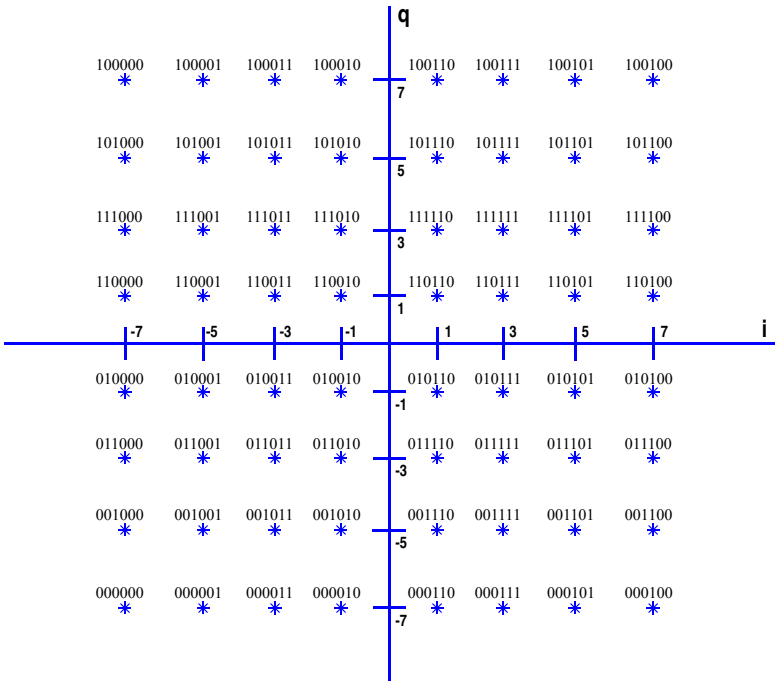
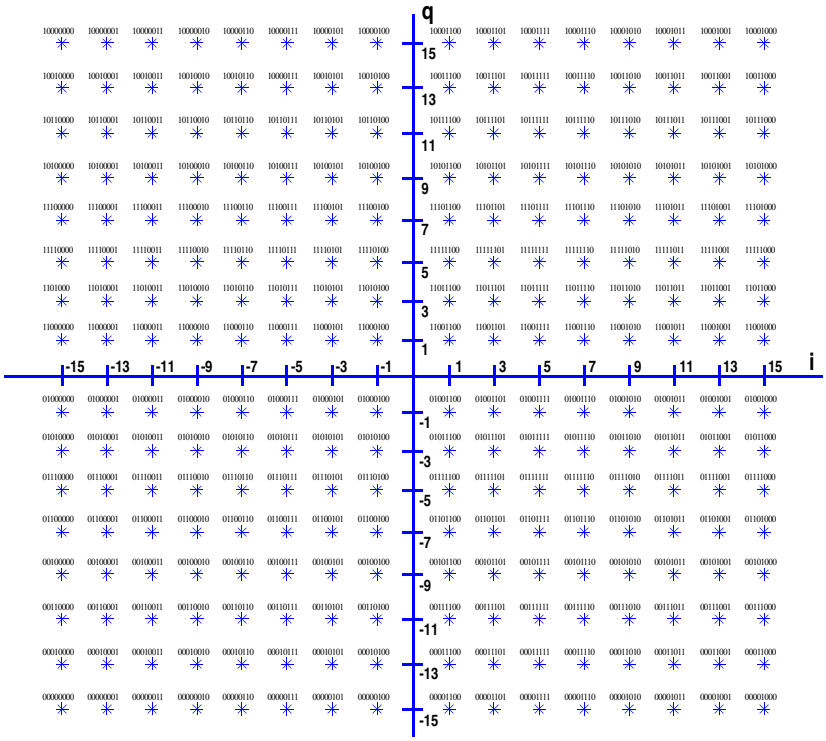


Figure 21: 256 QAM Symbol Definition



4.2.4 LLR NORMALIZATION

When using internal LLR computation with 8-PSK, 16-QAM, 64-QAM, or 256-QAM, the position of the constellation points relative to the input quantization range must be input to the device. This relationship is input using the **LLRMantissa** and **LLRExponent** registers. The equation for the LLR Normalization value is shown below.

cd - Constellation delta. This is the distance between the origin and the first constellation point, measured along the X axis. It is represented in terms of the programmed quantization. For 8-PSK, it is the distance between the origin and the unit circle.

$$LLRNormalization = \frac{1}{cd}$$

For example, if a system is using 16-QAM, with 2s complement notation and 4 bits each for I and Q, Figure 22 shows the measurement for *cd*.

So, the value for LLR Normalization is 1/1.75, or 0.5714. This value is written into the **LLRMantissa** and **LLRExponent** registers as a floating point number (See Section 9.7.3).

After determining a correct *cd* (constellation delta) the LLR Normalization factor can be calculated as follows in terms of **LLRMantissa** and **LLRExponent** register parameters (See Section 9.7.3).

$$LLRNormalization = \frac{1}{cd} = \left[1 + \frac{LLRMantissa}{256}\right] \times 2^{(LLRExponent - 8)}$$

Following are formulas for estimating the **LLRMntissa** and **LLRExponent**. Note, the solution for **LLRExponent** is required to solve for **LLRMantissa**.

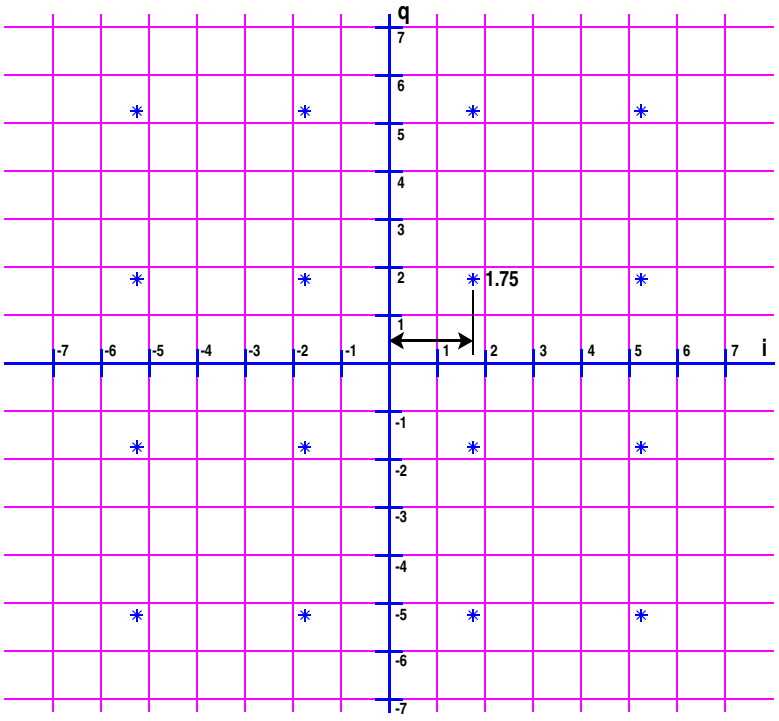
$$LLRExponent = \left\lfloor \frac{\ln(LLRNormalization)}{\ln(2)} + 8 \right\rfloor$$

Rounded down to the nearest whole number.

$$LLRMantissa = [(2^{(8 - LLRExponent)} \times LLRNormalization) - 1] \times 256$$

Rounded to the nearest whole number.

Figure 22: LLR Normalization with 16-QAM



4.2.5 FRAME SYNCHRONIZATION

The AHA4541 supports automatic block synchronization prior to TPC decoding. All synchronization is done at the bit level, after mapping from symbols to soft metrics. The AHA4541 supports synchronization marks of length up to 32 bits. Inverted synchronization marks are placed at the start of a TPC block. Non-inverted marks may be distributed throughout the block to allow for decreased synchronization time.

The synchronizer looks at multiple points in the data stream, separated by the period of the synchronization marks. The decoder uses the frame synchronization mark defined by **DFSyncLength, DFSyncMark, DFSyncFreq,** and **DFSyncPer** to determine where sync marks are expected (See Section 3.7 for frame sync description). The **FBitThresh** value determines how many bits in a sync mark can be incorrect, and still be considered a valid sync mark. If **FSyncInv** is set, the synchronizer also attempts to synchronize to an inverted bit stream. If synchronization is found in the inverted stream, the synchronizer inverts all following data bits.

The sync acquisition detection circuit is an up/down counter that is incremented for each valid sync mark (less than or equal to **FBitThresh** errors), and decremented for each invalid mark. When the count equals **FSyncOn**, the **FSynced** bit is set and data is input to the decoder. If packet synchronization is disabled (**DPSyncEnable** = 0), then the **SyncAcq** interrupt is also set. If packet synchronization is enabled, the **SyncAcq** interrupt is set after both frame and packet sync are acquired.

When synchronized, the device has two built in methods to detect loss of synchronization. An up/down counter monitors the synchronization marks coming over the channel. It is incremented for each invalid mark (greater than **FBitThresh** errors), and decremented for each valid mark. If the count equals **FSyncOff**, a loss of synchronization is assumed, and a resynchronization is executed. In addition, the CRC comparator keeps a count of consecutive failed CRC blocks. If this count equals the **CRCSyncOff** threshold, a loss of synchronization is assumed, and a resynchronization is executed. A different interrupt is generated for each of the above cases, and one or both loss of sync methods can be disabled.

The microprocessor can also issue a resynchronization at any time by writing a one to the **Resync** bit. This does not generate an interrupt. When an automatic or manual resync occurs, all data in the decode path is discarded.

The internal frame synchronizer can be disabled by clearing the **DFSyncEnable** bit. When cleared, synchronization must be achieved with external logic. The **CSTART** signal must be asserted with the first transfer of each frame. Any bits inserted beyond the end of the TPC block, and the beginning of the next frame are discarded by the device. The beginning of each frame must be aligned with the transfer. This requires padding externally if the frame is not a multiple of the bits per symbol. In addition, at the end of a block, if no further data is to be input to the device, 8 additional bits must be input to the device to flush internal pipes and allow the previous block to be decoded and output. These bits will automatically be discarded by the device, since they are not marked with a **CSTART**. **CSTART** should not be issued in the middle of a frame. If a resynchronization is required, the microprocessor must first issue a **Resync**, followed by data marked with **CSTART**. **CSTART** must be tied inactive when **DFSyncEnable** is set.

4.3 TPC DECODER

The Turbo Product Code decoder supports block sizes up to 16,384 encoded bits. The decoder supports iterative decoding of two or three dimensional codes built from extended Hamming or parity only codes of length up to 128 bits. The decoder supports decoding of product codes with hyper axis to improve low bit error rate (BER) performance.

4.3.1 HELICAL DEINTERLEAVER

The helical deinterleaver is enabled by setting **DHelical** to a 1. See Section 3.6 for a description of helical interleaving.

4.3.2 CODE CONFIGURATION

Turbo Product Codes are specified by the constituent codes of each axis in the code. The code configuration registers specify the code type (extended Hamming or Parity) and length of each axis, as well as enabling hyper axis support.

To generate a specific block size, the product code can be shortened using the shortening configuration registers. Using the **xShortX**, **xShortY**, **xShortZ**, **xShortB**, and **xShortR** registers, an exact data size can be configured.

The decoder can be configured to run a variable number of iterations per block. This allows the decoder to spend more time on difficult blocks, making up for this with less difficult blocks. In an AWGN channel, for example, this feature can allow decoder performance of 30 iterations, while only running an average of 10 iterations per block. The gain may vary in other channels. This feature is enabled by setting **StopIter**.The AHA4541 contains internal buffering to support this feature, as described in Section 5.6.

4.3.3 FEEDBACK

The TPC algorithm uses feedback, or weighting values for performance tuning. After each axis iteration, the output of the Soft Input Soft Output (SISO) decoder is multiplied by the feedback constant for that axis. These values are then fed back into the SISO for future iterations.

The feedback multiplier values used for each code axis vary from 1/32 to 31/32. The feedback multipliers must be tuned to give optimum decoder performance in a given system. The choice of feedback multiplier has no effect on throughput or latency. The following paragraphs describe the tuning process.

For 2D square (XCODE[3:0] = YCODE[3:0]) codes, a typical feedback multiplier value for both axes at 4 iterations is 16/32. For 3D cubic (XCODE[3:0] = YCODE[3:0] = ZCODE[3:0]) codes, a typical feedback multiplier value at 6 iterations is 14/32.

When using non-square or cubic codes, the following general rules should be applied. Parity codes should have their feedback multiplier values set higher than Hamming codes when mixed. For example, in a (32,26)x(32,26)x(4,3) code, the X and Y feedback multipliers should be set to 12/32 while the Z feedback should be set to 18/32 or 20/32. When mixing Hamming codes with shorter Hamming codes, the feedback multiplier should be set slightly higher for the shorter code. For example, in a (64,57)x(32,26) code, the X feedback multiplier could be set to 16/32, while the Y feedback multiplier could be set to 18/32.

When using eTPCs there are two added feedback terms. These are Enhanced Feedback and Enhanced Parity Feedback. The Enhanced Feedback is the feedback multiplier for the Enhanced TPC mode. The recommended starting value for this parameter is 24/32. The Enhanced Parity Feedback is the feedback multiplier for the Enhanced Mode parity axis. The recommended starting value for the Enhanced Parity Feedback is 24/32.

The feedback values must be tuned for the number of iterations allowed in a system. For less iterations than the above guidelines, the feedback values should be increased slighly. For more iterations, the values should be decreased. For example, when using a (64,57)x(64,57) code with only 2 iterations, the feedback multiplier for both axes should be set to 20/32. Conversely, in a system that allows 12 or more iterations, the value for the feedback should be set to 14/32.

The feedback parameters may also need to be tuned depending on the number of soft input bits (QBits2:0]). This will only affect the optimum feedback multiplier values slightly, meaning that they should be adjusted by only 1/32 to 4/32 to allow for these differences.

Since systems vary widely, the system designer should experiment with various feedback multiplier values to obtain the best performance. AHA also has available the Galaxy simulation software which is a very useful tool for modeling the effects of the Feedback parameters for all codes supported by the AHA4541 device. Please contact AHA applications engineering for more information about the Galaxy software.

4.3.4 CODE SHORTENING

There are two methods of shortening product codes. The first method is to remove an entire row or column from a 2-dimensional code, or an entire X, Y, or Z plane from a 3-dimensional code. This is equivalent to shortening the constituent codes that make up the product code, and is accomplished by writing the amount to shorten into the **xShortX**, **xShortY**, and **xShortZ** registers. This method enables a coarse granularity on shortening, and at the same time maintaining the highest code rate possible by removing both data and parity symbols. Further shortening is obtained by removing individual bits from the first row of a 2-dimensional code (using **xShortB**), or from the top plane of a 3-dimensional code (using **xShortR**).

Shortening will be described using a 2-dimensional code example, and a 3-dimensional code example. Assume a 456 bit block size is required, with code rate of approximately 0.6. The base code chosen before shortening is the (32,26)x(32,26) code which has a data size of 676 bits. Shortening all rows by 5 and all columns by 4 results in a (27,21)x(28,22) code, with a data size of 462 bits. To get the exact block size, the first row of the product is shortened by an additional 6 bits. The final code is a (750,456) code, with a code rate of 0.608. Figure 23 shows the structure of the resultant block. This shortening is programmed into the device by writing **xShortX** to 5, **xShortY** to 4, and **xShortB** to 6.

For the 3-dimensional example, suppose a 0.4-0.45 rate code is required with a data block size of 1100 bits. Starting with a (32,26)x(32,26)x(4,3) base code, the optimum shortening for this code is to remove rows and columns, while leaving the already very short z axis alone. Therefore, since we desire a 1100 bit 3D code, we can find the desired vector data size by taking the square root of 1100/3, and rounding up. This yields a row/column size of about 20. In fact, having a row size of 20, a column size of 19, and a z column size of 3 gives us the closest block size to 1100 bits. This is accomplished by writing **xShortX** to 6, **xShortY** to 7, and **xShortZ** to 0.

The code size is now a (26,19)x(25,20)x(4,3) = (2600,1140). To get the exact data size, we further shorten the first plane of the code by 40 bits. This is accomplished by shortening 2 full rows from the first plane (**xShortR**), with each row removing 19 bits from the data block, and shortening another 2 bits from the next row (**xShortB**). This results in a (2546,1100) code, with rate = 0.43. Figure 24 shows the original code, along with the physical location of the shortened bits. This is accomplished by writing **xShortB** to 2 and **xShortR** to 2.

Figure 23: Structure of Shortened Code

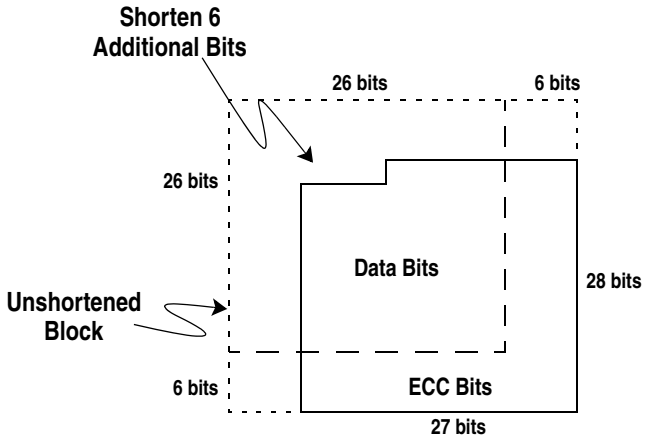
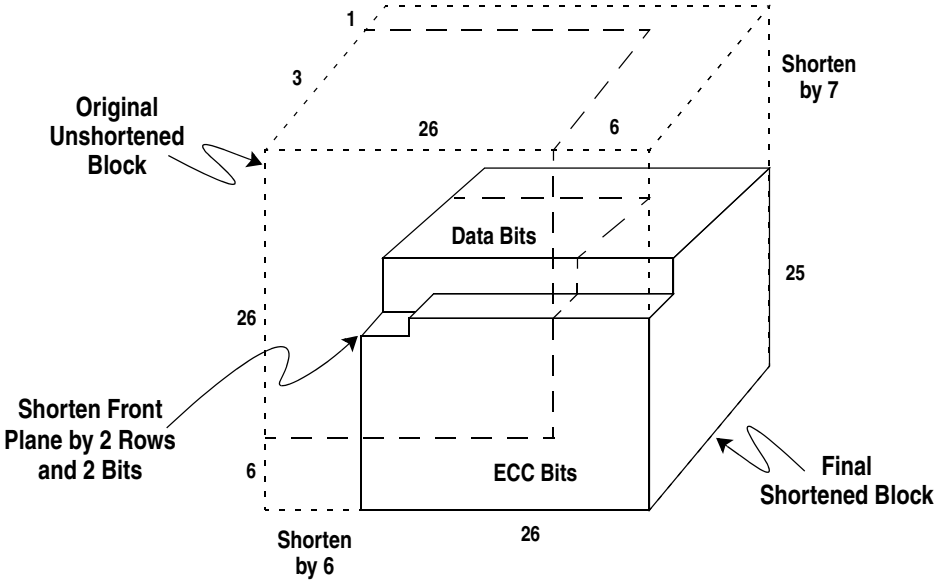


Figure 24: Structure of Shortened 3D Block



4.4 MODULATION SYMBOL DEINTERLEAVING

This option improves performance for systems using higher order modulation. For a detailed description of the function, see Section 3.5.

4.4.1 CODE PERFORMANCE

Table 2 gives an abridged list of possible codes supported by the AHA4541, along with the code rate and performance of each code. The performance is measured assuming a user (uncoded) data rate of 311 Mbits/sec. This is a very small subset of supported codes. AHA can provide software to assist the code selection process.

Table 2: Partial Code List and Performance at 155 Mbit/sec Data Rate

CODE (X)x(Y)x(Z)	BLOCK SIZE (bits)	DATA SIZE (bits)	RATE	CODING GAIN* (dB)	# of ITERATIONS at 311 Mbps
(128,127)x(128,126)+	16384	16002	0.977	4.3	4
(128,120)x(128,126)+	16384	15120	0.923	5.5	4
(128,127)x(64,62)+	8192	7874	0.961	4.4	3
(128,120)x(64,62)+	8192	7440	0.908	5.5	3
(64,63)x(64,62)+	4096	3906	0.954	4.5	3
(64,63)x(32,30)+	2048	1890	0.923	4.5	2
(128,120)x(128,120)	16384	14400	0.879	6.5	6

* *Estimated Coding Gain is measured on a Binary Input Additive White Gaussian Noise (AWGN) channel at 10^{-6} Bit Error Rate (BER) and Uncoded Data Rate of 311 Mbps.*

\+ *enhanced TPC (includes hyper axis). For 2D codes, the Y axis is shortened by 1. The shortening and addition of hyper axis is included in the code description.*

Table 3: Partial Code List and Achievable Data Rate (Payload) at Various Iterations

CODE (X)x(Y)x(Z)	RATE	4 ITERATIONS (Mbit/sec)	6 ITERATIONS (Mbit/sec)	12 ITERATIONS (Mbit/sec)
(128,127)x(128,126)+	0.977	351	238	119
(128,120)x(128,126)+	0.923	330	226	113
(128,120)x(128,120)	0.879	316	316	161
(64,57)x(16,15)x(16,15)	0.783	280	223	110
(32,26)x(32,26)x(16,15)	0.619	221	177	87
(128,127)x(64,62)+	0.961	320	213	106
(128,120)x(64,62)+	0.908	301	200	100
(128,120)x(64,57)	0.835	299	267	133
(32,26)x(16,15)x(16,15)	0.714	257	190	96
(32,26)x(32,26)x(8,7)	0.577	207	152	75
(64,63)x(64,62)+	0.954	267	177	87
(64,57)x(64,57)	0.793	284	217	108
(64,63)x(32,30)+	0.923	211	140	69
(64,57)x(32,26)	0.724	234	154	77
(32,26)x(32,26)	0.660	161	106	52

\+ *enhanced TPC (includes hyper axis). For 2D codes, the Y axis is shortened by 1. The shortening and addition of hyper axis is included in the code description.*

4.4.2 DECODER DATA RATES

Table 3 shows the data (payload) rates for various codes at 4, 6, and 12 iterations. This table assumes a decoder processing clock (DPCLK) of 180 MHz.

4.4.3 DATA RATE CALCULATION

DCodeX is **DCodeX[2:0]**, **DCodeY** is **DCodeY[2:0]**, and **DCodeZ** is **DCodeZ[2:0]**. Determine clocks per iteration:

if z code is > 0

$$x\_axis = \frac{2^{\mathbf{DCodeX}} \times (2^{\mathbf{DCodeY}} - \mathbf{DShortY}) \times (2^{\mathbf{DCodeZ}} - \mathbf{DShortZ})}{64} + 10 + \frac{2^{\mathbf{DCodeX}}}{2}$$

$$y\_axis = \frac{2^{\mathbf{DCodeY}} \times (2^{\mathbf{DCodeX}} - \mathbf{DShortX}) \times (2^{\mathbf{DCodeZ}} - \mathbf{DShortZ})}{64} + 10 + \frac{2^{\mathbf{DCodeY}}}{2}$$

$$z\_axis = \frac{2^{\mathbf{DCodeZ}} \times (2^{\mathbf{DCodeX}} - \mathbf{DShortX}) \times (2^{\mathbf{DCodeY}} - \mathbf{DShortY})}{64} + 10 + \frac{2^{\mathbf{DCodeZ}}}{2}$$

enhanced_axis = z_axis (if enabled)

$$total\_clks = x\_axis + y\_axis + z\_axis + enhanced\_axis + 10$$

if z code = 0

$$x\_axis = \frac{2^{\mathbf{DCodeX}} \times (2^{\mathbf{DCodeY}} - \mathbf{DShortY})}{64} + 10 + \frac{2^{\mathbf{DCodeX}}}{2}$$

$$y\_axis = \frac{2^{\mathbf{DCodeY}} \times (2^{\mathbf{DCodeX}} - \mathbf{DShortX})}{64} + 10 + \frac{2^{\mathbf{DCodeY}}}{2}$$

enhanced_axis = y_axis (if enabled)

$$total\_clks = x\_axis + y\_axis + enhanced\_axis + 10$$

$$\text{channel rate} = \frac{\text{dpclk freq} \times (\text{total \# bits in the block})}{\text{iterations per block} \times total\_clks \times 2}$$

$$\text{data rate} = \frac{\text{dpclk freq} \times (\text{data bits in the block})}{\text{iterations per block} \times total\_clks \times 2}$$

4.4.4 DECODER LATENCY CALCULATIONS

The decoder latency is defined as the time from when the last channel bit is input, until the **D_RDY** signal is asserted to indicate that a block is ready to be output. In the following equations, **DCodeX** is **DCodeX[2:0]**, **DCodeY** is **DCodeY[2:0]**, and **DCodeZ** is **DCodeZ[2:0]**. Determine clocks per iteration first.

If **DCodeZ** > 0,

$$x\_axis = \frac{2^{\mathbf{DCodeX}} \times (2^{\mathbf{DCodeY}} - \mathbf{DShortY}) \times (2^{\mathbf{DCodeZ}} - \mathbf{DShortZ})}{64} + 19 + 2^{\mathbf{DCodeX}}$$

$$y\_axis = \frac{2^{\mathbf{DCodeY}} \times (2^{\mathbf{DCodeX}} - \mathbf{DShortX}) \times (2^{\mathbf{DCodeZ}} - \mathbf{DShortZ})}{64} + 19 + 2^{\mathbf{DCodeY}}$$

$$z\_axis = \frac{2^{\mathbf{DCodeZ}} \times (2^{\mathbf{DCodeX}} - \mathbf{DShortX}) \times (2^{\mathbf{DCodeY}} - \mathbf{DShortY})}{64} + 29 + 2^{\mathbf{DCodeZ}}$$

Enhanced_axis = z_axis - 5 (if enabled)

$$total\_clks = (x\_axis + y\_axis + z\_axis + enhanced\_axis) \times \mathbf{Iterations}$$

If z code is 0

$$x\_axis = \frac{2^{\mathbf{DCodeX}} \times (2^{\mathbf{DCodeY}} - \mathbf{DShortY})}{64} + 19 + 2^{\mathbf{DCodeX}}$$

$$y\_axis = \frac{2^{\mathbf{DCodeY}} \times (2^{\mathbf{DCodeX}} - \mathbf{DShortX})}{64} + 19 + 2^{\mathbf{DCodeY}}$$

Enhanced_axis = y_axis (if enabled)

$$total\_clks = (x\_axis + y\_axis + enhanced\_axis) \times \mathbf{Iterations}$$

$$decoder\_latency = total\_clks + \text{the greater of} \begin{cases} total\_block\ size/4 \\ \text{or} \\ total\_clks \end{cases}$$

$$\text{Total latency (seconds)} = [8 \times CCLK\_period] + [(decoder\_latency + 23) \times (DPCLK\_period)] + [5 \times DCLK\_period]$$

Note: add [16 x CCLK_period] if demodulation is enabled.

4.4.5 CORRECTIONS COUNT

In order to assist the system with estimation of channel bit error rates, the AHA4541 device reports the number of bits corrected in each block decoded. The device can also be configured to compute a running average of the number of corrections per block over a programmable number of blocks.

When **CorrAvg** is set to 0, the **Corrections** register contains the number of corrections between incoming (channel) data and outgoing (decoded) data. This corrections value is the corrections done on all bits in the TPC block, including user data, packet sync marks, CRC, and TPC ECC bits. It is updated with every **DecComp** interrupt. When **CorrAvg** is set to a non-zero value, the corrections per block is averaged using a running average over the previous 2**CorrAvg** blocks. The **Corrections** register is updated with every **DecComp** interrupt, giving the running average value.

4.5 DECODED DATA INTERFACE

The decoded data interface checks the CRC for data integrity, and outputs this data to the **DDATA[7:0]** output port. Figure 25 shows the block diagram of the decoded data interface.

Figure 25: Decoded Data Interface Block Diagram

When using Stream Mode, the decoded data interface searches through the decoded stream to find the beginning of user data packets. When using Block Mode, the beginning of the packet is assumed to be the beginning of the TPC block. The CRC engine computes CRC on each user packet, and compares the computed CRC to that appended to the data. The packet stream is output from the device, along with packet start and packet error signals.

4.5.1 PACKET SYNCHRONIZATION

When using Stream Mode, the user data packets out of the decoder may not line up with the beginning of a TPC block. Therefore, it is necessary to locate the beginning of each packet so that a CRC can be computed over the packet. This is accomplished by searching for the packet sync marks in the stream out of the decoder. Since this data stream is post-decoder data, its bit error rate is extremely low, and there is no phase ambiguity. Therefore, synchronization marks must be perfect to be considered valid (there is no **FBitThresh** equivalent as in the frame synchronization function).

Packet synchronization is automatically enabled when **xBlockMode** is set to 0. If the packet synchronization marks are to be removed from the data stream, the **DPSyncRemove** bit should also be set. If this bit is not set, the synchronization marks remain in the data stream and are output with the user packet data. This can be used when the data stream already contains synchronization marks (such as MPEG).

The size of the packet synchronization mark can be 4, 8, 12 or 16 bits, selected with the **DPSyncLength** value. The value for the synchronization marks should be written into the **DPSyncMark** register. The **PSyncOn** register determines how many valid packet sync marks must be seen before declaring packet synchronization is acquired. The **SyncAcq** interrupt is set when both frame and packet synchronization occur. In addition, the **PSynced** bit is set when packet synchronization is achieved.

4.5.2 DESCRAMBLER

The descrambler is built with a 16 bit pseudo-random binary sequence generator with programmable polynomial, length, and initialization seed. The descrambler logic matches that of the scrambler, shown in Figure 6.

Figure 6 shows an example configuration for the descrambler that matches the DVB standard scrambler. The descrambler programming matches that of the scrambler, described in Section 3.3.

The packet synchronizer expects to find inverted packet sync marks at every **DScramPer** packets, where a value of 0 represents 16. The descrambler is reset at each inverted sync mark. If the packet synchronization marks are still in the data stream, they are inverted to return them to the original state. Note that when **DScramPer** is set to 1 the descrambler is reset at every packet sync mark, and no inversion of the sync marks is done. When using Block Mode or when the descrambler is disabled, **DScramPer** should by set to 1. Note that the packet synchronization marks are not descrambled by the descrambler. However, the PRBS shift register continues to shift over the synchronization marks.

4.5.3 CRC CHECKING

CRC checking is executed after decoding. Each packet has a separate CRC, with variable length up to 32 bits. The CRC comparator circuit matches that of the encoder, shown in Figure 5.

The shift register is reset at the beginning of each packet. Data from the packet (excluding the packet synchronization mark) is shifted into the circuit, until the number of bytes programmed into **DPSize** plus the number of bits in the CRC field is reached. At this point, the contents of the CRC shift register are examined. If all bits are zero, then the CRC is correct. Otherwise, a CRC error is detected, and the **PERR** output signal is asserted at the end of the output of the packet, at the same time as the **PEND** signal is asserted. In addition, the CRC failure count is incremented, and will cause a resync and interrupt if the **CRCSyncOff** threshold is met.

5.0 DATA FLOW CONTROL

The encoding and decoding processes of the AHA4541 device require data input and output in a bursty fashion. However, the device contains internal buffering to allow continuous data input and output for both encoding and decoding.

The relationship between the number of transfers input to the decoder relative to the number of transfers output from the decoder is dependent on the user packet size, TPC code rate, sync mark size, user packet size, CRC, pad bits, and symbol size. Since there are so many constraints that affect the transfer rate, system clocking issues can be difficult to solve.

The AHA4541 device contains internal logic to greatly simplify these system design issues, and reduce external logic requirements. In addition, transfer per clock input and output is supported with the use of an external Voltage Controlled Oscillator (VCO). The logic implemented is the same for the encoder and decoder. Therefore, the description given can be applied to both.

The maximum data throughput of the encoder is **EPCLK** frequency times 4 (bits). Setting up flow control to a data rate higher than this will cause the input accept (**UACPT**) to deassert. The decoder maximum data rate is limited by **DPCLK** times 2 (bits). In addition, the maximum rate through the encoder may be limited by the code and number of iterations.

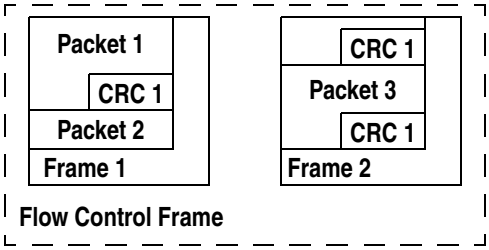
5.1 DATA INPUT TRANSFERS vs. DATA OUTPUT TRANSFERS

In order for the device to determine the rate at which data must be input and/or output, the ratio of the overall input vs. output transfer rates is programmed into the device. This ratio must take into account all data inserted and/or removed in the data stream, and the symbol size. When using Stream Mode, the numerator and denominator of this ratio can be quite large, since the user data packet need not be a multiple of the TPC block size.

A flow control frame is a group of one or more frames that have a fixed number of unencoded data interface transfers and encoded data interface transfers. The concept of a flow control frame will be described by example. Assume the device is set up for Stream Mode, with the user packet size equal to 3 bytes, with a 1 byte CRC, and a (64,48) bit code. Figure 26 shows two consecutive frames of this structure. Since one user data packet plus CRC is equal to 2/3 of the data size of a frame, two frames will contain three user packets.

The importance of a flow control frame can be appreciated as follows. If the input output transfer ratio were defined using only one frame, the ratio would not be exact. For example, frame 1 in Figure 26 contains 5 user bytes (3 bytes in packet 1, 2 bytes in packet 2). Using this frame would result in a ratio of 5 user bytes in for every 8 bytes out. However, frame 2 contains only 4 user bytes. Using this frame would result in a different ratio of 4 user bytes in for every 8 bytes out. The correct ratio can be found using the concept of a flow control frame, and is equal to 9 user bytes in for every 16 user bytes out.

Figure 26: Flow Control Frame Example



For the encoder, the number of unencoded input transfers per flow control frame is written into the **UTransfers** register. The number of encoded output transfers per flow control frame is written into the **ETransfers** register. Similarly, for the decoder, the number of decoded data transfers per flow control frame is written into the **DTransfers** register. The number of channel interface transfers per flow control frame is written into the **CTransfers** register. The following equations can be used to find the value for each of these registers.

5.1.1 xTRANSFERS WITH xBLOCKMODE = 1

pack – User Data Packet Size (bytes).

$$pack = xPSize$$

fsize – Frame Size including sync marks and pad (bits).

$$fsize = xFSyncPer \times xFSyncFreq$$

m – Bits per symbol.

$$m = xSymSize + 1$$

The overall code rate (*cr*) of the system is:

$$cr = \frac{pack \times 8}{fsize}$$

The number of unencoded transfers (*ut*) or decoded transfers (*dt*) per flow control frame is:

$$ut, dt = pack \times m$$

The number of encoded transfers (*et*) per flow control frame is:

$$et = fsize$$

When ChannelFormat = 0, 1, or 2, the number of channel transfers (*ct*) per flow control frame is:

$$ct = fsize$$

When ChannelFormat = 3, the number of channel transfers (*ct*) per flow control frame is:

$$ct = fsize \times \left\lceil \frac{m}{4} \right\rceil$$

5.1.2 xTRANSFERS WITH xBLOCKMODE = 0

pack – User Data Packet Size (bytes).

EPSInsert / DPSRemove = 1:

$$pack = xPSize$$

EPSInsert / DPSRemove = 0:

$$pack = xPSize + \frac{xPSyncLength + 1}{2}$$

crc – CRC Size (bits). If CRC is disabled, set to 0.

$$crc = xCRCSize^{*} \times 4$$

** use 8 if xCRCSize = 0.*

psync – Packet Sync Size (bits).

EPSInsert / DPSRemove = 1:

$$psync = 4 \times (xPSyncLength + 1)$$

EPSInsert / DPSRemove = 0:

$$psync = 0$$

tpck – Turbo Product Code Data Size (bits).

fsize – Frame Size including sync marks and pad (bits).

$$fsize = xFSyncPer \times xFSyncFreq$$

m – Bits per symbol.

$$m = xSymSize + 1$$

The overall code rate (*cr*) of the system is:

$$cr = \frac{tpck \times pack \times 8}{fsize \times (pack \times 8 + crc + psync)}$$

The number of unencoded transfers (*ut*) or decoded transfers (*dt*) per flow control frame is:

$$ut, dt = tpck \times pack \times m$$

The number of encoded transfers (*et*) per flow control frame is:

$$et = fsize \times (8 \times pack + crc + psync)$$

When ChannelFormat = 0, 1, or 2, the number of channel transfers (*ct*) per flow control frame is:

$$ct = fsize \times (8 \times pack + crc + psync)$$

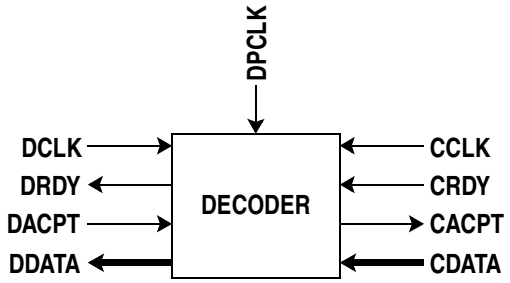
When ChannelFormat = 3, the number of channel transfers (*ct*) per flow control frame is:

$$ct = fsize \times (8 \times pack + crc + psync) \times \left\lceil \frac{m}{4} \right\rceil$$

5.2 FLOW CONTROL DISABLED

Flow control is disabled by setting **xFlowMode** to 0. When disabled, data is handshaked in and out of the device using full handshakes with the appropriate **RDY** and **ACPT** signals. This mode will generate the minimum possible latency through the device at the expense of added system complexity. Note that in this mode the internal clock domain crossing is still enabled, so that the various channel and data clocks may be driven at differing frequencies. In this mode, the **xTransfers**, **DBufferSize** and **DumpThresh** registers are ignored. Figure 27 shows a wiring diagram of the encoder when flow control is disabled.

Figure 27: Flow Control Disabled



Data is input to the device on the rising edge of **UCLK** when both **URDY** and **UACPT** are asserted. Similarly, data is output from the device on the rising edge of **ECLK** when both **ERDY** and **EACPT** are asserted. This same handshaking occurs when the decoder flow control is enabled.

All internal buffering in the encode path is disabled. The device will handshake data in as needed to feed the encoder. When the encoder is outputting ECC or other overhead bits, the input handshake will pause.

The decoder buffering is reduced to the minimum necessary to simultaneously load, decode, and unload three consecutive blocks. While the decoder is working on a block, a one block input buffer can be loading a following block. In addition a one block output buffer can be unloading a previously decoded block.

5.3 INTERNAL BUFFERING ENABLED

Setting **xFlowMode** to 1 enables encoder or decoder data buffering. In this mode, data is handshaked in and out of the device using full handshakes with the appropriate **RDY** and **ACPT** signals. In this mode, the **xTransfers** registers are ignored. The wiring is the same as mode 0, shown in Figure 27.

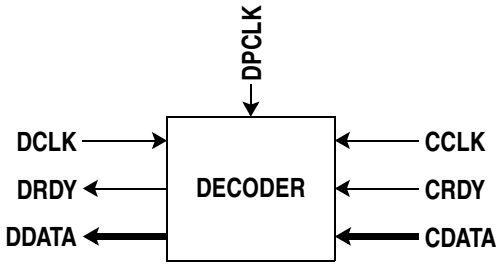
The encoder FIFO is enabled to smooth the burst nature of the encoder input. The logical size of the encoder FIFO is set via **EBufferSize**.

The decoder data buffer is enabled as two FIFOs, each of logical size determined by **DBufferSize**. One FIFO is at the input to the decoder, and one is at the output of the decoder. Note that these FIFOs are in addition to the one block buffering that is available in the decoder (see Section 5.2). The size of the FIFOs ranges from 128 to 32,768 bits (a bit in the decoder input FIFO is one soft metric). If the variable iterations feature (**StopIter**) is enabled, the **DumpThresh** can be set to instruct the decoder to stop iterating on a block when the input FIFO fills. See Section 5.6 for a description of this feature.

5.4 FLOW CONTROL VIA HANDSHAKING

Flow control via handshaking is enabled by setting **xFlowMode** to 2 or 3. Mode 2 is used when the channel side (encoded interface for the encoder) is the master. Mode 3 is only valid with the encoder, and allows the unencoded interface to be the master. Figure 28 shows a wiring diagram for the decoder when the channel interface is the master.

Figure 28: Handshake with Channel Interface Master



Data is input to the device on the rising edge of **CCLK** when the **CRDY** input is asserted. In this mode, the **CRDY** signal is a data valid signal. If the demodulator can input data at one transfer per clock, the **CRDY** signal should be tied active, causing the AHA4541 device to assume every clock has valid input data. In this mode, the **CACPT** output will only assert if an internal buffer overflow occurred. This is caused by an incorrect configuration, and can be used in debugging.

Data is output from the device on the rising edge of **DCLK** when the **DRDY** output signal is asserted. The **DACPT** input should be tied active in this mode. The **DRDY** output signal will assert at regular intervals to transfer data out of the device. The number of transfers output versus input is configured with the **CTransfers** and **DTransfers** registers.

When the encoder is configured for flow control mode 3, the unencoded interface **URDY** input signal controls data transfer into the device. The ratio of the **UTransfers** to **ETransfers** sets the rate at which the encoded data interface **ERDY** output signal asserts to transfer data out of the device. When the encoder is configured for mode 2, the encoded data interface **EACPT** input signal controls data transfer out of the device, and the **UACPT** output signal controls data transfer in to the device. Note that in this case, the **UACPT** output remains active until the first transfer out of the encoded port. After this transfer, the **UACPT** signal will assert at regular intervals to transfer the data out of the device.

When using flow control via handshaking, an inactive input handshake signal on the master port causes an immediate deassertion of the output handshake on the slave port. Therefore, if the rate of data input to the device changes, the rate output will immediately change to reflect this difference. This requires the data input to be a constant stream. If this is not possible, the device must be flushed by driving the input handshake active until all data is output from the device. It must then be reset with a soft reset (**EReset** or **DReset**) to remove the flushed data from the internal data paths.

When using the decoder with flow control via handshaking, the decoder buffer can be configured for a variable number of iterations per block. Data buffering around the decoder guarantees a constant delay through the device, even though the decoder is running a different number of iterations on each block. See Section 5.6 for a description of this feature.

5.5 FLOW CONTROL VIA CLOCK FREQUENCY SYNTHESIS

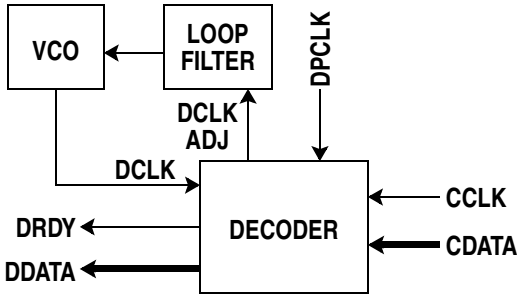
Many communications systems operate at one transfer per clock across each interface. The transfer may contain one bit, one channel symbol, or one data byte, and no handshaking occurs across the interfaces. To use this data flow scheme, the rate of the clock at the input to the encoder must be lower than the rate at the output of the encoder by the difference in data added to the data stream. Similarly, the rate at the input to the decoder must be higher than the rate at the output of the decoder by the difference in data removed from the data stream. This is often handled using error correction codes that have a simple code rate, such as 2/3 or 5/6. Using this method constrains the code options and system configurations when using Turbo Product Codes.

The AHA4541 contains internal clock frequency synthesis logic to generate a clock given the other clock. For instance, the decoded data clock, **DCLK**, can be generated with a phase-locked loop using the code rate and the channel clock, **CCLK**, as a reference. Similarly, the unencoded data clock, **UCLK**, can be generated using the encoded data clock, **ECLK**, as a reference. As with the handshaking flow control, the code rate is configured using the **xTransfers** registers, as described in Section 5.1.

To enable frequency synthesis, **xFlowMode** should be set to 4 or 5. Mode 4 is used when the data is an intermittent stream. Mode 5 is used when the data is a constant stream and the system requires a constant latency through the encoder or decoder.

Figure 29 shows a block diagram of the decoder configured to generate the decoded data clock, **DCLK**.

Figure 29: Decoder Clock Frequency Synthesis



Using the code rate specified in the **xTransfers** registers, a phase-locked loop is created using **CCLK** as a reference. The **CCLK** frequency is multiplied by the code rate, and the resultant clock is compared to **DCLK**. The phase and frequency difference are output over the **DCLKADJ** bus. This output can be configured as a charge pump to drive an external loop filter and Voltage Controlled Oscillator (VCO). See Section 6.1 for more information. This phase-locked loop will generate **DTransfers** clock edges on **DCLK** per flow control frame, while the reference clock, **CCLK**, has **CTransfers**.

Data is input to the decoder on the rising edge of **CCLK**, with one transfer per clock. When the decoded data reaches the output of the decoder, the **DRDY** signal will assert. This data is transferred out of the device on the rising edge of **DCLK**, with one transfer per clock. The **DRDY** signal will only deassert when an internal buffer underflow occurs. This condition is caused by improperly configured flow control, and should not occur in normal operation.

When using the decoder with **DFlowMode** set to 4, the decoder data buffer is enabled as two FIFOs, each of logical size determined by **DBufferSize**. One FIFO is at the input to the decoder, and one is at the output of the decoder. Note that these FIFOs are in addition to the one block buffering that is available in the decoder (see Section 5.2). The size of the FIFOs ranges from 128 to 32,768 bits (a bit in the decoder input FIFO is one soft metric).

The same logic exists for encoder clocking. In addition, the above circuit can be reversed, using **DCLK** as the reference clock, and generating **CCLK** with the PLL. The decoder processing clock, **DPCLK**, must be set in order to achieve the desired number of iterations for the given code, as described in Section 4.4.2

When using the decoder with a constant data stream, **DFlowMode** is set to 5. This mode allows for variable iterations per block. The data buffering around the decoder guarantees a constant latency through the device, even though the decoder is running a different number of iterations on each block. See Section 5.6 for a description of this feature.

5.6 DECODER DATA BUFFER WITH VARIABLE ITERATIONS

When **DFlowMode** is set to 2 or 5, the TPC decoder can be configured to run a variable number of iterations by setting **StopIter**. The device contains internal buffering to allow a variable number of iterations per block, with a constant data flow through the device. This feature is automatically enabled when flow control is enabled.

When the decoder requires more iterations on certain blocks, the buffer stores incoming data bits until the decoder completes the block. A second logical buffer is placed on the output of the decoder to give a fixed latency to the decoder.

The logical size of this buffer is set via the **DBufferSize** register. Setting this to a larger value allows the decoder to iterate more times on difficult blocks. Setting this to a smaller value decreases the latency through the device. See Section 9.8.2 *Buffer Configuration (EBUFCON, DBUFCON0-1)* equations used to determine the maximum value for **DBufferSize**.

When **DFlowMode** is not set to zero, the **DumpThresh** can be used to stop iterating when the input buffer fills. When the input buffer becomes nearly full, the device will automatically stop iterating on the current block, send this block to the output buffer, and begin loading the next block. When using this feature, the maximum iterations can be programmed to a value higher than normal, and the device will iterate the maximum amount available. The threshold at which the decoder stops iterating is determined by the value of **DumpThresh**. When the input buffer is within **DumpThresh***128 bits of being full, a dump is issued to the decoder.

The use of the **DumpThresh** feature for variable iterations per block imposes a maximum channel rate on the decoder set by the following equation. If this maximum rate equation cannot be met, **DumpThresh** must be set to 0.

$x = 2^{\text{DCODEX[2:0]}} - \text{DSHORTX}$

$m = 2^{\max(\text{DCODEX[2:0]}, \text{DCODEY[2:0]}, \text{DCODEZ[2:0]})}$

n = TPC encoded block size

f = Frequency of DPCLK ÷ 2

The maximum channel rate (bits/sec) allowed with **DumpThresh** ≠ 0 is:

$$C_{max} < \frac{f}{\frac{\left\lceil \frac{x}{4} \right\rceil}{x} + \frac{1}{32} + \frac{m+64}{n}}$$

DumpThresh should be configured according to the following equation.

$$\textbf{DumpThresh} = \left\lceil \frac{n}{1024} + \frac{m}{32} \right\rceil + 2$$

6.0 CLOCKING SCHEME

The AHA4541 device has 7 clocks. User data bytes are input into the encoder synchronous to the unencoded data clock, **UCLK**. The encoder processing logic operates synchronous to **EPCLK**. Encoded symbols are output from the encoder synchronous to the encoded data clock, **ECLK**. The device contains internal logic to drive an external VCO, generating either **ECLK** or **UCLK** with a phase-locked loop.

Demodulated channel symbols are input to the decoder synchronous to the channel clock, **CCLK**. The decoder processing logic operates synchronous to **DPCLK**. Decoded data bytes are output from the decoder synchronous to the decoded data clock, **DCLK**. The device contains internal logic to drive an external VCO, generating either **CCLK** or **DCLK** with a phase-locked loop.

The microprocessor interfaces operates with a 1/2 divided version of the decoder processing clock, **DPCLK**.

6.1 FREQUENCY SYNTHESIZER CONFIGURATION

Two internal frequency synthesizers are available to generate an encoder and/or decoder clock. The encoder synthesizer compares the phase and frequency of the unencoded clock, **UCLK**, to the encoded clock, **ECLK**. The error is output over the **UCLKADJ** bus. The decoder synthesizer compares the phase and frequency of the decoded data clock, **DCLK**, to the channel clock, **CCLK**.

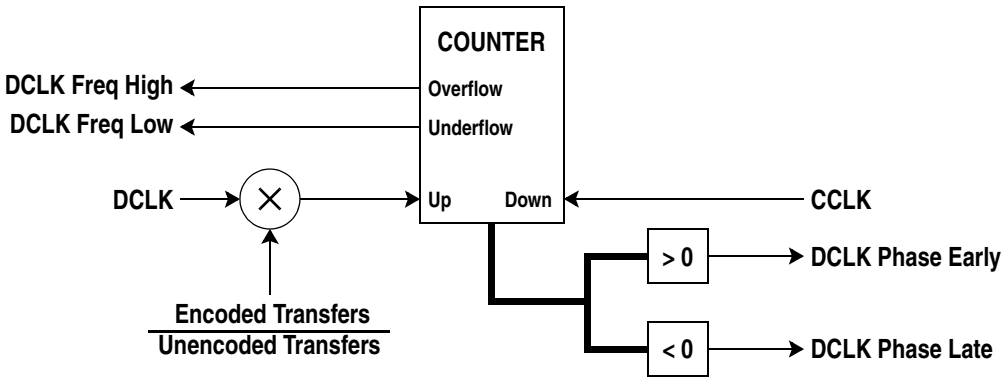
Figure 30 shows a diagram of the decoder frequency synthesizer. The **DCLK** clock is frequency multiplied by the flow control code rate programmed into **xTransfers**. An up/down counter is decremented for each **CCLK** clock pulse, and incremented for each **DCLK** (multiplied by the code rate) pulse. When the clocks are frequency and phase locked, the count will remain at 0. Otherwise, the count will increment or decrement according to the frequency or phase difference.

When **xClkAdjMode** is set, the clock phase error is output on **DCLKADJ[0]**. When the **DCLK** phase is early (the count value is greater than 0), a low value is driven on **XCLKADJ[0]**. When the **DCLK** phase is late (the count value is greater than 0), a high value is driven on **XCLKADJ[0]**. When the phase is matched, **XCLKADJ[0]** is tristated. The **XCLKADJ[1]** signal is a charge pump generated from the clock frequency difference. When the **DCLK** frequency is too high (code rate multiplied **DCLK** pulse without a **CCLK** pulse), **XCLKADJ[1]** is driven low. When the frequency is too low, **XCLKADJ[1]** is driven high. If the frequencies match, the **XCLKADJ[1]** is tristated. **XCLKADJ[1]** can be used to decrease PLL lock time, but cannot be used without also looking at the clock phase. Doing so can result in overfill or underfill of internal buffers due to clock cycle slips. The polarity of the **XCLKADJ[1:0]** signals can be inverted by clearing the **xClkAdjPolarity** bit. **XCLKADJ[2]** is a phase lock indicator. The lock indicator is asserted when the frequency of the two clocks match the required frequencies, and the counter value is at zero.

When **xClkAdjMode** is cleared, the value of the uppermost bits of the up/down phase counter is output on **XCLKADJ[2:0]**. This is a signed, 2's complement number giving the phase offset of the code rate multiplied **DCLK** and **CCLK**.

Note that in both modes, the **UCLKADJ[2:0]** bus is updated synchronous to **ECLK**, and **DCLKADJ[2:0]** is update synchronous to **CCLK**.

Figure 30: Frequency Synthesizer Block Diagram



7.0 GENERAL OUTPUT SIGNALS

The AHA4541 device contains two general output signals that can be used to control external logic, or for system debugging. The signals can be driven to a 1 or 0, or can be driven according to events within the device, including synchronization status, decode complete, etc. See Section 9.9.3 for a complete list of available status outputs.

8.0 MICROPROCESSOR INTERFACE

The device is capable of interfacing directly to a microprocessor for embedded applications. All register accesses to the device are performed on an 8-bit bidirectional bus, using either an Intel® or Motorola® style interface. The interface is in Motorola® mode when the **PROCMODE** input signal is asserted, otherwise the interface is in Intel® mode.

A **MUXMODE** input is also provided to allow the data and address to be multiplexed on the **MDATA[7:0]** bus when using Intel mode (**PROCMODE**=0). The data and address are multiplexed when **MUXMODE** is asserted, otherwise both **MDATA[7:0]** and **MA[6:0]** busses are used. When **MUXMODE** is asserted, all register addresses must be multiplied by 2. This allows support for 16-bit microprocessor busses

See Section 13.7 for microprocessor interface timing diagrams.

9.0 REGISTER DESCRIPTIONS

The microprocessor configures, controls and monitors operation through the use of the registers defined in this section. The bits labelled "*res*" or "reserved" are reserved and must be set to zero for a write and the value returned by a read is undefined. Reserved registers should not be written and the value returned by a read is undefined. Register addresses are byte-wide.

9.1 CONFIGURATION SEQUENCE

The following sequence should be followed when configuring the device with Rapid Code Reconfiguration disabled. See Sections 9.2 and 9.3 for a description of Rapid Code Reconfiguration.

After a hard reset (**RESETN**), both the encoder and decoder are disabled. This allows the microprocessor to write all appropriate configuration registers before the device accepts data. Similarly, after a soft reset (**EReset** or **DReset**), the encoder or decoder are disabled. The **EStart** or **DStart** bits are set to allow the encoder or decoder to begin processing data. The following sequence should be followed when configuring (and reconfiguring) either the encoder or decoder.

1) Power on reset (**RESETN**).
2) If using frequency synthesis, configure flow control and enable by setting **EFlowMode** or **DFlowMode**.
3) Configure encoder and/or decoder.
4) Encoder: Write **EStart** to 1. This will cause the encoder to begin reading data and encoding. Decoder: Write **DStart** to 1. This will cause the decoder to begin reading channel data, looking for synchronization.
5) If a configuration change is necessary, first write **EReset** or **DReset** to 1.
6) Write new configuration registers, then set **EStart** or **DStart**. Repeat as necessary.

Note that the encoder and decoder are independent paths. Either can be reset while the opposite path is processing data.

9.2 ENCODER RAPID CODE RECONFIGURATION

The AHA4541 contains support for on-the-fly code changing. This is useful in systems that require consecutive blocks of different code types to be encoded. For example, a multi-channel system that uses the same encoder to encode multiple channels, each with a different code type. In this case, the encoder is rapidly switching between code types.

To enable this mode in the encoder, the **ECodeReconfig** bit should be set to a 1. When using this mode, frame synchronization must be added externally, and the **EFSyncInsert** bit must be set to zero. The **EFSyncPer** register must also be written to zero, and the **ESymSize** must be set to 3 or 7 (4 or 8 bits per symbol). Helical interleaving must be set for all blocks, or cleared for all blocks. The CRC and scrambler settings must be the same for all blocks. **EBlockMode** must be set to 1, and **EFlowMode** must be set to 0 or 1. Finally, code changing on-the-fly is not supported when both **EHelical** and (**EModRowRotate** or **EmodPlaneRotate**) are set.

The following registers may be changed for each block: **ECodeX, ECodeY, ECodeZ, EEnhanced, EShortX, EShortY, EShortZ, EShortB, EShortR, EPSize**. Each of these registers is mirrored, so that the microprocessor can write a new code type while the encoder is encoding according to a previous code type. All other registers must be the same for all blocks.

The steps for rapid code reconfiguration code changing are as follows:

1) Set **ECodeReconfig** and write fixed register values.
2) Write code configuration and packet length for first block.
3) Start the encoder by writing the **EStart** bit to a one. The encoder will begin handshaking input data.
4) Configure the device for the code type of the next block. This occurs in parallel with loading of the first block of the first group. When configuration for the next code type is complete, write the **EStart** bit to a 1.
5) Wait for **EProcComp** interrupt. This interrupt indicates that the load of the first block has completed. If **EStart** was set before this interrupt occurs, the encoder will automatically begin encoding this block, and loading the next block.
6) Repeat steps 4 and 5 for each block to be decoded.

Note the when using rapid code reconfiguration, the external block must input the exact number of bytes per block according to the value programmed into **EPSize**. If the encoded TPC block size is not a multiple of the symbol size, the encoder will pad zeros to make the output symbol aligned.

Figure 31 shows the block timings of rapid code reconfiguration with helical interleaving disabled. Note that since the encoder has near zero latency, each block is simultaneously loaded and unloaded from the device.

Figure 32 shows the block timings with helical interleaving enabled. Since the helical interleaver adds a one block latency, the loading of each block must complete before the unloading of the same block begins. The **EProcComp** interrupt is issued at the end of loading each block into the interleaver. Note that no interrupt occurs at the end of unloading. The device supports loading a block of a given code type, while unloading a block of a different code type. The code type for block 2 shown in the figure is much larger than that of the other blocks.

Figure 31: Encoder Rapid Code Reconfiguration with Helical Interleaving Disabled

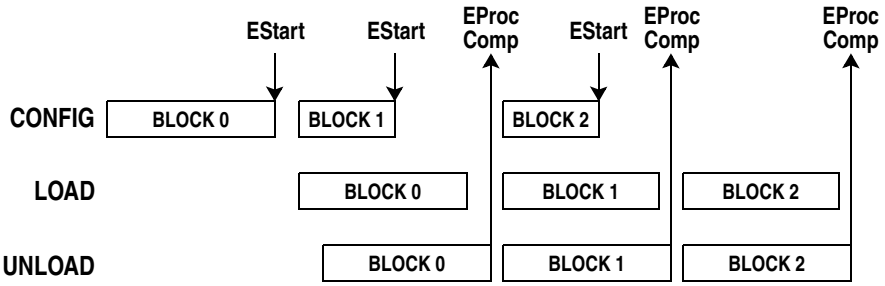
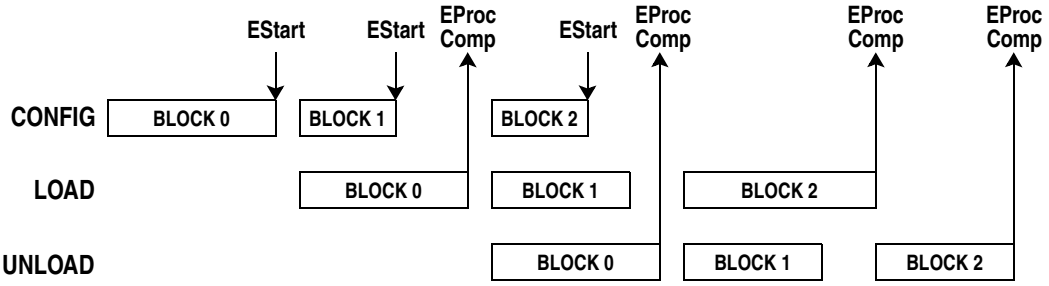


Figure 32: Encoder Rapid Code Reconfiguration with Helical Interleaving Enabled



9.3 DECODER RAPID CODE RECONFIGURATION

Similar to the encoder, the AHA4541 contains support for on-the-fly code changing. To enable this mode in the decoder, the **DCodeReconfig** bit should be set to a 1. When using this mode, frame synchronization must be handled externally, the **DFSyncEnable** bit must be set to zero, and **DScramPer** must be set to 1 if using internal descrambling. The modulation type, CRC and scrambler must be the same for all blocks. **DBlockMode** must be set to 1. **DFlowMode** must be set to 0 or 1.

The following registers may be changed for each block: **DCodeX, DCodeY, DCodeZ, DEnhanced, DShortX, DShortY, DShortZ, DShortB, DShortR, DPSize, FeedbackX, FeedbackY, FeedbackZ, FeedbackH, FeedbackHP, Iterations**. Each of these registers is mirrored, so that the microprocessor can write a new code type while the decoder is decoding according to a previous code type. All other registers must be the same for all blocks.

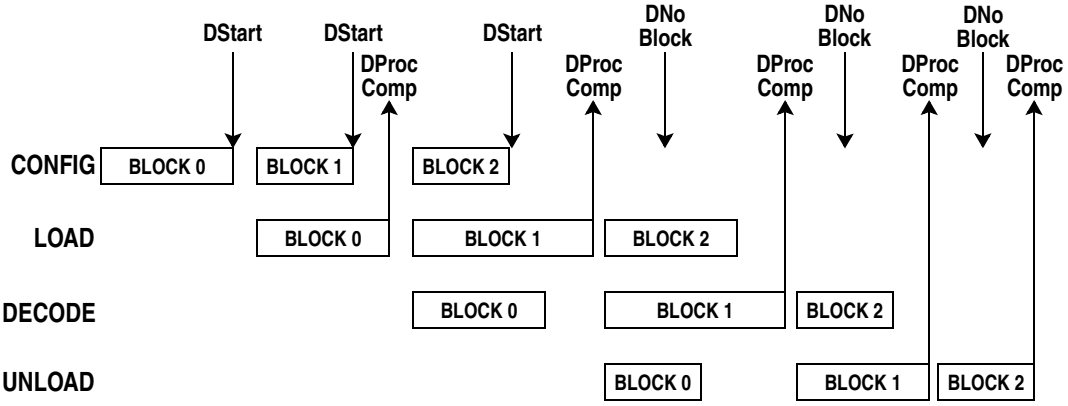
The steps for rapid code reconfiguration code changing are as follows:

1) Set **DCodeReconfig** and write fixed register values.
2) Write code configuration and packet length for first block.
3) Start the decoder by writing the **DStart** bit to a one. The decoder will begin handshaking input data.
4) Configure the device for the code type of the next block. This occurs in parallel with loading of the first block of the first group. When configuration for the next code type is complete, write the **DStart** bit to a 1.
5) Wait for **DProcComp** interrupt. This interrupt indicates that the load of the first block has completed. If **DStart** was set before this interrupt occurs, the decoder will automatically begin decoding this block, and begin loading the next block.
6) Repeat steps 4 and 5 for each block to be decoded.
7) If no additional data is to be input to the device, the **DNoBlock** bit must be set. This has the same effect as writing **DStart**, except that the decoder does not expect a block to be input.

If the decoder finishes processing the block before **DStart** or **DNoBlock** is written, the **DProcComp** interrupt will be issued, and the device will stop processing and wait until one of the two bits is written. The **DProcComp** will occur after **DStart** or **DNoBlock** is written.

Figure 33 shows the block timing when using decoder rapid code reconfiguration. The figure shows a code type for block 1 that is much larger than for the other blocks.

Figure 33: Decoder Rapid Code Reconfiguration



Note that when using rapid code reconfiguration, any data beyond the end of the TPC block, before the next sync input is discarded by the device.

9.4 REGISTER LIST

The register set is broken up into encoder and decoder registers. Register addresses 0 through 0x3f belong to the encoder, while register addresses 0x40 through 0x7f belong to the decoder. Many encoder registers have the same effect as an equivalent decoder register. In these cases, the lower six bits (0 through 5) of each register address will match, with bit 6 indicating encoder or decoder. In addition, these registers are grouped together in the following register description section. All register bits labelled as "res" are reserved and must be written to zero. Reads return unpredictable values.

Table 4: Register Bits - Alphabetical

REGISTER BIT NAME	*ADDRESS**	*BIT*	*REGISTER SECTION*
CACPTPolarity	0x38	2	Section 9.8.3 *Signal Polarity (POLARITY)*
ChannelFormat[1:0]	0x69	3:2	Section 9.7.3 *Decoder Modulation Format (DECMOD0-3)*
CorrAvg[2:0]	0x7a	7:5	Section 9.9.4 *Correction Count (DCORRCNT0-1)*
Corrections[11:8]	0x7a	3:0	
Corrections[7:0]	0x7b	7:0	
CRCLoss	0x3c	3	Section 9.9.5 *Interrupt (INTERRUPT)*
CRCLossM	0x3d	3	Section 9.9.6 *Interrupt Mask (INTMASK)*
CRCSyncOff[7:0]	0x4e	7:0	Section 9.5.4 *CRC Failure Threshold (DCRCTHRSH)*
CRDYPolarity	0x38	3	Section 9.8.3 *Signal Polarity (POLARITY)*
CTransfers[15:8]	0x72	7:0	Section 9.8.1 *Data Flow Configuration (EDATACON0-7, DDATACON0-7)*
CTransfers[23:16]	0x71	7:0	
CTransfers[27:24]	0x70	3:0	
CTransfers[7:0]	0x73	7:0	
DACPTPolarity	0x38	0	Section 9.8.3 *Signal Polarity (POLARITY)*
DBlockMode	0x40	7	Section 9.5.1 *Packet Configuration (EPACKCON0-3, DPACKCON0-3)*
DBufferSize[7:0]	0x5e	7:0	Section 9.8.2 *Buffer Configuration (EBUFCON, DBUFCON0-1)*
DClkAdjMode	0x74	6	Section 9.8.1 *Data Flow Configuration (EDATACON0-7, DDATACON0-7)*
DClkAdjPolarity	0x74	7	
DClkLock	0x3b	5	Section 9.9.2 *Status (STATUS)*
DCodeReconfig	0x51	5	Section 9.6.1 *TPC Constituent Code (ETPCCODE0-1, DTPCCODE0-1)*
DCodeX[3:0]	0x50	7:4	
DCodeY[3:0]	0x50	3:0	
DCodeZ[3:0]	0x51	3:0	
DCRCEnable	0x44	3	Section 9.5.2 *CRC Configuration (ECRCCON0-4, DCRCCON0-4)*
DCRCPoly[15:8]	0x47	7:0	
DCRCPoly[23:16]	0x46	7:0	
DCRCPoly[31:24]	0x45	7:0	
DCRCPoly[7:0]	0x48	7:0	
DCRCSize[2:0]	0x44	2:0	
DecComp	0x3c	0	Section 9.9.5 *Interrupt (INTERRUPT)*
DecCompM	0x3d	0	Section 9.9.6 *Interrupt Mask (INTMASK)*
DEnhanced	0x51	4	Section 9.6.1 *TPC Constituent Code (ETPCCODE0-1, DTPCCODE0-1)*
DFlowMode[2:0]	0x70	7:5	Section 9.8.1 *Data Flow Configuration (EDATACON0-7, DDATACON0-7)*

* *All register addresses must be doubled when MUXMODE is asserted.*

REGISTER BIT NAME	ADDRESS*	BIT	REGISTER SECTION
DFSyncEnable	0x60	5	Section 9.7.1 *Frame Synchronization Mark Configuration (EFRMSYNC0-7, DFRMSYNC0-7)*
DFSyncFreq[3:0]	0x65	3:0	
DFSyncLength[4:0]	0x60	4:0	
DFSyncMark[15:8]	0x63	7:0	
DFSyncMark[23:16]	0x62	7:0	
DFSyncMark[31:24]	0x61	7:0	
DFSyncMark[7:0]	0x64	7:0	
DFSyncPer[14:8]	0x66	6:0	
DFSyncPer[7:0]	0x67	7:0	
DHelical	0x51	7	Section 9.6.1 *TPC Constituent Code (ETPCCODE0-1, DTPCCODE0-1)*
DModPlaneRotate	0x6b	6	Section 9.7.3 *Decoder Modulation Format (DECMOD0-3)*
DModRowRotate	0x6b	7	
DNoBlock	0x79	3	Section 9.9.1 *Control (ECONTROL, DCONTROL)*
DOutputECC	0x51	6	Section 9.6.1 *TPC Constituent Code (ETPCCODE0-1, DTPCCODE0-1)*
DProcComp	0x3c	4	Section 9.9.5 *Interrupt (INTERRUPT)*
DProcCompM	0x3d	4	Section 9.9.6 *Interrupt Mask (INTMASK)*
DPSize[10:8]	0x40	2:0	Section 9.5.1 *Packet Configuration (EPACKCON0-3, DPACKCON0-3)*
DPSize[7:0]	0x41	7:0	
DPSyncLength[1:0]	0x40	4:3	
DPSyncMark[15:8]	0x42	7:0	
DPSyncMark[7:0]	0x43	7:0	
DPSyncRemove	0x40	6	
DRDYPolarity	0x38	1	Section 9.8.3 *Signal Polarity (POLARITY)*
DReset	0x79	0	Section 9.9.1 *Control (ECONTROL, DCONTROL)*
DScramEnable	0x49	4	Section 9.5.3 *Scrambler Configuration (ESCRAMCON0-4, DSCRAMCON0-4)*
DScramPer[3:0]	0x49	3:0	
DScramPoly[15:8]	0x4a	7:0	
DScramPoly[7:0]	0x4b	7:0	
DScramSeed[15:8]	0x4c	7:0	
DScramSeed[7:0]	0x4d	7:0	
DShortB[6:0]	0x55	6:0	Section 9.6.2 *Shortening Configuration (ESHORTCON0-4, DSHORTCON0-4)*
DShortR[6:0]	0x56	6:0	
DShortX[6:0]	0x52	6:0	
DShortY[6:0]	0x53	6:0	
DShortZ[3:0]	0x54	3:0	
DStart	0x79	4	Section 9.9.1 *Control (ECONTROL, DCONTROL)*
DSymSize[2:0]	0x68	2:0	Section 9.7.3 *Decoder Modulation Format (DECMOD0-3)*
DTransfers[15:8]	0x76	7:0	Section 9.8.1 *Data Flow Configuration (EDATACON0-7, DDATACON0-7)*
DTransfers[23:16]	0x75	7:0	
DTransfers[27:24]	0x74	3:0	
DTransfers[7:0]	0x77	7:0	
DumpThresh[4:0]	0x5d	4:0	Section 9.8.2 *Buffer Configuration (EBUFCON, DBUFCON0-1)*
EACPTPolarity	0x38	6	Section 9.8.3 *Signal Polarity (POLARITY)*
EBufferSize[7:0]	0x1e	7:0	Section 9.8.2 *Buffer Configuration (EBUFCON, DBUFCON0-1)*
EBlockMode	0x00	7	Section 9.5.1 *Packet Configuration (EPACKCON0-3, DPACKCON0-3)*

* *All register addresses must be doubled when MUXMODE is asserted.*

REGISTER BIT NAME	ADDRESS*	BIT	REGISTER SECTION
ECodeReconfig	0x11	5	Section 9.6.1 *TPC Constituent Code (ETPCCODE0-1, DTPCCODE0-1)*
ECodeX[3:0]	0x10	7:4	
ECodeY[3:0]	0x10	3:0	
ECodeZ[3:0]	0x11	3:0	
ECRCEnable	0x04	3	Section 9.5.2 *CRC Configuration (ECRCCON0-4, DCRCCON0-4)*
ECRCPoly[15:8]	0x07	7:0	
ECRCPoly[23:16]	0x06	7:0	
ECRCPoly[31:24]	0x05	7:0	
ECRCPoly[7:0]	0x08	7:0	
ECRCSize[2:0]	0x04	2:0	
EEnhanced	0x11	4	Section 9.6.1 *TPC Constituent Code (ETPCCODE0-1, DTPCCODE0-1)*
EFlowMode[2:0]	0x30	7:5	Section 9.8.1 *Data Flow Configuration (EDATACON0-7, DDATACON0-7)*
EFSyncFreq[3:0]	0x25	3:0	Section 9.7.1 *Frame Synchronization Mark Configuration (EFRMSYNC0-7, DFRMSYNC0-7)*
EFSyncInsert	0x20	5	
EFSyncLength[4:0]	0x20	4:0	
EFSyncMark[15:8]	0x23	7:0	
EFSyncMark[23:16]	0x22	7:0	
EFSyncMark[31:24]	0x21	7:0	
EFSyncMark[7:0]	0x24	7:0	
EFSyncPer[14:8]	0x26	6:0	
EFSyncPer[7:0]	0x27	7:0	
EHelical	0x11	7	Section 9.6.1 *TPC Constituent Code (ETPCCODE0-1, DTPCCODE0-1)*
EModPlaneRotate	0x28	6	Section 9.7.2 *Encoder Symbol Mapping (ENCMAP0-1)*
EModRowRotate	0x28	7	
EPassThru	0x11	6	Section 9.6.1 *TPC Constituent Code (ETPCCODE0-1, DTPCCODE0-1)*
EProcComp	0x3c	5	Section 9.9.5 *Interrupt (INTERRUPT)*
EProcCompM	0x3d	5	Section 9.9.6 *Interrupt Mask (INTMASK)*
EPSize[10:8]	0x00	2:0	Section 9.5.1 *Packet Configuration (EPACKCON0-3, DPACKCON0-3)*
EPSize[7:0]	0x01	7:0	
EPSyncInsert	0x00	6	
EPSyncLength[1:0]	0x00	4:3	
EPSyncMark[15:8]	0x02	7:0	
EPSyncMark[7:0]	0x03	7:0	
ERDYPolarity	0x38	7	Section 9.8.3 *Signal Polarity (POLARITY)*
EReset	0x39	0	Section 9.9.1 *Control (ECONTROL, DCONTROL)*
EScramEnable	0x09	4	Section 9.5.3 *Scrambler Configuration (ESCRAMCON0-4, DSCRAMCON0-4)*
EScramPer[3:0]	0x09	3:0	
EScramPoly[15:8]	0x0a	7:0	
EScramPoly[7:0]	0x0b	7:0	
EScramSeed[15:8]	0x0c	7:0	
EScramSeed[7:0]	0x0d	7:0	
EShortB[6:0]	0x15	6:0	Section 9.6.2 *Shortening Configuration (ESHORTCON0-4, DSHORTCON0-4)*
EShortR[6:0]	0x16	6:0	
EShortX[6:0]	0x12	6:0	
EShortY[6:0]	0x13	6:0	
EShortZ[3:0]	0x14	3:0	
EStart	0x39	4	Section 9.9.1 *Control (ECONTROL, DCONTROL)*

* *All register addresses must be doubled when MUXMODE is asserted.*

REGISTER BIT NAME	ADDRESS*	BIT	REGISTER SECTION
ESymMap[7:0]	0x29	7:0	Section 9.7.2 *Encoder Symbol Mapping (ENCMAP0-1)*
ESymMapEnable	0x28	4	
ESymSize[2:0]	0x28	2:0	
ETransfers[15:8]	0x32	7:0	Section 9.8.1 *Data Flow Configuration (EDATACON0-7, DDATACON0-7)*
ETransfers[23:16]	0x31	7:0	
ETransfers[27:24]	0x30	3:0	
ETransfers[7:0]	0x33	7:0	
FBitThresh[1:0]	0x6c	7:6	Section 9.7.4 *Decoder Synchronization Control (DECSYNC0-3)*
FBitThresh[2]	0x6d	7	
FeedbackH[4:0]	0x5a	4:0	Section 9.6.3 *Feedback (DFEEDBCK0-3)*
FeedbackHP[4:0]	0x5b	4:0	
FeedbackX[4:0]	0x58	4:0	
FeedbackY[2:0]	0x58	7:5	
FeedbackY[4:3]	0x59	6:5	
FeedbackZ[4:0]	0x59	4:0	
FSynced	0x3b	3	Section 9.9.2 *Status (STATUS)*
FSyncInv	0x6e	7	Section 9.7.4 *Decoder Synchronization Control (DECSYNC0-3)*
FSyncOff[6:0]	0x6d	6:0	
FSyncOn[5:0]	0x6c	5:0	
FSyncTime[5:0]	0x6e	5:0	
GOut0Config[2:0]	0x3a	6:4	Section 9.9.3 *General Output (OUTPUT)*
GOut1Config[2:0]	0x3a	2:0	
Iterations[7:0]	0x5c	7:0	Section 9.6.4 *Iterations (DITER)*
LLRExponent[3:0]	0x6b	3:0	Section 9.7.3 *Decoder Modulation Format (DECMOD0-3)*
LLRMantissa[7:0]	0x6a	7:0	
PreRotate	0x68	7	
PSynced	0x3b	4	Section 9.9.2 *Status (STATUS)*
PSyncOn[5:0]	0x6f	5:0	Section 9.7.4 *Decoder Synchronization Control (DECSYNC0-3)*
QBits[2:0]	0x69	6:4	Section 9.7.3 *Decoder Modulation Format (DECMOD0-3)*
QMode[1:0]	0x69	1:0	
ResetMapPtr	0x39	5	Section 9.9.1 *Control (ECONTROL, DCONTROL)*
Resync	0x79	1	
RotateEnable	0x69	7	Section 9.7.3 *Decoder Modulation Format (DECMOD0-3)*
Rotation[2:0]	0x3b	2:0	Section 9.9.2 *Status (STATUS)*
StopIter	0x5b	7	Section 9.6.3 *Feedback (DFEEDBCK0-3)*
SyncAcq	0x3c	1	Section 9.9.5 *Interrupt (INTERRUPT)*
SyncAcqM	0x3d	1	Section 9.9.6 *Interrupt Mask (INTMASK)*
SyncLoss	0x3c	2	Section 9.9.5 *Interrupt (INTERRUPT)*
SyncLossM	0x3d	2	Section 9.9.6 *Interrupt Mask (INTMASK)*
UACPTPolarity	0x38	4	Section 9.8.3 *Signal Polarity (POLARITY)*
UClkAdjMode	0x34	6	Section 9.8.1 *Data Flow Configuration (EDATACON0-7, DDATACON0-7)*
UClkAdjPolarity	0x34	7	
UClkLock	0x3b	6	Section 9.9.2 *Status (STATUS)*
URDYPolarity	0x38	5	Section 9.8.3 *Signal Polarity (POLARITY)*
UTransfers[15:8]	0x36	7:0	Section 9.8.1 *Data Flow Configuration (EDATACON0-7, DDATACON0-7)*
UTransfers[23:16]	0x35	7:0	
UTransfers[27:24]	0x34	3:0	
UTransfers[7:0]	0x37	7:0	
Version[7:0]	0x3f	7:0	Section 9.10.1 *Version*

* *All register addresses must be doubled when MUXMODE is asserted.*

Table 5: Register Description by Address

ADDRESS MMODE=0	ADDRESS MMODE=1	R/W	MNEMONIC	REGISTER NAME	HARD RESET
0x00	0x00	R/W	EPACKCON0	Encoder Packet Configuration 0	0x00
0x01	0x02	R/W	EPACKCON1	Encoder Packet Configuration 1	0x00
0x02	0x04	R/W	EPACKCON2	Encoder Packet Configuration 2	0x00
0x03	0x06	R/W	EPACKCON3	Encoder Packet Configuration 3	0x00
0x04	0x08	R/W	ECRCCON0	Encoder CRC Configuration 0	0x00
0x05	0x0a	R/W	ECRCCON1	Encoder CRC Configuration 1	0x00
0x06	0x0c	R/W	ECRCCON2	Encoder CRC Configuration 2	0x00
0x07	0x0e	R/W	ECRCCON3	Encoder CRC Configuration 3	0x00
0x08	0x10	R/W	ECRCCON4	Encoder CRC Configuration 4	0x00
0x09	0x12	R/W	ESCRAMCON0	Encoder Scrambler Configuration 0	0x00
0x0a	0x14	R/W	ESCRAMCON1	Encoder Scrambler Configuration 1	0x00
0x0b	0x16	R/W	ESCRAMCON2	Encoder Scrambler Configuration 2	0x00
0x0c	0x18	R/W	ESCRAMCON3	Encoder Scrambler Configuration 3	0x00
0x0d	0x1a	R/W	ESCRAMCON4	Encoder Scrambler Configuration 4	0x00
0x10	0x20	R/W	ETPCCODE0	Encoder TPC Constituent Code 0	0x00
0x11	0x22	R/W	ETPCCODE1	Encoder TPC Constituent Code 1	0x00
0x12	0x24	R/W	ESHORTCON0	Encoder Shortening Configuration 0	0x00
0x13	0x26	R/W	ESHORTCON1	Encoder Shortening Configuration 1	0x00
0x14	0x28	R/W	ESHORTCON2	Encoder Shortening Configuration 2	0x00
0x15	0x2a	R/W	ESHORTCON3	Encoder Shortening Configuration 3	0x00
0x16	0x2c	R/W	ESHORTCON4	Encoder Shortening Configuration 4	0x00
0x1e	0x3c	R/W	EBUFCON	Encoder Buffer Configuration	0x00
0x20	0x40	R/W	EFRMSYNC0	Encoder Frame Synchronization Mark Configuration 0	0x00
0x21	0x42	R/W	EFRMSYNC1	Encoder Frame Synchronization Mark Configuration 1	0x00
0x22	0x44	R/W	EFRMSYNC2	Encoder Frame Synchronization Mark Configuration 2	0x00
0x23	0x46	R/W	EFRMSYNC3	Encoder Frame Synchronization Mark Configuration 3	0x00
0x24	0x48	R/W	EFRMSYNC4	Encoder Frame Synchronization Mark Configuration 4	0x00
0x25	0x4a	R/W	EFRMSYNC5	Encoder Frame Synchronization Mark Configuration 5	0x00
0x26	0x4c	R/W	EFRMSYNC6	Encoder Frame Synchronization Mark Configuration 6	0x00
0x27	0x4e	R/W	EFRMSYNC7	Encoder Frame Synchronization Mark Configuration 7	0x00
0x28	0x50	R/W	ENCMAP0	Encoder Symbol Mapping 0	0x00
0x29	0x52	R/W	ENCMAP1	Encoder Symbol Mapping 1	0x00
0x30	0x60	R/W	EDATACON0	Encoder Data Flow Configuration 0	0x00
0x31	0x62	R/W	EDATACON1	Encoder Data Flow Configuration 1	0x00
0x32	0x64	R/W	EDATACON2	Encoder Data Flow Configuration 2	0x00
0x33	0x66	R/W	EDATACON3	Encoder Data Flow Configuration 3	0x00
0x34	0x68	R/W	EDATACON4	Encoder Data Flow Configuration 4	0x00
0x35	0x6a	R/W	EDATACON5	Encoder Data Flow Configuration 5	0x00
0x36	0x6c	R/W	EDATACON6	Encoder Data Flow Configuration 6	0x00
0x37	0x6e	R/W	EDATACON7	Encoder Data Flow Configuration 7	0x00
0x38	0x70	R/W	POLARITY	Signal Polarity	0x00
0x39	0x72	R/W	ECONTROL	Encoder Control	0x00

ADDRESS MMODE=0	ADDRESS MMODE=1	R/W	MNEMONIC	REGISTER NAME	HARD RESET
0x3a	0x74	R/W	OUTPUT	General Output	0x00
0x3b	0x76	R	STATUS	Status	0x00
0x3c	0x78	R	INTERRUPT	Interrupt	0x00
0x3d	0x7a	R/W	INTMASK	Interrupt Mask	0xff
0x3e	0x7e	R	VERSION	Version	0x41
0x40	0x80	R/W	DPACKCON0	Decoder Packet Configuration 0	0x00
0x41	0x82	R/W	DPACKCON1	Decoder Packet Configuration 1	0x00
0x42	0x84	R/W	DPACKCON2	Decoder Packet Configuration 2	0x00
0x43	0x86	R/W	DPACKCON3	Decoder Packet Configuration 3	0x00
0x44	0x88	R/W	DCRCCON0	Decoder CRC Configuration 0	0x00
0x45	0x8a	R/W	DCRCCON1	Decoder CRC Configuration 1	0x00
0x46	0x8c	R/W	DCRCCON2	Decoder CRC Configuration 2	0x00
0x47	0x8e	R/W	DCRCCON3	Decoder CRC Configuration 3	0x00
0x48	0x90	R/W	DCRCCON4	Decoder CRC Configuration 4	0x00
0x49	0x92	R/W	DSCRAMCON0	Decoder Scrambler Configuration 0	0x00
0x4a	0x94	R/W	DSCRAMCON1	Decoder Scrambler Configuration 1	0x00
0x4b	0x96	R/W	DSCRAMCON2	Decoder Scrambler Configuration 2	0x00
0x4c	0x98	R/W	DSCRAMCON3	Decoder Scrambler Configuration 3	0x00
0x4d	0x9a	R/W	DSCRAMCON4	Decoder Scrambler Configuration 4	0x00
0x4e	0x9c	R/W	DCRCTHRSH	Decoder CRC Failure Threshold	0x00
0x50	0xa0	R/W	DTPCCODE0	Decoder TPC Constituent Code 0	0x00
0x51	0xa1	R/W	DTPCCODE1	Decoder TPC Constituent Code 1	0x00
0x52	0xa4	R/W	DSHORTCON0	Decoder Shortening Configuration 0	0x00
0x53	0xa6	R/W	DSHORTCON1	Decoder Shortening Configuration 1	0x00
0x54	0xa8	R/W	DSHORTCON2	Decoder Shortening Configuration 2	0x00
0x55	0xaa	R/W	DSHORTCON3	Decoder Shortening Configuration 3	0x00
0x56	0xac	R/W	DSHORTCON4	Decoder Shortening Configuration 4	0x00
0x58	0xb0	R/W	DFEEDBCK0	Decoder Feedback 0	0x00
0x59	0xb2	R/W	DFEEDBCK1	Decoder Feedback 1	0x00
0x5a	0xb4	R/W	DFEEDBCK2	Decoder Feedback 2	0x00
0x5b	0xb6	R/W	DFEEDBCK3	Decoder Feedback 3	0x00
0x5c	0xb8	R/W	DITER	Decoder Iterations	0x00
0x5d	0xba	R/W	DBUFCON0	Decoder Buffer Configuration 0	0x00
0x5e	0xbc	R/W	DBUFCON1	Decoder Buffer Configuration 1	0x00
0x60	0xc0	R/W	DFRMSYNC0	Decoder Frame Synchronization Mark Configuration 0	0x00
0x61	0xc2	R/W	DFRMSYNC1	Decoder Frame Synchronization Mark Configuration 1	0x00
0x62	0xc4	R/W	DFRMSYNC2	Decoder Frame Synchronization Mark Configuration 2	0x00
0x63	0xc6	R/W	DFRMSYNC3	Decoder Frame Synchronization Mark Configuration 3	0x00
0x64	0xc8	R/W	DFRMSYNC4	Decoder Frame Synchronization Mark Configuration 4	0x00
0x65	0xca	R/W	DFRMSYNC5	Decoder Frame Synchronization Mark Configuration 5	0x00
0x66	0xcc	R/W	DFRMSYNC6	Decoder Frame Synchronization Mark Configuration 6	0x00
0x67	0xce	R/W	DFRMSYNC7	Decoder Frame Synchronization Mark Configuration 7	0x00
0x68	0xd0	R/W	DECMOD0	Decoder Modulation 0	0x00

ADDRESS MMODE=0	ADDRESS MMODE=1	R/W	MNEMONIC	REGISTER NAME	HARD RESET
0x69	0xd2	R/W	DECMOD1	Decoder Modulation 1	0x00
0x6a	0xd4	R/W	DECMOD2	Decoder Modulation 2	0x00
0x6b	0xd6	R/W	DECMOD3	Decoder Modulation 3	0x00
0x6c	0xd8	R/W	DECSYNC0	Decoder Synchronization Control 0	0x00
0x6d	0xda	R/W	DECSYNC1	Decoder Synchronization Control 1	0x00
0x6e	0xdc	R/W	DECSYNC2	Decoder Synchronization Control 2	0x00
0x6f	0xde	R/W	DECSYNC3	Decoder Synchronization Control 3	0x00
0x70	0xe0	R/W	DDATACON0	Decoder Data Flow Configuration 0	0x00
0x71	0xe2	R/W	DDATACON1	Decoder Data Flow Configuration 1	0x00
0x72	0xe4	R/W	DDATACON2	Decoder Data Flow Configuration 2	0x00
0x73	0xe6	R/W	DDATACON3	Decoder Data Flow Configuration 3	0x00
0x74	0xe8	R/W	DDATACON4	Decoder Data Flow Configuration 4	0x00
0x75	0xea	R/W	DDATACON5	Decoder Data Flow Configuration 5	0x00
0x76	0xec	R/W	DDATACON6	Decoder Data Flow Configuration 6	0x00
0x77	0xee	R/W	DDATACON7	Decoder Data Flow Configuration 7	0x00
0x79	0xf2	R/W	DCONTROL	Decoder Control	0x00
0x7a	0xf4	R/W	DCORRCNT0	Decoder Correction Count 0	0x00
0x7b	0xf6	R/W	DCORRCNT1	Decoder Correction Count 1	0x00

9.5 USER DATA FORMATTING REGISTERS

9.5.1 PACKET CONFIGURATION (EPACKCON0-3, DPACKCON0-3)

Read/Write
Reset Value (hex)= 00 00 00 00 00 00 00 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x00	0x00	**EBlock Mode**	**EPSync Insert**	*res*	**EPSyncLength[1:0]**		**EPSize[10:8]**		
0x01	0x02	**EPSize[7:0]**							
0x02	0x04	**EPSyncMark[15:8]**							
0x03	0x06	**EPSyncMark[7:0]**							
0x40	0x80	**DBlock Mode**	**DPSync Remove**	*res*	**DPSyncLength[1:0]**		**DPSize[10:8]**		
0x41	0x82	**DPSize[7:0]**							
0x42	0x84	**DPSyncMark[15:8]**							
0x43	0x86	**DPSyncMark[7:0]**							

xBlockMode - Block Mode. When set, the data packing mode is set to Block. When cleared, the data packing mode is set to Stream, and the device will assume that packet synchronization marks will be in the frame structure.

xPSyncLength[1:0] - Packet Sync Mark length. The length of the packet sync mark is (**xPSyncLength[1:0]**+1)x4. **xPSyncLength[1:0]** must be set to 1 or 3 (8 or 16 bits) when **EPSyncInsert** / **DPSyncRemove** is cleared.

EPSyncInsert - Insert Packet Sync Mark. When this bit is set in Stream Mode, the encoder will insert the packet sync mark configured by **EPSyncLength[1:0]** and **EPSyncMark[15:0]**. Note that an inverted synchronization mark is inserted every **EScramPer[3:0]** packets, to allow synchronization of the descrambler. When cleared in Stream Mode, the encoder will not insert packet sync marks. If the data stream input to the encoder already contains a packet sync mark, setting **EPSyncInsert** to a zero will cause the encoder to ignore the sync mark for CRC and scrambling. It will assume that the sync mark is of the length specified in **EPSyncLength[1:0]**. These pre-inserted sync marks are inverted by the scrambler every **EScramPer[3:0]** packets for descrambler synchronization. This mode is useful when transmitting MPEG packet data, that already contains packet synchronization marks. This bit should be cleared in Block Mode.

DPSyncRemove - Remove Packet Sync Mark. When this bit is set in Stream Mode, the decoder will remove the packet sync marks from the data stream, so that they are not output from the device. When cleared, the decoder leaves the packet sync marks in the data stream. Every inverted packet sync mark is again inverted to return to the non-inverted state. When in Block Mode, packet synchronization marks are not in the data stream, and this bit must be set to 0.

xPSize[10:0] - User Data Packet Size. The size of a packet, in bytes, not including the packet synchronization or CRC bits. A value of 0 represents 2048 bytes.

xPSyncMark[15:0] - User Packet Synchronization Mark. The bit pattern of the synchronization mark. Note that the least significant bit of this register is the first bit of the mark inserted into the data stream. The length of the mark is determined by **xPSyncLength[1:0]**, and the mark value should be written right justified into this register.

9.5.2 CRC CONFIGURATION (ECRCCON0-4, DCRCCON0-4)

Read/Write
Reset Value (hex) = 00 00 00 00 00 00 00 00 00 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x04	0x08	*res*				**ECRCEnable**	**ECRCSize[2:0]**		
0x05	0x0a	**ECRCPoly[31:24]**							
0x06	0x0c	**ECRCPoly[23:16]**							
0x07	0x0e	**ECRCPoly[15:8]**							
0x08	0x10	**ECRCPoly[7:0]**							
0x44	0x88	*res*				**DCRCEnable**	**DCRCSize[2:0]**		
0x45	0x8a	**DCRCPoly[31:24]**							
0x46	0x8c	**DCRCPoly[23:16]**							
0x47	0x8e	**DCRCPoly[15:8]**							
0x48	0x90	**DCRCPoly[7:0]**							

xCRCEnable - CRC Enable. When set, the encoder will compute and insert CRC bits, and the decoder will compute, check and remove CRC.

xCRCSize[2:0] - CRC Size, in nibbles. The size of the CRC, in bits, is **xCRCSize[2:0]** x 4. A value of 0 represents 32 bits.

xCRCPoly[31:0] - CRC Polynomial. Refer to Figure 5 on Page 5 for configuration of the polynomial.

9.5.3 SCRAMBLER CONFIGURATION (ESCRAMCON0-4, DSCRAMCON0-4)

Read/Write
Reset Value (hex) = 00 00 00 00 00 00 00 00 00 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x09	0x12	*res*			**EScram Enable**	**EScramPer[3:0]**			
0x0a	0x14	**EScramPoly[15:8]**							
0x0b	0x16	**EScramPoly[7:0]**							
0x0c	0x18	**EScramSeed[15:8]**							
0x0d	0x1a	**EScramSeed[7:0]**							
0x49	0x92	*res*			**DScram Enable**	**DScramPer[3:0]**			
0x4a	0x94	**DScramPoly[15:8]**							
0x4b	0x96	**DScramPoly[7:0]**							
0x4c	0x98	**DScramSeed[15:8]**							
0x4d	0x9a	**DScramSeed[7:0]**							

xScramEnable - Scrambler Enable. When set, the scrambler or descrambler will scramble the data according to the polynomial. When cleared, the scrambler or descrambler is disabled.

xScramPoly[15:0] - Scrambler polynomial. For each exponential term in the polynomial, X^N, a binary one should be loaded at bit location N-1. X^0 is assumed to be a term in the polynomial. For example, the polynomial $1+X^3+X^5+X^8$ is programmed as 0b0000000010010100. Note that the length of the PRBS is determined by the most significant bit location containing a one in **xScramPoly[15:0]**. Refer to Figure 6 on Page 7 for configuration of the polynomial.

xScramSeed[15:0] - Seed for Descrambler. The PRBS is reset to this seed at the beginning of every **xScramPer[3:0]** packets. Refer to Figure 6 on Page 7 for configuration of the seed.

xScramPer[3:0] - Scrambler Period. The scrambler is reset to the value programmed in **xScramSeed[15:0]** at the beginning of every **xScramPer[3:0]** packets. A value of 0 represents 16. Note that if packet sync marks are being used, the sync mark that is aligned with the scrambler period is inverted at the encoder and the decoder regardless of the setting of **EPSyncInsert** or **DPSyncRemove**. This inversion does not occur when **xScramPer[3:0]** is set to 1. When **xBlockMode** = 1 or **xScramEnable** = 0, this value should be programmed to 1.

9.5.4 CRC FAILURE THRESHOLD (DCRCTHRSH)

Read/Write
Reset Value (hex) = 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x4e	0x9c	**CRCSyncOff[7:0]**							

CRCSyncOff[7:0] -CRC failure for resync threshold. The CRC comparator keeps a running count of failed CRCs that is reset with each CRC that passes. If this count exceeds the **CRCSyncOff[7:0]** value, then a **CRCLoss** interrupt is generated, the **Synced** bit is reset, and the sync block begins resynchronization. When **CRCSyncOff[7:0]** is set to zero, the automatic resync on CRC failure feature is disabled.

9.6 CODE CONFIGURATION REGISTERS

9.6.1 TPC CONSTITUENT CODE (ETPCCODE0-1, DTPCCODE0-1)

Read/Write
Reset Value (hex) = 00 00 00 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x10	0x20	**ECodeX[3:0]**				**ECodeY[3:0]**			
0x11	0x22	**EHelical**	**EPass Thru**	**ECode Reconfig**	**EEnhanced**	**ECodeZ[3:0]**			
0x50	0xa0	**DCodeX[3:0]**				**DCodeY[3:0]**			
0x51	0xa2	**DHelical**	**DOutput ECC**	**DCode Reconfig**	**DEnhanced**	**DCodeZ[3:0]**			

xCodeX[3:0] - X axis code. **xCodeX[3]** determines whether the code is an extended Hamming or parity only code. When set, the code is a parity only code, when cleared, it is an extended Hamming code. **xCodeX[2:0]** determines the size of the code, as follows

xCode[2:0] †*	*EXTENDED HAMMING*	*PARITY ONLY*
0x2	N/A	(4,3)*
0x3	(8,4)	(8,7)
0x4	(16,11)	(16,15)
0x5	(32,26)	(32,31)
0x6	(64,57)	(64,63)
0x7	(128,120)	(128,127)

† xCode applies to ECodeX, ECodeY, ECodeZ, DCodeX, DCodeY, and DCodeZ.*
** (4,3) parity code is not allowed for xCodeX[3:0].*

xCodeY[3:0] - Y axis code. Defined the same as **xCodeX[3:0]**, except the (4,3) code is allowed.

xEnhanced - enhanced TPC Enable. When set, a hyper-diagonal axis is added to the code. This adds one additional row of parity bits to 2-dimensional codes, and one addition plane of parity bits to 3-dimensional codes. When **xEnhanced** is set with a 2-dimensional code, **xShortY** must be greater than or equal to 1. When **xEnhanced** is set with a 3-dimensional code, **xShortZ** must be greater than or equal to 1.

xCodeZ[3:0] - Z axis code. Defined the same as **xCodeX[3:0]**. Set **xCodeZ[3:0]** to 0 for two dimensional codes. The maximum length for code Z is 16 bits and the (4,3) code is allowed.

xCodeReconfig - Rapid Code Reconfiguration Enable. When set, the rapid code reconfiguration option is enabled. When cleared, this option is disabled. See Section 9.2 *Encoder Rapid Code Reconfiguration* and Section 9.3 *Decoder Rapid Code Reconfiguration* for details.

EPassThru - Encoder Pass Through. When set, the encoder does not add ECC bits. The block size is still configured by the code configuration, but the encoder simply passes all n bits through from the input to the output.

DOutputECC - Decoder Output ECC. When set, the decoder will output both the user data and the TPC ECC bits. Packet synchronization, descrambling, and CRC checking must be disabled when this bit is set. The **DPSize** must be set to match the size of the entire TPC block.

xHelical - Helical Interleaver/Deinterleaver Enable. See Section 3.6 for a description of helical interleaving.

9.6.2 SHORTENING CONFIGURATION (ESHORTCON0-4, DSHORTCON0-4)

Read/Write
Reset Value (hex) = 00 00 00 00 00 00 00 00 00 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x12	0x24	*res*	**EShortX[6:0]**						
0x13	0x26	*res*	**EShortY[6:0]**						
0x14	0x28	*res*				**EShortZ[3:0]**			
0x15	0x2a	res	**EShortB[6:0]**						
0x16	0x2c	res	**EShortR[6:0]**						
0x52	0xa4	*res*	**DShortX[6:0]**						
0x53	0xa6	*res*	**DShortY[6:0]**						
0x54	0xa8	*res*				**DShortZ[3:0]**			
0x55	0xaa	res	**DShortB[6:0]**						
0x56	0xac	res	**DShortR[6:0]**						

xShortX[6:0] - Number of bits to shorten from the X axis code. This value must be set to zero when helical interleaving or enhanced TPC is enabled.

xShortY[6:0] - Number of bits to shorten from the Y axis code. For 2-dimensional codes, this value must be greater than or equal to 1 if enhanced TPC is enabled. For 3-dimensional codes, this value must be set to zero when helical interleaving or enhanced TPC is enabled.

xShortZ[3:0] - Number of bits to shorten from the Z axis code. For 2-dimensional codes, this value must be set to zero. For 3-dimensional codes, this value must be greater than or equal to 1 if enhanced TPC is enabled.

xShortB[6:0] - Number of bits to shorten from the first row of a 2-dimensional code, or from the **xShortR**+1 row of the top plane of a 3-dimensional code. This value must be set to zero when helical interleaving is enabled.

xShortR[6:0] - Number of full rows to shorten from the top plane of a 3-dimensional code. For 2-dimensional codes, this value must be set to zero. This value must be set to zero when helical interleaving is enabled.

9.6.3 FEEDBACK (DFEEDBCK0-3)

Read/Write
Reset Value (hex) = 00 00 00 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x58	0xb0	**FeedbackY[2:0]**			**FeedbackX[4:0]**				
0x59	0xb2	*res*	**FeedbackY[4:3]**		**FeedbackZ[4:0]**				
0x5a	0xb4	*res*			**FeedbackH[4:0]**				
0x5b	0xb6	**StopIter**	*res*		**FeedbackHP[4:0]**				

FeedbackX[4:0] - Feedback for the X axis. The output of the SISO is multiplied by **FeedbackX[4:0]**, then shifted by 5 (divided by 32) before being input to following iterations.

FeedbackY[4:0] - Feedback for the Y axis.

FeedbackZ[4:0] - Feedback for the Z axis.

FeedbackH[4:0] - Feedback for the hyper axis.

FeedbackHP[4:0] - Feedback for the hyper parity row.

StopIter - When set, decoder will stop iterating when convergence is detected.

Note: Refer to Section 4.3.3 Feedback for a description of the feedback values.

9.6.4 ITERATIONS (DITER)

Read/Write
Reset Value (hex) = 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x5c	0xb8	**Iterations[7:0]**							

Iterations[7:0] - Maximum number of iterations to perform. When set to zero, the decoder outputs the hard decision values for each bit with no corrections.

9.7 CHANNEL INTERFACE REGISTERS

9.7.1 FRAME SYNCHRONIZATION MARK CONFIGURATION (EFRMSYNC0-7, DFRMSYNC0-7)

Read/Write
Reset Value (hex) = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x20	0x40	*res*		**EFSyncInsert**	**EFSyncLength[4:0]**				
0x21	0x42	**EFSyncMark[31:24]**							
0x22	0x44	**EFSyncMark[23:16]**							
0x23	0x46	**EFSyncMark[15:8]**							
0x24	0x48	**EFSyncMark[7:0]**							
0x25	0x4a	*res*				**EFSyncFreq[3:0]**			
0x26	0x4c	*res*	**EFSyncPer[14:8]**						
0x27	0x4e	**EFSyncPer[7:0]**							
0x60	0xc0	*res*		**DFSyncEnable**	**DFSyncLength[4:0]**				
0x61	0xc2	**DFSyncMark[31:24]**							
0x62	0xc4	**DFSyncMark[23:16]**							
0x63	0xc6	**DFSyncMark[15:8]**							
0x64	0xc8	**DFSyncMark[7:0]**							
0x65	0xca	*res*				**DFSyncFreq[3:0]**			
0x66	0xcc	*res*	**DFSyncPer[14:8]**						
0x67	0xce	**DFSyncPer[7:0]**							

EFSyncInsert - Insert Frame Sync Mark at Encoder. When set, the encoder will insert the synchronization mark defined here. When cleared, no synchronization mark is inserted, the **EFSyncPer[14:0]** value should be set equal to the desired size of the entire frame, and **EFSyncFreq[3:0]** should be set to one.

DFSyncEnable - Enable Decoder Synchronization. When set, the decoder will automatically sync to the data stream by searching for the programmed sync mark. When cleared, external logic must generate the **CSTART** signal, indicating the beginning of a TPC block.

xFSyncLength[4:0] - Sync Mark Length. The length of the synchronization mark, in bits. Valid values for sync length are from 4 to 32, where 32 is written into the register as 0.

xFSyncMark[31:0] - Synchronization Mark. The non-inverted synchronization mark pattern is written into bits (**xFSyncLength[4:0]**-1:0). The least significant bit of this value is the first bit transmitted over the channel. The device will operate with any synchronization pattern, however improved synchronization is achieved using marks with low auto-correlation.

xFSyncFreq[3:0] - Inverted Synchronization Mark Frequency. The number of synchronization marks expected to include one inverted synchronization mark. If set to 1, no inverted synchronization marks are expected, and a value of 0 represents 16.

xFSyncPer[14:0] - Sync Mark Period. Sync marks are expected every **xFSyncPer[14:0]** bits. Note that the **xFSyncPer[14:0]** value includes the length of the synchronization mark. When sync insertion is not used, this value should be set to the desired length of the entire frame. This may be equal to the encoded TPC block size, or a larger value can be used to guarantee that the encoded block size is a multiple of the symbol size. In this case, the device will automatically pad any data between the TPC block size and the frame size with zeros, and remove these at the decoder. A value of zero for the decoder is illegal. A value of zero for the encoder is used only for rapid code reconfiguration, and is illegal otherwise.

9.7.2 ENCODER SYMBOL MAPPING (ENCMAP0-1)

Read/Write (0x28), Write Only (0x29)
Reset Value (hex) = 00 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x28	0x50	**EModRow Rotate**	**EModPlane Rotate**	*res*	**ESymMap Enable**	*res*	**ESymSize[2:0]**		
0x29	0x52	**ESymMap[7:0]**							

ESymSize[2:0] - Modulated Symbol Size. The number of bits output per clock across the **EDATA[7:0]** bus is **ESymSize[2:0]**+1.

ESymMapEnable - Symbol Map Table Enable. When set, the symbol map table is enabled. When disabled, the encoded data is output directly, with no mapping.

ESymMap[7:0] - Symbol Map Table Access. Each write to this address causes a write to the location pointed to by the symbol map pointer in the symbol map table. The pointer is automatically incremented, causing the next write to occur at the next location in the symbol map table. The pointer can be reset by writing a one to the **ResetMapPtr** bit. The symbol map RAM table is not readable.

EModRowRotate - Setting this bit enables symbol interleaving rotation after each row in the TPC block. This can be used with higher order modulation to spread the less confident bits in a symbol across all axes of the array. See Section 3.5 for a description of when to enable this option. Encoder rotates right.

EModPlaneRotate - Setting this bit enables symbol interleaving after each plane in the TPC block. This can be used with higher order modulation to spread the less confident bits in a symbol across all axes of the array. See Section 3.5 for a description of when to enable this option. The encoder frame size, not including sync marks must be a multiple of four when **EModRowRotate** is set.

9.7.3 DECODER MODULATION FORMAT (DECMOD0-3)

Read/Write
Reset Value (hex) = 00 00 00 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x68	0xd0	**PreRotate**	*res*				**DSymSize[2:0]**		
0x69	0xd2	**Rotate Enable**	**QBits[2:0]**			**ChannelFormat[1:0]**		**QMode[1:0]**	
0x6a	0xd4	**LLRMantissa[7:0]**							
0x6b	0xd6	**DModRow Rotate**	**DModPlane Rotate**	*res*		**LLRExponent[3:0]**			

PreRotate - Pi/8 Rotate Incoming Symbols. The device expects incoming 8-PSK symbols to be aligned at 22.5 (Pi/8) and 67.5 degrees. If the demodulator aligns the symbols at 0 and 45 degrees, this bit should be set. The device will rotate the incoming symbols by 22.5 degrees before processing.

DSymSize[2:0] - Modulation Format. The number of bits per symbol is **DSymSize** + 1. See Section 4.2.1 *Channel Input Formatting*, for a description of input signal wiring.

RotateEnable - Enable internal symbol rotation. When set, the synchronization logic will rotate the phase internally until synchronization is achieved. When cleared, no internal rotation is done. In this case, external logic must monitor the **ROTATE** output signal and execute an required rotation.

QBits[2:0] - Quantization bits. A value of 0 represents 8 bits. When **ChannelFormat** is set to 2 or 3, the maximum value for **QBits[2:0]** is 4.

ChannelFormat[1:0] - Channel Input Format. This value determines internal signal routing, and should be set according to the modulation format and input signal wiring. The following table describes each mode. See Section 4.2.1, for a description of input signal wiring.

Channel Format[1:0]	*INPUT CHANNEL FORMAT*
00	8-PSK, 16, 64, 256-QAM with internal rotation and soft metric (LLR) computation. Input data is one in-phase (I) and one quadrature (Q) component per transfer. Maximum symbol rate is 90 Msym/sec.
01	BPSK/QPSK Modulation with internal rotation and soft metric computation. Input data is one in-phase (I) and one quadrature (Q) component per transfer. Maximum symbol rate is 90 Msym/sec.
10	QPSK Modulation at Channel Rates > 50 Msym/sec, with internal symbol rotation. Input data is two in-phase (I) and two quadrature (Q) components per transfer. Maximum symbol rate is 180 MSym/sec.
11	Other modulations with external rotation and soft metric computation. Input data is one to four soft metrics per transfer. Maximum input rate is 360 Mbit/sec (where bit is a soft metric).

QMode[1:0] - Quantization Mode for soft input data. When set to "11," input data is assumed to be in signed 2's complement notation (mid-tread). When set to "01," data is assumed to be mid-riser sign/ magnitude notation. When set to "00," data is assumed to be mid-riser unsigned. When set to "10," data is assumed to be in "half 2's complement" mid-riser notation. The confidence mapping for each mode is shown below with four bit quantization

QMode[1:0]	*INPUT DATA TYPE*	*HARD DECISION 0 CONFIDENCE RANGE*			*NO CONFIDENCE*	*HARD DECISION 1 CONFIDENCE RANGE*		
		Max	*. . .*	*Min*		*Min*	*. . .*	*Max*
00	Unsigned	0000	. . .	0111	N/A	1000	. . .	1111
01	Sign/Magnitude	0111	. . .	0000	N/A	1000	. . .	1111
10	Half 2's Compl	1000	. . .	1111	N/A	0000	. . .	0111
11	2's Complement	1000	. . .	1111	0000	0001	. . .	0111

LLRMantissa[7:0] - LLR Normalization Mantissa Term. When internal LLR (soft metric) computation is used with 8-PSK, 16,64,256-QAM, a normalization term is required that depends on the distance between received constellation points, relative to the input scaling. This normalization term is expressed in floating point notation, with **LLRMantissa** being the mantissa term (M), and **LLRExponent** the exponent term (E). The radix (or base) for the floating point number is two. Therefore, the LLR Normalization value used is shown here

$$LLRNormalization = \left(1 + \frac{LLRMantissa}{256}\right) \times 2^{LLRExponent - 8}$$

The equation for the LLR Normalization is described in Section 4.2.4 *LLR Normalization*.

LLRExponent[3:0] - LLR Normalization Exponent Term.

DModRowRotate - Setting this bit enables symbol interleaving after each row in the TPC block. See Section 3.5 for a description of when to enable this option. Decode rotates left.

DModPlaneRotate - Setting this bit enables symbol interleaving after each plane in the TPC block. See Section 3.5 for a description of when to enable this option. The decoder frame size, not including sync marks, must be a multiple of four when **DModRowRotate** is set.

9.7.4 DECODER SYNCHRONIZATION CONTROL (DECSYNC0-3)

Read/Write
Reset Value (hex) = 00 00 00 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x6c	0xd8	**FBitThresh[1:0]**		**FSyncOn[5:0]**					
0x6d	0xda	**FBitThresh[2]**	**FSyncOff[6:0]**						
0x6e	0xdc	**FSyncInv**	*res*	**FSyncTime[5:0]**					
0x6f	0xde	*res*		**PSyncOn[5:0]**					

FBitThresh[2:0] - Frame synchronization mark bit threshold. When searching for a frame synchronization mark, the decoder declares a bit stream a valid synchronization mark if it has **FBitThresh[2:0]** or less bits different from the value in **DFSyncMark[31:0]**. The first sync mark detected must have **FBitThresh** -1 or less bits different than **DFSyncMark[31:0]** when **FBitThresh[2:0]** is not equal to zero.

FSyncOn[5:0] - Frame Synchronization Achieved Threshold. When the sync mark count equals **FSyncOn[5:0]**, the **FSynced** bit is set, and data is input to the decoder following the next inverted sync mark. If packet synchronization is disabled, the **SyncAcq** interrupt is also set.

FSyncOff[6:0] -Loss of Frame Synchronization Threshold. When the missed sync mark count equals **FSyncOff[6:0]**, then the **SyncLoss** interrupt is issued, the **FSynced** bit is reset, and the device begins resynchronization. When **SyncOff[6:0]** is set to zero, the automatic resync feature is disabled.

FSyncInv - Detect Inverted Frame Synchronization. If the data stream may be inverted by the channel, setting this bit will cause the frame synchronization logic to detect the inverted synchronization and invert the bit stream. For example, with QPSK systems, this will generally allow synchronization to occur in about one half the average time. When cleared, the frame synchronization logic will only lock on a non-inverted bit stream.

FSyncTime[5:0] - Synchronization Time. The amount of time, in terms of periods of length **DFSyncPer[15:0]** bits, in which the synchronizer must achieve synchronization. If **FSyncTime[5:0]** x **DFSyncPer[15:0]** bits are examined without achieving synchronization, the current phase rotation is incremented by one, and synchronization is attempted with the new rotation. A value of zero disables phase rotation and assertion of the **ROTATE** signal. **FSyncTime[5:0]** must be set greater than **FSyncOn[5:0]** for the device to acquire frame synchronization.

PSyncOn[5:0] - Packet Synchronization Achieved Threshold. When **PSyncOn[5:0]** sync marks are identified in the proper location, the **PSynced** bit and the **SyncAcq** interrupt bit are set, and the data is ready to be output from the device.

9.8 DATA INPUT/OUTPUT CONFIGURATION

9.8.1 DATA FLOW CONFIGURATION (EDATACON0-7, DDATACON0-7)

Read/Write
Reset Value (hex) = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x30	0x60	**EFlowMode[2:0]**			*res*	**ETransfers[27:24]**			
0x31	0x62	**ETransfers[23:16]**							
0x32	0x64	**ETransfers[15:8]**							
0x33	0x66	**ETransfers[7:0]**							
0x34	0x68	**UClkAdj Polarity**	**UClkAdj Mode**	*res*		**UTransfers[27:24]**			
0x35	0x6a	**UTransfers[23:16]**							
0x36	0x6c	**UTransfers[15:8]**							
0x37	0x6e	**UTransfers[7:0]**							
0x70	0xe0	**DFlowMode[2:0]**			*res*	**CTransfers[27:24]**			
0x71	0xe2	**CTransfers[23:16]**							
0x72	0xe4	**CTransfers[15:8]**							
0x73	0xe6	**CTransfers[7:0]**							
0x74	0xe8	**DClkAdj Polarity**	**DClkAdj Mode**	*res*		**DTransfers[27:24]**			
0x75	0xea	**DTransfers[23:16]**							
0x76	0xec	**DTransfers[15:8]**							
0x77	0xee	**DTransfers[7:0]**							

ETransfers[27:0] - Encoded Transfers per Flow Control Frame. When encoder flow control is enabled, this register determines the number of encoded data transfers that should occur for each flow control frame. The device will guarantee that one flow control frame contains exactly **ETransfers[27:0]** encoded transfers, and **UTransfers[27:0]** unencoded transfers.

UTransfers[27:0] - Unencoded Transfers per Flow Control Frame. When encoder flow control is enabled, this register determines the number of unencoded data transfers that should occur for each flow control frame. The device will guarantee that one flow control frame contains exactly **ETransfers[27:0]** encoded transfers, and **UTransfers[27:0]** unencoded transfers.

CTransfers[27:0] - Channel Transfers per Flow Control Frame. When decoder flow control is enabled, this register determines the number of channel data transfers that should occur for each flow control frame. The device will guarantee that one flow control frame contains exactly **CTransfers[27:0]** channel transfers, and **DTransfers[27:0]** decoded transfers.

DTransfers[27:0] - Decoded Transfers per Flow Control Frame. When decoder flow control is enabled, this register determines the number of decoded data transfers that should occur for each flow control frame. The device will guarantee that one flow control frame contains exactly **ETransfers[27:0]** encoded transfers, and **DTransfers[27:0]** decoded transfers.

xFlowMode - Flow Control Mode. Data flow through the device can be controlled to create a constant delay pipe. This greatly simplifies system integration, as the device handles all clock domain crossing and buffering. The following table describes each mode. See Section 5.0 for a full description of flow control.

xFlowMode [2:0]	***DESCRIPTION***
0	Flow Control Disabled. All transfers in and out of the device are handshaked with **RDY** and **ACPT**. The decoder has only minimal buffering to load, decode, and unload three different blocks. All buffering in encoder is disabled. See Section 5.2.
1	Internal Buffering Enabled. All transfers in and out of the device are handshaked with **RDY** and **ACPT**. The decoder has variable sized input and output buffers. The encoder has a variable size input FIFO. See Section 5.3.
2	Flow Control via Handshaking, Channel/Encoded Interface Master. The data interface handshake is regulated by the device to generate a constant data flow, depending on the rate of handshaking at the channel interface. The data must be a continuous stream. See Section 5.4.
3	Flow Control via Handshaking, Unencoded Interface Master. This mode is valid for the encoder only. The encoded data interface handshake is regulated by the device to generate a constant data flow, depending on the rate of handshaking at the unencoded interface. The data must be a continuous stream. See Section 5.4.
4	Flow Control via Frequency Synthesis with Intermittent Data Stream. All data transfers in and out of the device occur with one transfer per rising clock edge. An external VCO is used to generate the data clock. The device drives the VCO to create the appropriate clock frequency. See Section 5.5.
5	Flow Control via Frequency Synthesis with Continuous Data Streams. This mode is the same as mode 4, except that the data must be a continuous stream. When using **DFlowMode** = 5, the decoder can be used with variable iterations per block. See Section 5.5 and 5.6.

When modifying any data flow configuration values, the **xFlowMode** register should be first written to 0 to disable flow mode. After the configuration is complete, **xFlowMode** should be written to the appropriate value.

xClkAdjMode - Clock Adjust Mode. When set, the clock adjust output (**UCLKADJ[2:0]** or **DCLKADJ[2:0]**) generates two charge pump outputs, and a lock detect. **CLKADJ[0]** is a charge pump created by comparing clock phases, **CLKADJ[1]** is a charge pump created by comparing clock frequencies, and **CLKADJ[2]** is a phase lock detect output. When **xClkAdjMode** is cleared, the three most significant bits of the current value of the synthesizer up/down counter (the clock phase error) are output on **CLKADJ[2:0]**. This is a signed 2's complement number representing the phase error between the two clocks.

xClkAdjPolarity - Clock Adjust Polarity. When set, the **CLKADJ[1:0]** bits are normal polarity (as specified in Section 6.1). When cleared, the **CLKADJ[1:0]** bits are inverted. Note that this bit is only valid when the corresponding **xClkAdjMode** mode is set.

9.8.2 BUFFER CONFIGURATION (EBUFCON, DBUFCON0-1)

Read/Write
Reset Value (hex) = 00 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x1e	0x3c	**EBufferSize[7:0]**							
0x5d	0xba	*res*			**DumpThresh[4:0]**				
0x5e	0xbc	**DBufferSize[7:0]**							

EBufferSize[7:0] - Encoder Logical Buffer Size. When **EFlowMode[2:0]** is set to a non-zero value, an encoder input buffer is enabled to smooth the burst nature of the TPC encoder. Note that **EBufferSize[7:0]** does not set a maximum size on the encoder buffer, but instead it determines how many bytes the buffer should delay the datapath in order to guarantee a smooth data flow at both the unencoded and encoded data interfaces. **EBufferSize[7:0]** should be set according to the following equation:

$$\textbf{EBufferSize} = \left\lceil [n_x \times n_y \times (n_z - k_z) + n_x \times (n_y - k_y) + n_x - k_x]\frac{cr}{32} \right\rceil + 1$$

Where the code is $(n_x,k_x)(n_y,k_y)(n_z,k_z)$ and *cr* is computed as shown in Section 5.1. For 2D codes, n_z and k_z are zero. ky or kz should include the hyper axis parity bits.

DBufferSize[7:0] - Decoder Buffer Size. When **DFlowMode[2:0]** is set to 2 or 5, this register determines the size of the decoder variable iteration buffer, in steps of 128 bits. Setting this value to a larger number allows the decoder to execute more iterations per block; setting this value to a smaller number decreases latency. When **DFlowMode[2:0]** is set to 1 or 4, **DBufferSize[7:0]** determines the size of two identical decoder input and output FIFOs, in steps of 128 bits.

The maximum value for **DBufferSize[7:0]** is:

$$\left.\begin{array}{c} \textbf{DBufferSize} \le \left\lfloor 255 - \dfrac{n-k}{128} \right\rfloor \\ \text{and} \\ \textbf{DBufferSize} \le \left\lfloor \dfrac{n}{16} \right\rfloor \end{array}\right\} \textbf{DFlowMode} = 2 \text{ or } 5$$

$$\left.\begin{array}{c} \textbf{DBufferSize} \le \left\lfloor 127 - \dfrac{n-k}{256} \right\rfloor \\ \text{and} \\ \textbf{DBufferSize} \le \left\lfloor \dfrac{n}{16} \right\rfloor \end{array}\right\} \textbf{DFlowMode} = 1 \text{ or } 4$$

When **DFlowMode[20]** is set to 2 or 5, and **DumpThresh[4:0]** is non-zero, the minimum value for **DBufferSize[7:0]** is:

$$\textbf{DBufferSize} \ge \left\lceil \frac{n}{128} \right\rceil$$

If this equation cannot be met, **DumpThresh[4:0]** must be set to 0. A value of zero is illegal. When **DumpThresh[4:0]** is set to 0, the minimum value for **DBufferSize[7:0]** is 1.

DumpThresh[4:0] - Decoder Stop Iterating Threshold. When the decoder does not complete the iterating process before the buffer fills, a dump is issued. The decoder stops after the next iteration, and outputs the current data. The value of **DumpThresh[4:0]** depends on the code configuration as described in Section 5.6.

For block sizes smaller than 512 bits, the automatic dump feature must be disabled by setting **DumpThresh[4:0]** to 0.

9.8.3 SIGNAL POLARITY (POLARITY)

Read/Write
Reset Value (hex) = ff

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x38	0x70	**ERDY Polarity**	**EACPT Polarity**	**URDY Polarity**	**UACPT Polarity**	**CRDY Polarity**	**CACPT Polarity**	**DRDY Polarity**	**DACPT Polarity**

xPolarity - Handshake Signal Polarity. When set, the respective handshake signal is active high. When cleared, the respective handshake signal is active low.

9.9 CONTROL AND STATUS

9.9.1 CONTROL (ECONTROL, DCONTROL)

Read/Write
Reset Value (hex) = 00 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x39	0x72	*res*		**Reset MapPtr**	**EStart**	*res*			**EReset**
0x79	0xf2	*res*			**DStart**	**DNo Block**	*res*	**Resync**	**DReset**

Note: All bits in these registers are control bits. Writing a one to each bit causes the specified action to occur. Writing a zero to any bit has no effect.

ResetMapPtr - Reset Symbol Map Table Pointer Control Bit. Writing a one to this bit causes the next write to **ESymMap[7:0]** to be written to address 0x0 of the symbol map. This bit is automatically cleared. Writing a zero has no effect.

EStart - Start Encoder. Setting this bit indicates to the encoder that all configuration registers have been set for the new code type. The encoder will immediately begin loading data through the unencoded data interface. When using Rapid Code Reconfiguration, this bit must be set after each block code type is written into the device. See Section 9.2 *Encoder Rapid Code Reconfiguration* for more details. This bit is automatically cleared.

DStart - Start Decoder. Setting this bit indicates to the decoder that all configuration registers have been set for the new code type. The decoder will immediately begin loading data through the channel interface. When using Rapid Code Reconfiguration, this bit must be set after each block code type is written into the device. See Section 9.3 *Decoder Rapid Code Reconfiguration* for more details. This bit is automatically cleared.

DNoBlock - No block input when using Rapid Code Reconfiguration. If no data block is to be input during the next decoding cycle, this bit must be set to cause the decoder to continue processing (decoding and unloading) the previously loaded blocks. This bit is automatically cleared.

Resync - Resynchronize command. Setting this bit to a one resets all counts in the synchronizer, clears the **FSynced** and **PSynced** bits, and initiates a search for synchronization. All data in the device is cleared. This bit is automatically cleared.

EReset - Encoder Soft Reset. When written to a 1, all data in the encode path is cleared. No register contents are affected. This bit remains set until **EStart** is written to a 1. After a hard reset (**RESETN**) or **EReset** the encoder does not accept data until the **EStart** bit is set. If encoder rapid code reconfiguration is disabled, this bit must be set before modifying any encoder configuration registers.

DReset - Decoder Soft Reset. When written to a 1, all data in the decode path is cleared. No register contents are affected. This bit remains set until **DStart** is written to a 1. After a hard reset (**RESETN**) or **DReset** the decoder does not accept data until the **DStart** bit is set. If decoder rapid code reconfiguration is disabled, this bit must be set before modifying any decoder configuration registers.

9.9.2 STATUS (STATUS)

Read Only
Reset Value (hex) = 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x3b	0x76	*res*	**UClkLock**	**DClkLock**	**PSynced**	**FSynced**	**Rotation[2:0]**		

xClkLock - Clock locked when using frequency synthesis. Set when the frequency synthesizer has locked to the programmed clock frequency. This bit is only valid when **xFlowEnable** = 1, and **xHandshake** = 0.

PSynced - Packet Synchronized. Set after the synchronizer achieves packet synchronization. When set, data is ready to be output from the device. This bit is reset after a **Resync**.

FSynced - Frame Synchronized. Set after the synchronizer achieves frame synchronization. When set, data is passed to the TPC decoder. This bit is reset after a **Resync**, or if **SyncOff[5:0]** sync marks are missed.

Rotation[2:0] - Current phase rotation. When the device is synchronizing to the incoming symbol stream, this value will increment for each possible phase in the constellation. After synchronization is detected, this value will remain stable, indicating the current phase rotation. Note that when **RotateEnable** is cleared, this value will always return 0.

9.9.3 GENERAL OUTPUT (OUTPUT)

Read/Write
Reset Value (hex) = 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x3a	0x74	*res*	**GOut0Config[2:0]**			*res*	**GOut1Config[2:0]**		

GOutxConfig[2:0] - General output control. These registers control the state of the general output signals (**GOUT[1:0]**). The general outputs can be driven to a 1 or 0, or any one of six signals can be muxed to either of the general output signals. This can be used for system debugging or control of external device. The following table shows the options for each **GOUT** signal.

GOutx Config[2:0]	*SELECTED OUTPUT SIGNAL*
0	Drive **GOUT** to 0 (low).
1	Drive **GOUT** to 1 (high).
2	Drive value of **FSynced** status bit.
3	Drive value of **PSynced** status bit.
4	Differentially drive decoder load complete event. The **GOUT** will change state each time the decoder completes loading one block of data.
5	Differentially drive decode complete event. The **GOUT** will change state each time the decoder completes decoding one block of data.
6	Differentially drive next block ready for encoder when using Rapid Code Reconfiguration. The **GOUT** will change state each time the device is ready to accept a block at the encoder input. This will occur after the later of the microprocessor writing **EStart** or the **EProcComp** interrupt from the previous block.
7	Differentially drive next block ready for decoder when using Rapid Code Reconfiguration. The **GOUT** will change state each time the device is ready to accept a block at the decoder input. This will occur after the later of the microprocessor writing **DStart** or the **DProcComp** interrupt from the previous block.

9.9.4 CORRECTION COUNT (DCORRCNT0-1)

Read/Write
Reset Value (hex) = 00 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x7a	0xf4	**CorrAvg[2:0]**			*res*	**Corrections[11:8]**			
0x7b	0xf6	**Corrections[7:0]**							

CorrAvg[2:0] - Blocks to Average. This value is log base 2 of the number of blocks to average when computing the average corrections per block. A value of zero disables corrections averaging, and the value of **Corrections[11:0]** reflects the correction count for the most recent decoded block.

Corrections[11:0] - Average Corrections per block. The total number of bits corrected by the TPC decoder averaged over the 2**CorrAvg[2:0]** blocks. This value is generated by low pass filtering the corrections count per block (equivalent to a running average). After each block decoded, the value in the **Corrections[11:0]** register is subtracted from the correction count. The result of this subtraction is right shifted by **CorrAvg[2:0]**, and then added to the **Corrections[11:0]** register. The result is that the value of **Corrections[11:0]** is updated every block, and is approximately equal to the average number of corrections over the last 2**CorrAvg[2:0]** blocks. Writes to these bits have no effect. This register is reset by **DReset**.

9.9.5 INTERRUPT (INTERRUPT)

Read Only
Reset Value (hex) = 00

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x3c	0x78	*res*		**EProc Comp**	**DProc Comp**	**CRCLoss**	**SyncLoss**	**SyncAcq**	**DecComp**

Note: The interrupt status registers are updated regardless of the setting of the respective interrupt mask bit. All bits are cleared by reading this register.

xProcComp - Process Complete when using Rapid Code Reconfiguration. After each block processing is complete, this interrupt is set indicating that the code type for the next code can now be written to the respective registers.

CRCLoss - CRC Loss of Synchronization Detected. This interrupt is set when the number of consecutive CRC failures equals the **CRCSyncOff[7:0]** value. Simultaneously, the **FSynced** and **PSynced** status bits are set to zero, and a resync is initiated.

SyncLoss - Loss of Synchronization Detected. This interrupt is set when the number of consecutive missed synchronization marks equals the **SyncOff[7:0]** value. Simultaneously, the **FSynced** and **PSynced** status bits are set to zero, and a resync is initiated.

SyncAcq - Synchronization Acquired. This interrupt is set when frame and packet (if enabled) synchronization is acquired by the device.

DecComp - Decode Complete. This interrupt is set for each TPC block that is decoded. Note that if the iteration buffer is enabled, there will be a delay of multiple blocks between the completion of decoding of a block and the output of the block.

9.9.6 INTERRUPT MASK (INTMASK)

Read/Write
Reset Value (hex) = ff

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x3d	0x7a	*res*		**EProc CompM**	**DProc CompM**	**CRCLossM**	**SyncLossM**	**SyncAcqM**	**DecCompM**

xProcCompM - xProcComp Interrupt Mask. When cleared, the **UPINTN** signal will be asserted if the **xLoadComp** interrupt bit is set. When set, the **xLoadComp** interrupt bit will not cause assertion of the **UPINTN** signal.

CRCLossM - CRCLoss Interrupt Mask. When cleared, the **UPINTN** signal will be asserted if the **CRCLoss** interrupt bit is set. When set, the **CRCLoss** interrupt bit will not cause assertion of the **UPINTN** signal.

SyncLossM - SyncLoss Interrupt Mask. When cleared, the **UPINTN** signal will be asserted if the **SyncLoss** interrupt bit is set. When set, the **SyncLoss** interrupt bit will not cause assertion of the **UPINTN** signal.

SyncAcqM - SyncAcq Interrupt Mask. When cleared, the **UPINTN** signal will be asserted if the **SyncAcq** interrupt bit is set. When set, the **SynAcq** interrupt bit will not cause assertion of the **UPINTN** signal.

DecCompM - DecComp Interrupt Mask. When cleared, the **UPINTN** signal will be asserted if the **DecComp** interrupt bit is set. When set, the **DecComp** interrupt bit will not cause assertion of the **UPINTN** signal.

9.10 MISCELLANEOUS

9.10.1 VERSION

Read Only
Reset Value (hex) = 41

MUXMODE =

0	*1*	*bit7*	*bit6*	*bit5*	*bit4*	*bit3*	*bit2*	*bit1*	*bit0*
0x3f	0x7e	**Version[7:0]**							

Version[7:0] - Version number of the device.

10.0 SIGNAL DESCRIPTIONS

This section contains descriptions for all the pins. Each signal has a type code associated with it. The type codes are described in the following table. Unused input signals must be tied to their inactive state and not left floating.

TYPE CODE	*DESCRIPTION*
I	Input only pin
O	Output only pin
I/O	Input/Output pin

10.1 SYSTEM CONTROL AND MISCELLANEOUS

SIGNAL	*TYPE*	*DESCRIPTION*
RESETN	I	Power on Reset. Asynchronous. Active low reset signal. Must be active for 8 clocks of the lowest frequency clock.
TRISTATEN	I	When low, all outputs are tristated. Tie high for normal operation.
EPCLK	I	Encoder Processing Clock. This clock is used for all TPC encoding logic. The encoder operates at a rate of 4 channel bits per clock. Therefore, the TPC encoder frequency must be greater than or equal to 1/4 the channel bit rate. The maximum frequency for **EPCLK** is 90 MHz.
DPCLK	I	Decoder Processing Clock. This clock is used for all TPC decoding logic. The decoder data rate is dependent on the code selection and the number of iterations. The maximum frequency for **DPCLK** is 180 MHz.
TESTI[6:0]	I	Test mode input signals. Tie all signals low for normal operation.
TESTO[2:0]	O	Test mode output signals. Leave unconnected for normal operation
GOUT[1:0]	O	General output signals.

10.2 MICROPROCESSOR INTERFACE

SIGNAL	TYPE	DESCRIPTION
MDATA[7:0]	I/O	Processor Data. Data and address (**MUXMODE** = 1) for all microprocessor reads and writes of registers within the device transfers across this bus.
MA[6:0]	I	Processor Address Bus. Used to address internal registers within the device.
MCSN	I	Processor Chip Select. Selects the device as the source or destination of the current microprocessor bus cycle. **MCSN** needs to be active for a minimum of two **DPCLK** cycles to start a microprocessor access.
MWRN_RWN	I	Processor Read/Write Select. When PROCMODE is deasserted, this is the active low write enable signal. When PROCMODE is asserted, this is the active low read/write select signal.
MRDY_DTACKN	O	Processor Ready/Data Transfer Acknowledge. When PROCMODE is deasserted, this is an active high ready signal. When PROCMODE is asserted, this signal is an active low data transfer acknowledge. This signal is tristated when **MCSN** is inactive.
MRDN_DSN	I	Processor Read Enable/Data Strobe. When PROCMODE is deasserted, this is the active low read enable signal. When PROCMODE is asserted, this is the active low data strobe signal.
MINTN	O	Active Low Processor Interrupt. Open drain output with an internal pull-up.
MALE	I	Processor Address Latch Enable. When PROCMODE is not asserted and MUXMODE is asserted, this signal is the active high address latch enable. Otherwise, this pin is not used and must be tied low.
PROCMODE	I	Processor Mode. Intel® mode when deasserted, Motorola® mode when asserted.
MUXMODE	I	Muxed Processor Mode. Deassert for non-muxed address and data bus mode, assert for muxed address and data bus mode.

10.3 UNENCODED DATA INTERFACE

SIGNAL	TYPE	DESCRIPTION
UCLK	I	Unencoded Data Clock. The maximum frequency is 90 MHz, and it must be less than or equal to the **EPCLK** frequency.
UDATA[7:0]	I	Unencoded Input Data. Synchronous to **UCLK**. Data is handshaked into the device on the rising edge of **UCLK** when both **URDY** and **UACPT** are asserted.
URDY	I	Unencoded Data Ready. Synchronous to **UCLK**. Indicates to the device that data is ready to be handshaked across the **UDATA[7:0]** bus. The polarity of this signal is selected in the **URDYPolarity** register, Section 9.8.3
UACPT	O	Unencoded Data Accept. Synchronous to **UCLK**. Indicates that the device can accept data on the **UDATA[7:0]** bus. The polarity of this signal is selected in the **UACPTPolarity** register, Section 9.8.3
UCLKADJ[2:0]	O	Unencoded Clock Adjustment. Synchronous to **ECLK**. These signals are used to adjust the clock frequency of **UCLK**. Refer to Section 6.1 *Frequency Synthesizer Configuration*.

10.4 ENCODED DATA INTERFACE

SIGNAL	TYPE	DESCRIPTION
ECLK	I	Encoded Symbol Clock. The maximum frequency is 90 MHz, and it must be less than or equal to the **EPCLK** frequency.
EDATA[7:0]	O	Encoded Output Symbols. Synchronous to **ECLK**. Data is handshaked out of the device on the rising edge of **ECLK** when both **ERDY** and **EACPT** are asserted.
ERDY	O	Encoded Data Ready. Synchronous to **ECLK**. Indicates that the device has data ready to be handshaked across **EDATA[7:0]** bus. The polarity of this signal is selected in the **ERDYPolarity** register, Section 9.8.3
EACPT	I	Encoded Data Accept. Synchronous to **ECLK**. Indicates to the device that the data on the **EDATA[7:0]** bus can be accepted by an external device. The polarity of this signal is selected in the **EACPTPolarity** register, Section 9.8.3
ESTART	O	Encoded Data Frame Start. Synchronous to **ECLK**. Indicates that the data on the **EDATA[7:0]** bus is the first symbol of a new frame. This signal is handshaked with the data.
EEND	O	Encoded Data Frame End. Synchronous to **ECLK**. Indicates that the data on the **EDATA[7:0]** bus is the last symbol of the frame. This signal is handshaked with the data.

10.5 CHANNEL INTERFACE

SIGNAL	TYPE	DESCRIPTION
CCLK	I	Channel Symbol Clock. The maximum frequency is 90 MHz, and it must be less than or equal to the **DPCLK** frequency divided by 2.
CDATA[15:0]	I	Channel Symbol Data. Synchronous to **CCLK**. When using internal LLR generation, **CDATA[15:8]** is the in-phase (I) component of the demodulated input symbol, and **CDATA[7:0]** is the quadrature (Q) component of the same input symbol. When using external LLR generation, **CDATA[15:0]** is divided into n 4 bit busses, where n depends on the bit input rate. See Section 4.2.1. Data is handshaked into the device on the rising edge of **CCLK** when **CRDY** and **CACPT** are asserted.
CRDY	I	Channel Symbol Ready. Synchronous to **CCLK**. Indicates to the device that data is ready to be handshaked across the **CDATA[7:0]** bus. The polarity of this signal is selected in the **CRDYPolarity** register, Section 9.8.3
CACPT	O	Channel Symbol Accept. Synchronous to **CCLK**. Indicates that the device can accept data on the **CDATA[7:0]** bus. The polarity of this signal is selected in the **CACPTPolarity** register, Section 9.8.3
CLSB	I	Channel Symbol Least Significant Bits. Synchronous to **CCLK**. This signal is only used when **ChannelFormat** = 0b11, and the modulation bits per symbol is greater than 4. This signal must be asserted when transferring the least significant LLRs of the symbol. This signal is handshaked with the data.
CSTART	I	Channel Synchronization. Synchronous to **CCLK**. When **DFSyncEnable** is cleared, external logic must detect the beginning of a TPC block, and assert **CSTART** with the first transfer of each block. This signal is handshaked with the data.
ROTATE	O	Rotate Phase. Synchronous to **DPCLK_I**. When using external logic for phase ambiguity resolution, this signal is used to rotate the phase and attempt synchronization on the new rotation. The synchronizer will attempt to detect synchronization on each input data stream. After the synchronization block has reached **FSyncTime** without achieving synchronization, **ROTATE** is inverted, indicating that external logic should rotate the phase. **ROTATE** is driven low by **RESETN** assertion.

10.6 DECODED DATA INTERFACE

SIGNAL	TYPE	DESCRIPTION
DCLK	I	Decoded Data Clock. The maximum frequency is 90 MHz, and it must be less than or equal to the **DPCLK** frequency divided by 2.
DDATA[7:0]	O	Output Data Bus. Synchronous to **DCLK**. Data is handshaked out of the device on the rising edge of **DCLK** when **DRDY** and **DACPT** are asserted.
DRDY	O	Decoded Data Ready. Synchronous to **DCLK**. Indicates that the device has data ready to be handshaked across **DDATA[7:0]** bus. The polarity of this signal is selected in the **DRDYPolarity** register, Section 9.8.3
DACPT	I	Decoded Data Accept. Synchronous to **DCLK**. Indicates to the device that the data on the **DDATA[7:0]** bus can be accepted by an external device. The polarity of this signal is selected in the **DACPTPolarity** register, Section 9.8.3
DSTART	O	Decoded Packet Start. Synchronous to **DCLK**. Indicates that the data on **DDATA[7:0]** is the first byte in a packet. This signal is handshaked with the data.
DEND	O	Decoded Packet End. Synchronous to **DCLK**. Indicates that the data on **DDATA[7:0]** is the last byte in a packet. This signal is handshaked with the data.
DERR	O	Decoded Packet Error. Synchronous to **DCLK**. Asserted with **DERR** indicating that the CRC detected errors in the packet that was output. This signal is handshaked with the data.
DCLKADJ[2:0]	O	Decoded Clock Adjustment. Synchronous to **CCLK**. These signals are used to adjust the clock frequency of **DCLK**. Refer to Section 6.1 *Frequency Synthesizer Configuration*.

11.0 PINOUT

Table 6: Pinout - Pin Number Order

PIN	SIGNAL	PIN	SIGNAL	PIN	SIGNAL	PIN	SIGNAL
1	GNDIO	53	MDATA[7]	105	GND	157	CDATA[12]
2	VDDIO	54	GND	106	VDD	158	CDATA[13]
3	GND	55	VDD	107	ESTART	159	GND
4	VDD	56	MRDY_DTACKN	108	EEND	160	VDD
5	RESETN	57	MINTN	109	EDATA[0]	161	CDATA[14]
6	TRISTATEN	58	GOUT[0]	110	EDATA[1]	162	CDATA[15]
7	TESTI[0]	59	GOUT[1]	111	GND	163	CRDY
8	TESTI[1]	60	GND	112	VDD	164	CSTART
9	TESTI[2]	61	VDD	113	GNDIO	165	DACPT
10	GND	62	GNDIO	114	VDDIO	166	GND
11	VDD	63	VDDIO	115	EDATA[2]	167	VDD
12	TESTI[3]	64	UACPT	116	EDATA[3]	168	DCLK
13	TESTI[4]	65	GNDIO	117	EDATA[4]	169	GNDIO
14	TESTI[5]	66	VDDIO	118	GND	170	VDDIO
15	MUXMODE	67	UCLKADJ[0]	119	VDD	171	DPCLK
16	PROCMODE	68	UCLKADJ[1]	120	EDATA[5]	172	GND
17	GND	69	GND	121	EDATA[6]	173	VDD
18	VDD	70	VDD	122	EDATA[7]	174	GND
19	GNDIO	71	UCLKADJ[2]	123	GNDIO	175	VDD
20	VDDIO	72	TESTO[0]	124	VDDIO	176	TESTI[6]
21	MA[0]	73	GND	125	GND	177	GNDIO
22	MA[1]	74	VDD	126	VDD	178	VDDIO
23	GND	75	TESTO[1]	127	ROTATE	179	GND
24	VDD	76	TESTO[2]	128	CACPT	180	VDD
25	MA[2]	77	GNDIO	129	GNDIO	181	DCLKADJ[0]
26	MA[3]	78	VDDIO	130	VDDIO	182	DCLKADJ[1]
27	MA[4]	79	GND	131	GND	183	DCLKADJ[2]
28	MA[5]	80	VDD	132	VDD	184	GNDIO
29	MA[6]	81	UCLK	133	CCLK	185	VDDIO
30	GND	82	URDY	134	CDATA[0]	186	GND
31	VDD	83	UDATA[0]	135	CDATA[1]	187	VDD
32	MWRN_RWN	84	UDATA[1]	136	CDATA[2]	188	DRDY
33	MRDN_DSN	85	UDATA[2]	137	GND	189	DSTART
34	MCSN	86	GND	138	VDD	190	DEND
35	MALE	87	VDD	139	CDATA[3]	191	DERR
36	GND	88	UDATA[3]	140	CDATA[4]	192	GND
37	VDD	89	UDATA[4]	141	CDATA[5]	193	VDD
38	GNDIO	90	UDATA[5]	142	GND	194	DDATA[0]
39	VDDIO	91	UDATA[6]	143	VDD	195	DDATA[1]
40	MDATA[0]	92	GND	144	CDATA[6]	196	GNDIO
41	MDATA[1]	93	VDD	145	CDATA[7]	197	VDDIO
42	MDATA[2]	94	UDATA[7]	146	GND	198	GND
43	GND	95	EACPT	147	VDD	199	VDD
44	VDD	96	ECLK	148	CDATA[8]	200	DDATA[2]
45	MDATA[3]	97	GNDIO	149	CDATA[9]	201	DDATA[3]
46	MDATA[4]	98	VDDIO	150	CLSB	202	DDATA[4]
47	MDATA[5]	99	GND	151	GNDIO	203	GNDIO
48	GND	100	VDD	152	VDDIO	204	VDD
49	VDD	101	EPCLK	153	GND	205	DDATA[5]
50	GNDIO	102	GNDIO	154	VDD	206	DDATA[6]
51	VDDIO	103	VDDIO	155	CDATA[10]	207	DDATA[7]
52	MDATA[6]	104	ERDY	156	CDATA[11]	208	GND

Figure 34: Pinout

Top View

Notes:

1) *YYWWD = Date Code*
2) *LLLLL = Lot Number*
3) *Unused input signals must be tied to their inactive state and not left floating.*

12.0 DC ELECTRICAL SPECIFICATIONS

Information in this section represents design goals. While every effort will be made to meet these goals, values presented should be considered targets until characterization is complete. Please consult with Comtech AHA Corporation for the most up-to-date values.

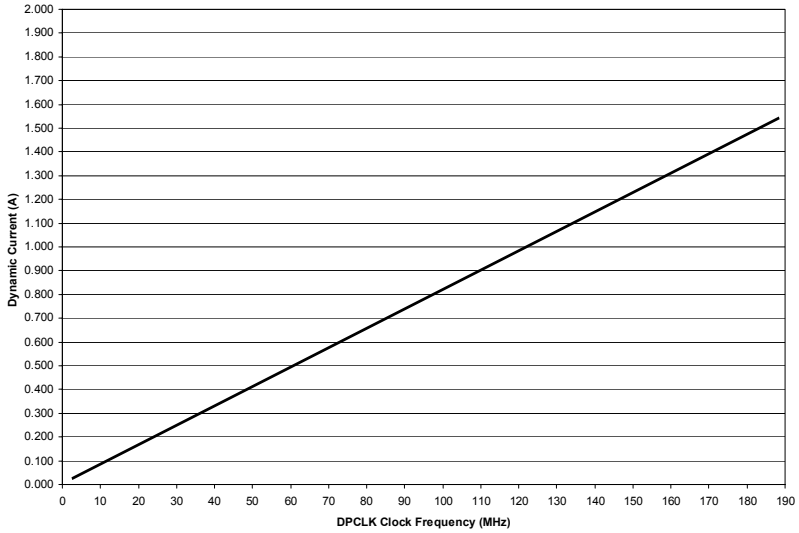
12.1 OPERATING CONDITIONS

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTES
Vdd	Core Supply voltage (VDD)	1.7	1.9	V	5
Vdd	Core Supply voltage (VDD)	1.9	2.1	V	2,5
Vddio	Input/Output Supply Voltage (VDDIO)	3.1	3.5	V	5
Idd	Core Supply current (static)		5	mA	
Idd	Core Supply current (active)		1500	mA	6
Iddio	Input/Output Supply current (static)		1	mA	
Iddio	Input/Output Supply current (active)		50	mA	6
Ta	Ambient temperature (commercial)	0	70	°C	3
Vil	Input low voltage		0.8	V	
Vih	Input high voltage	2.0		V	
Iin	Input leakage current	-5	5	uA	4
Vol	Output low voltage (Iol=-2mA)		0.4	V	
Voh	Output high voltage (Ioh=2mA)	2.4		V	
Iol	Output low current	2		mA	
Ioh	Output high current	2		mA	
Ioz	Output leakage current during tristate	-5	5	uA	
Cin	Input capacitance		10	pF	
Cio	Input/Output capacitance		10	pF	
Cout	Output load capacitance		50	pF	1

Notes:

1) Timings referenced to this load.
2) Vdd MIN = 1.9V when DPCLK > 160 MHz.
3) Requires 200 LFPM forced air when DPCLK > 160 MHz.
4) Measured at OV and Vddio nominal.
5) During power-on sequencing, Vdd must reach 1.7V concurrent or before Vddio reaches 1.7V.
6) At maximum DPCLK and EPCLK frequencies, nominal Vdd and Vddio, and Full Duplex operation.

Figure 35: Current (Idd) Vs. DPCLK



12.2 ABSOLUTE MAXIMUM STRESS RATINGS

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTES
Tstg	Storage temperature	-50	150	°C	
V1.8 Vsupply	Supply voltage on Vdd	-0.5	2.3	V	
V3.3 Vsupply	Supply voltage on Vddio	-0.5	4.3	V	
Vin (3.3V)	Input voltage	-0.5	4.1	V	

12.3 TEST CONDITIONS

PARAMETER	VALUE
AC timing reference	1.4 V

* *The timing diagrams for these signals assume a capacitive load of 50 pF.*

13.0 AC ELECTRICAL SPECIFICATIONS

13.1 DPCLK CLOCK TIMING

Figure 36: DPCLK Clock Timing

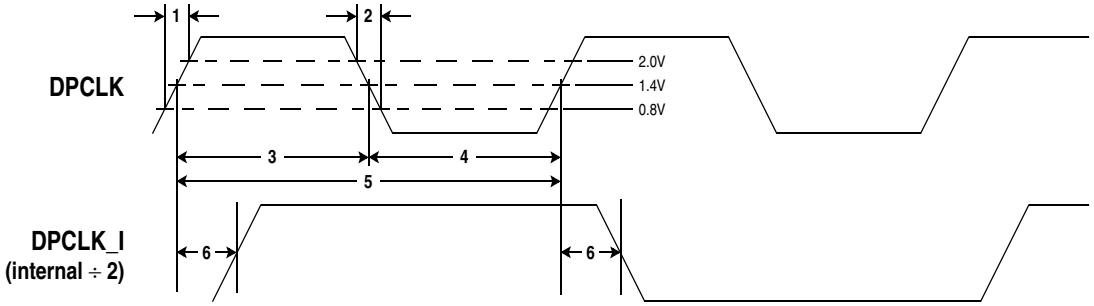


Table 7: DPCLK Clock Timings

NUMBER	PARAMETER	MIN	MAX	UNITS	NOTES
1	**DPCLK** rise time		1	ns	
2	**DPCLK** fall time		1	ns	
6	**DPCLK** to **DPCLK_I** delay		3	ns	1
3, 4	**DPCLK** high time, low time	2.5		ns	2
5	**DPCLK** period (T_{cp})	6.25		ns	2
5	**DPCLK** frequency		160	MHz	2
3, 4	**DPCLK** high time, low time	2.22		ns	3
5	**DPCLK** period (T_{cp})	5.56		ns	3
5	**DPCLK** frequency		180	MHz	3

Notes:

1) *DPCLK_I is an internally generated divide by 2 clock. It is specified here because it is referenced in the microprocessor interface timings.*
2) *Vdd = 1.8V nominal*
3) *Vdd = 2.0V nominal*

13.2 ECLK, UCLK, CCLK, DCLK, AND EPCLK CLOCK TIMING

Figure 37: ECLK, UCLK, CCLK, DCLK, and EPCLK Clock Timing

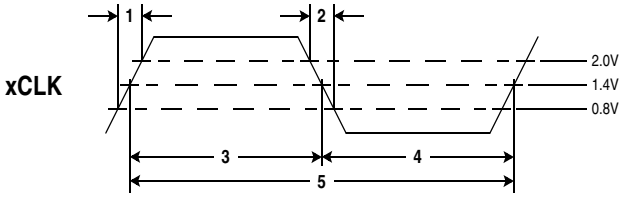


Table 8: ECLK, UCLK, CCLK, DCLK, and EPCLK Clock Timings

NUMBER	PARAMETER	MIN	MAX	UNITS	NOTES
1	**ECLK, UCLK, CCLK, DCLK, EPCLK** rise time		2	ns	
2	**ECLK, UCLK, CCLK, DCLK, EPCLK** fall time		2	ns	
3, 4	**ECLK, UCLK, CCLK, DCLK, EPCLK** high time, low time	5		ns	1
3, 4	**ECLK, UCLK, CCLK, DCLK, EPCLK** high time, low time	4.44		ns	2
5	**ECLK, UCLK, CCLK, DCLK, EPCLK** period (T_{cp})	12.5		ns	1
5	**ECLK, UCLK, CCLK, DCLK, EPCLK** frequency		80	MHz	1
5	**ECLK, UCLK, CCLK, DCLK, EPCLK** period (T_{cp})	11.11		ns	2
6	**ECLK, UCLK, CCLK, DCLK, EPCLK** frequency		90	MHz	2,3,4

Notes: *1) Vdd = 1.8V nominal*
2) Vdd = 2.0V nominal
3) ECLK and UCLK frequncy must be less than or equal to the EPCLK frequency
4) CCLK and DCLK must be less than or equal to the DPCLK frequency divided by 2

13.3 UNENCODED DATA INTERFACE

Figure 38: Unencoded Data Interface – Data Input Timing

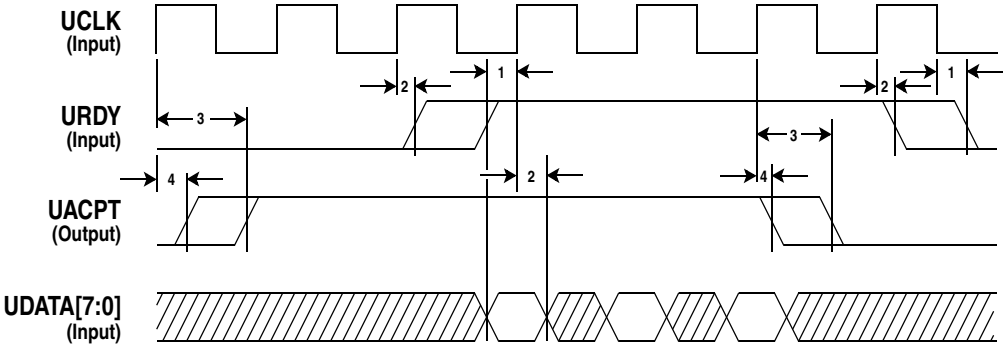


Table 9: Unencoded Data Interface – Data Input Timing

NUMBER	PARAMETER	MIN	MAX	UNITS	NOTES
1	**URDY/UDATA[7:0]** setup to **UCLK**	3		ns	
2	**URDY/UDATA[7:0]** hold from **UCLK**	1		ns	
3	**UACPT** drive delay from **UCLK**		9	ns	1
4	**UACPT** output hold from **UCLK**	2			1

Notes:
1) With 50 pF load.

13.4 ENCODED DATA INTERFACE

Figure 39: Encoded Data Interface – Data Output Timing

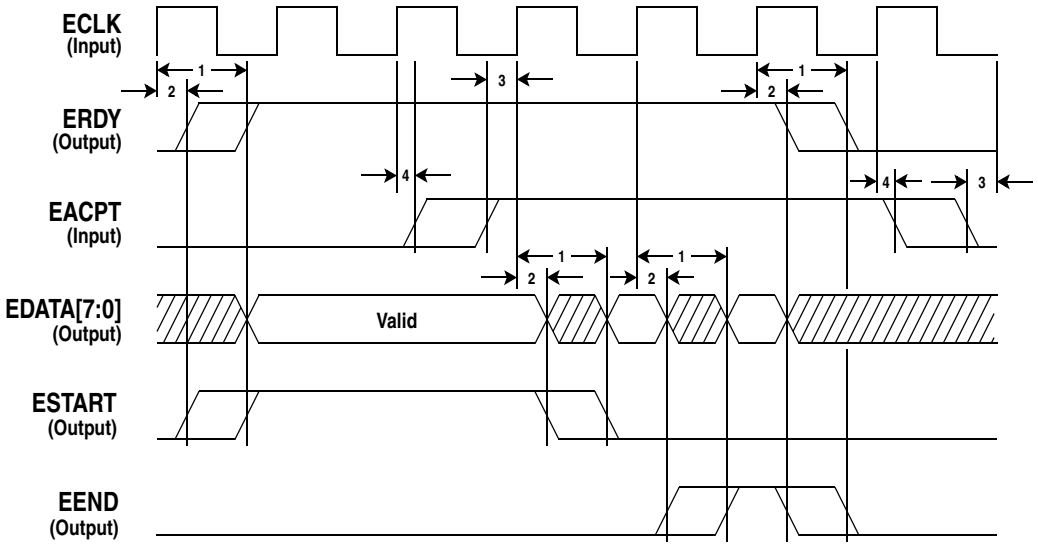


Table 10: Encoded Data Interface – Data Output Timing

NUMBER	PARAMETER	MIN	MAX	UNITS	NOTES
1	**ERDY**, **EDATA[7:0]**, **ESTART**, **EEND** drive delay from **ECLK**		8	ns	1
2	**ERDY**, **EDATA[7:0]**, **ESTART**, **EEND** output hold from **ECLK**	2		ns	1
3	**EACPT** setup to **ECLK**	3		ns	
4	**EACPT** hold from **ECLK**	1		ns	

Notes:

1) With 50 pF load.

13.5 CHANNEL INTERFACE

Figure 40: Channel Symbol Input Timing

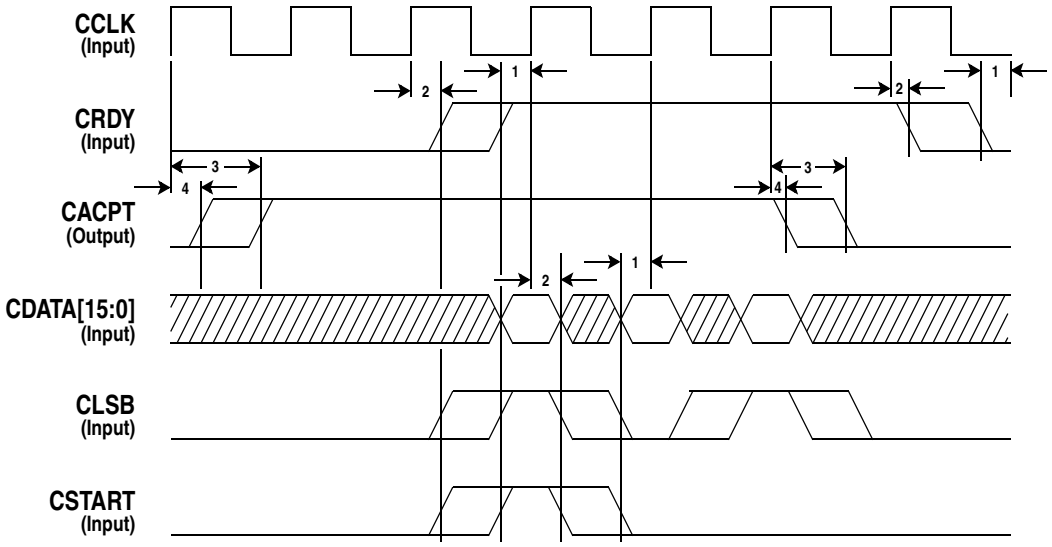


Table 11: Channel Symbol Input Timing

NUMBER	PARAMETER	MIN	MAX	UNITS	NOTES
1	**CRDY, CDATA[15:0], CSTART, CLSB** setup to **CCLK**	3		ns	
2	**CRDY, CDATA[15:0], CSTART, CLSB** hold from **CCLK**	1		ns	
3	**CACPT** drive delay from **CCLK**		8	ns	1
4	**CACPT** output hold from **CCLK**	2		ns	1

Notes:

1) With 50 pF load.

13.6 DECODED DATA INTERFACE

Figure 41: Decoded Data Interface – Data Output Timing

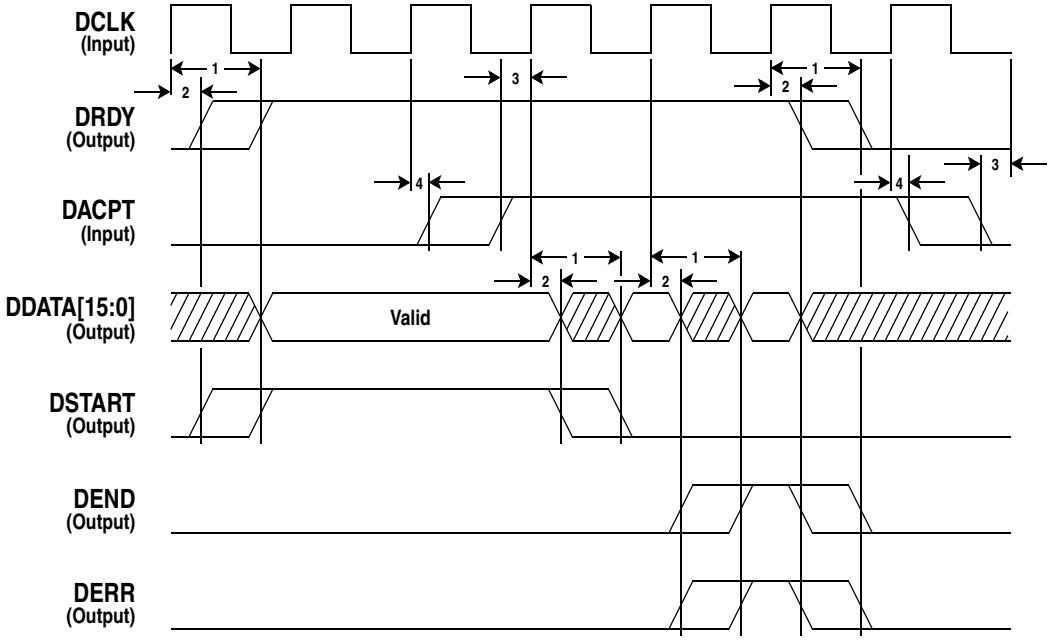


Table 12: Decoded Data Interface – Data Output Timing

NUMBER	*PARAMETER*	*MIN*	*MAX*	*UNITS*	*NOTES*
1	**DRDY, DDATA[15:0], DSTART, DEND, DERR** drive delay from **DCLK**		8	ns	1
2	**DRDY, DDATA[15:0], DSTART, DEND, DERR** output hold from **DCLK**	2		ns	1
3	**DACPT** setup to **DCLK**	3		ns	
4	**DACPT** hold from **DCLK**	1		ns	

Notes:

1) With 50 pf load.

13.7 MICROPROCESSOR INTERFACE

Figure 42: Microprocessor Interface Timing (Write); PROCMODE=0, MUXMODE=0

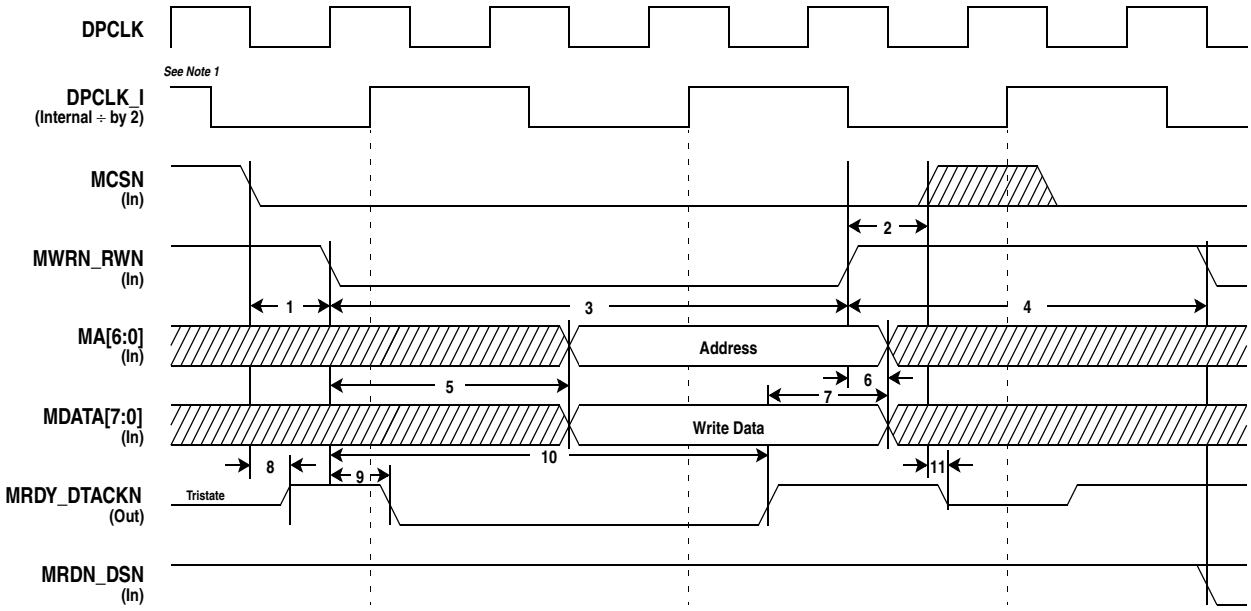


Table 13: Microprocessor Interface Timing (Write); PROCMODE=0, MUXMODE=0

NUMBER	PARAMETER	MIN	MAX	UNITS	NOTES
1	**MCSN** low to **MWRN_RWN** low	0		ns	2
2	**MWRN_RWN** high to **MCSN** high	0		ns	4
3	**MWRN_RWN** pulse width	2 Tcp+3ns			3, 5
4	**MWRN_RWN** high to next **MWRN_RWN** or **MRDN_DSN** low	1 Tcp+3ns			3
5	**MWRN_RWN** low to **MA[6:0]** and **MDATA[7:0]** valid		1 Tcp–3ns		3
6	**MA[6:0]** and **MDATA[7:0]** hold from **MWRN_RWN** high	0		ns	
7	**MA[6:0]** and **MDATA[7:0]** hold from **MRDY_DTACKN** high	0		ns	
8	**MRDY_DTACKN** driven from **MCSN** low		12	ns	
9	**MRDY_DTACKN** low from **MWRN_RWN** low		12	ns	
10	**MRDY_DTACKN** high from **MWRN_RWN** low		2 Tcp+12ns		3
11	**MRDY_DTACKN** tristate from **MCSN** high		12	ns	6

Notes:

1) *The microprocessor interface signals are internally synchronized to DPCLK_I.*
2) *Write cycle begins when both MCSN and MWRN_RWN are low at the rising edge of DPCLK_I.*
3) *Tcp = DPCLK_I clock period = 2 × DPCLK clock period.*
4) *MCSN may be held low continuously for back to back accesses.*
5) *This timing assumes a fixed number of wait states per cycle. It can be ignored if MRDY_DTACKN is used to insert wait states.*
6) *Not tested in production.*

Figure 43: Microprocessor Interface Timing (Read); PROCMODE=0, MUXMODE=0

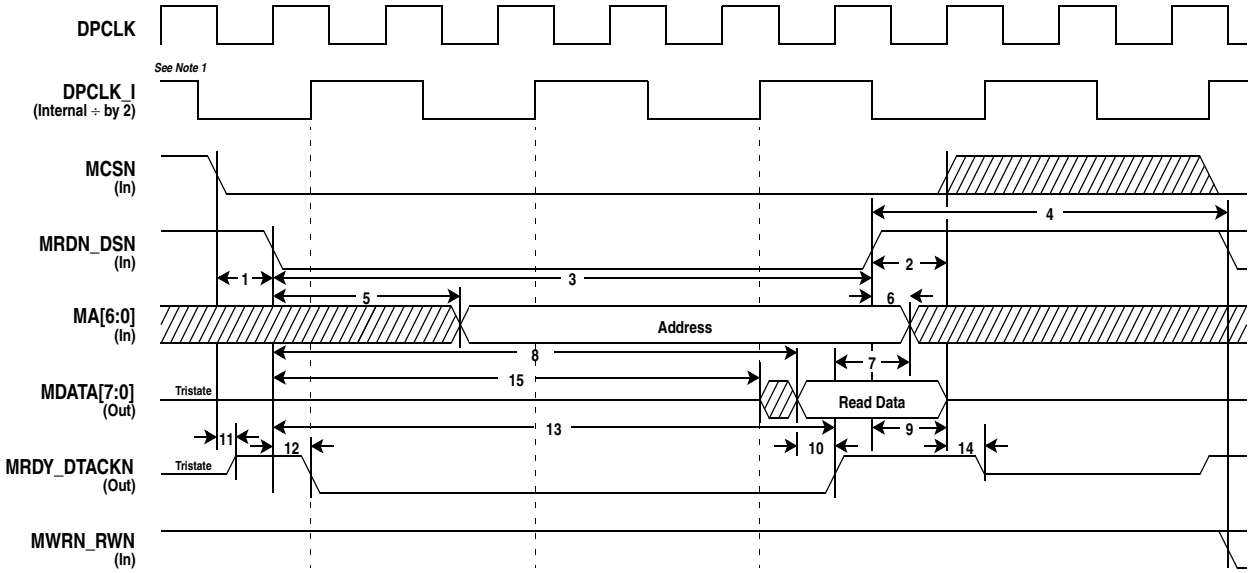


Table 14: Microprocessor Interface Timing (Read); PROCMODE=0, MUXMODE=0

NUMBER	PARAMETER	MIN	MAX	UNITS	NOTES
1	**MCSN** low to **MRDN_DSN** low	0		ns	2
2	**MRDN_DSN** high to **MCSN** high	0		ns	4
3	**MRDN_DSN** pulse width	3 Tcp+3ns			3, 5
4	**MRDN_DSN** high to next **MRDN_DSN** or **MWRN_RWN** low	1 Tcp+3ns			3
5	**MRDN_DSN** low to **MA[6:0]** valid		1 Tcp–3ns		3
6	**MA[6:0]** hold from **MRDN_DSN** high	0		ns	
7	**MA[6:0]** hold from **MRDY_DTACKN** high	0		ns	
8	**MDATA[7:0]** valid from **MRDN_DSN** low		3 Tcp+10ns		3
9	**MDATA[7:0]** tristate from **MRDN_DSN** high	2	12	ns	6
10	**MDATA[7:0]** valid to **MRDY_DTACKN** high	0		ns	
11	**MRDY_DTACKN** driven from **MCSN** low		12	ns	
12	**MRDY_DTACKN** low from **MRDN_DSN** low		12	ns	
13	**MRDY_DTACKN** high from **MRDN_DSN** low		3 Tcp+12ns		3
14	**MRDY_DTACKN** tristate from **MCSN** high		12	ns	
15	**MDATA[7:0]** driven from **MRDN_DSN** low	2 Tcp–3ns			3, 6

Notes:

1) *The microprocessor interface signals are internally synchronized to DPCLK_I.*
2) *Read cycle begins when both MCSN and MRDN_DSN are low at the rising edge of DPCLK_I.*
3) *Tcp = DPCLK_I clock period = 2 × DPCLK clock period.*
4) *MCSN may be held low continuously for back to back accesses.*
5) *This timing assumes a fixed number of wait states per cycle. It can be ignored if MRDY_DTACKN is used to insert wait states.*
6) *Not tested in production.*

Figure 44: Microprocessor Interface Timing (Write); PROCMODE=0, MUXMODE=1

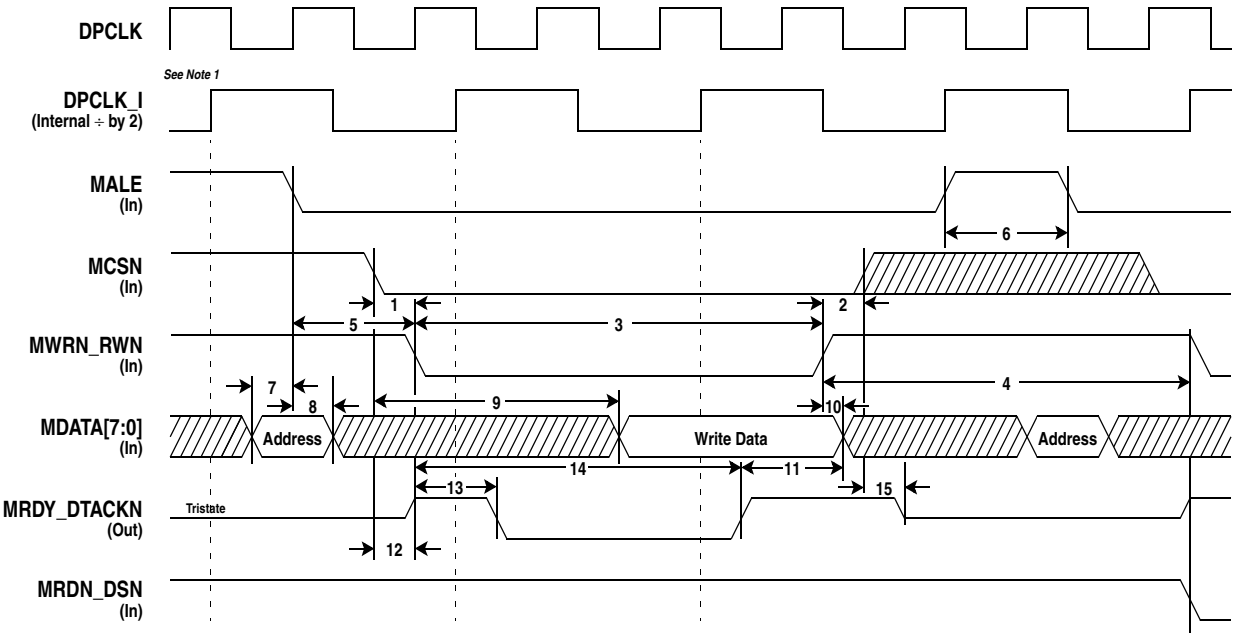


Table 15: Microprocessor Interface Timing (Write); PROCMODE=0, MUXMODE=1

NUMBER	PARAMETER	MIN	MAX	UNITS	NOTES
1	**MCSN** low to **MWRN_RWN** low	0		ns	2
2	**MWRN_RWN** high to **MCSN** high	0		ns	4
3	**MWRN_RWN** pulse width	2 Tcp+3ns			3, 5
4	**MWRN_RWN** high to next **MWRN_RWN** or **MRDN_DSN** low	1 Tcp+3ns			3
5	**MALE** low to **MWRN_RWN** low	0		ns	
6	**MALE** high pulse width	10		ns	
7	Address setup to **MALE** falling edge	7		ns	
8	Address hold from **MALE** falling edge	7		ns	
9	**MWRN_RWN** low to Write Data valid		1 Tcp–3ns		3
10	Write Data hold from **MWRN_RWN** high	0		ns	
11	Write Data hold from **MRDY_DTACKN** high	0		ns	
12	**MRDY_DTACKN** driven from **MCSN** low		12	ns	
13	**MRDY_DTACKN** low from **MWRN_RWN** low		12	ns	
14	**MRDY_DTACKN** high from **MWRN_RWN** low		2 Tcp+12ns		3
15	**MRDY_DTACKN** tristate from **MCSN** high		12	ns	6

Notes:

1) The microprocessor interface signals are internally synchronized to DPCLK_I.
2) Write cycle begins when both MCSN and MWRN_RWN are low at the rising edge of DPCLK_I.
3) Tcp = DPCLK_I clock period = 2 × DPCLK clock period.
4) MCSN may be held low continuously for back to back accesses.
5) This timing assumes a fixed number of wait states per cycle. It can be ignored if MRDY_DTACKN is used to insert wait states.
6) Not tested in production.

Figure 45: Microprocessor Interface Timing (Read); PROCMODE=0, MUXMODE=1

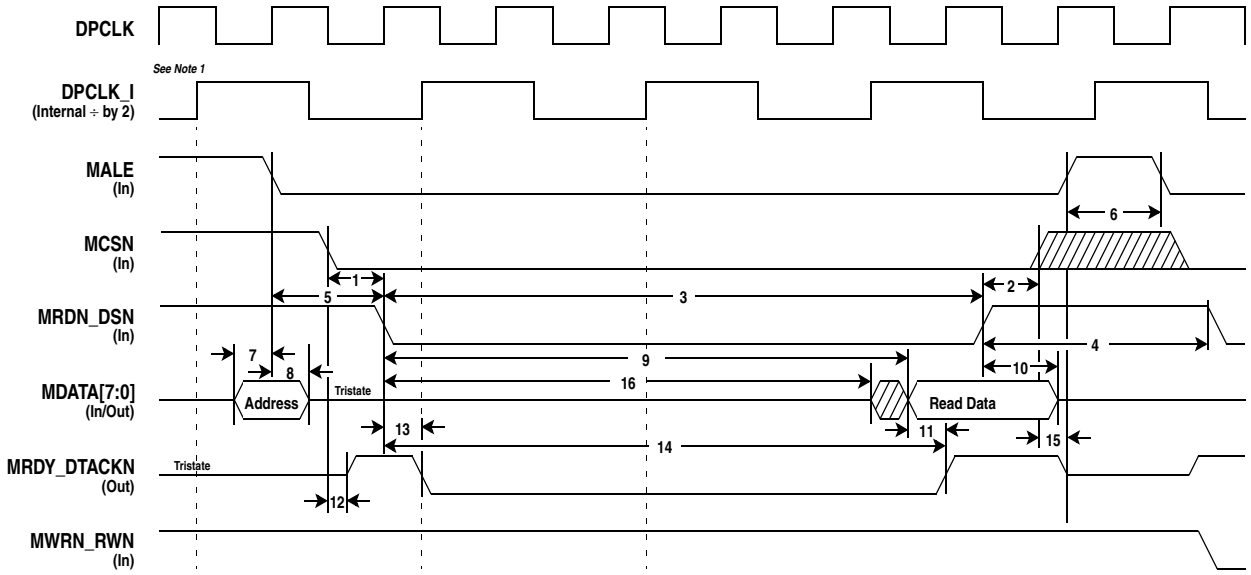


Table 16: Microprocessor Interface Timing (Read); PROCMODE=0, MUXMODE=1

NUMBER	PARAMETER	MIN	MAX	UNITS	NOTES
1	**MCSN** low to **MRDN_DSN** low	0		ns	2
2	**MRDN_DSN** high to **MCSN** high	0		ns	4
3	**MRDN_DSN** pulse width	3 Tcp+3ns			3, 5
4	**MRDN_DSN** high to next **MRDN_DSN** or **MWRN_RWN** low	1 Tcp+3ns			3
5	**MALE** low to **MRDN_DSN** low	0		ns	
6	**MALE** high pulse width	10		ns	
7	Address setup to **MALE** falling edge	7		ns	
8	Address hold from **MALE** falling edge	7		ns	
9	**MRDN_DSN** low to Read Data valid		3 Tcp+10ns		3
10	Read Data tristate from **MRDN_DSN** high	2	12	ns	6
11	Read Data valid to **MRDY_DTACKN** high	0		ns	
12	**MRDY_DTACKN** driven from **MCSN** low		12	ns	
13	**MRDY_DTACKN** low from **MRDN_DSN** low		12	ns	
14	**MRDY_DTACKN** high from **MRDN_DSN** low		3 Tcp+12ns		3
15	**MRDY_DTACKN** tristate from **MCSN** high		12	ns	
16	Read Data driven from **MRDN_DSN** low	2 Tcp–3ns			3, 6

Notes:

1) *The microprocessor interface signals are internally synchronized to DPCLK_I.*
2) *Read cycle begins when both MCSN and MRDN_DSN are low at the rising edge of DPCLK_I.*
3) *Tcp = DPCLK_I clock period = 2 × DPCLK clock period.*
4) *MCSN may be held low continuously for back to back accesses.*
5) *This timing assumes a fixed number of wait states per cycle. It can be ignored if MRDY_DTACKN is used to insert wait states.*
6) *Not tested in production.*

Figure 46: Microprocessor Interface Timing (Write); PROCMODE=1, MUXMODE=0

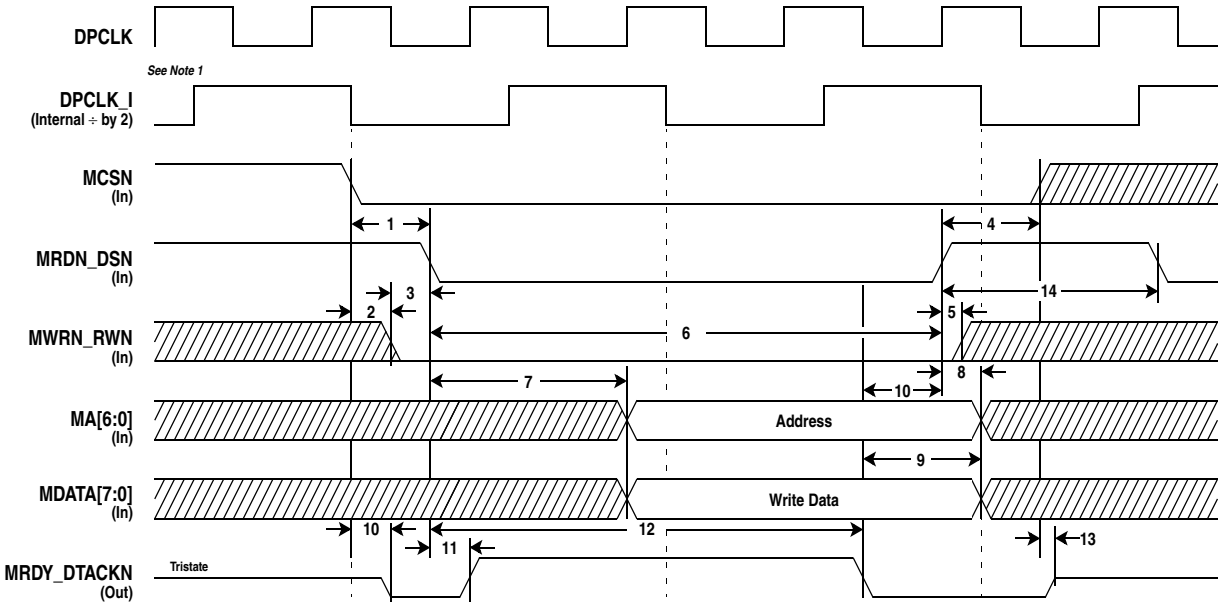


Table 17: Microprocessor Interface Timing (Write); PROCMODE=1, MUXMODE=0

NUMBER	PARAMETER	MIN	MAX	UNITS	NOTES
1	**MCSN** low to **MRDN_DSN** low	0		ns	2
2	**MCSN** low to **MWRN_RWN** low	0		ns	
3	**MWRN_RWN** setup to **MRDN_DSN** low	3		ns	
4	**MRDN_DSN** high to **MCSN** high	0		ns	4
5	**MWRN_RWN** hold from **MRDN_DSN** high	0		ns	
6	**MRDN_DSN** low pulse width	2 Tcp+3ns			3, 5
7	**MRDN_DSN** low to **MA[6:0]** and **MDATA[7:0]** valid		1 Tcp–3ns		3
8	**MA[6:0]** and **MDATA[7:0]** hold from **MRDN_DSN** high	0		ns	
9	**MA[6:0]** and **MDATA[7:0]** hold from **MRDY_DTACKN** low	0		ns	
10	**MRDY_DTACKN** driven from **MCSN** low		12	ns	
11	**MRDY_DTACKN** high from **MRDN_DSN** low		12	ns	
12	**MRDY_DTACKN** low from **MRDN_DSN** low		2 Tcp+12ns		3
13	**MRDY_DTACKN** tristate from **MCSN** high		12	ns	6
14	**MRDN_DSN** high to next **MRDN_DSN** low	1 Tcp+3ns			3

Notes:

1) *The microprocessor interface signals are internally synchronized to DPCLK_I.*
2) *Write cycle begins when both MCSN and MRDN_DSN are low at the rising edge of DPCLK_I.*
3) *Tcp = DPCLK_I clock period = 2 × DPCLK clock period.*
4) *MCSN may be held low continuously for back to back accesses.*
5) *This timing assumes a fixed number of wait states per cycle. It can be ignored if MRDY_DTACKN is used to insert wait states.*
6) *Not tested in production.*

Figure 47: Microprocessor Interface Timing (Read); PROCMODE=1, MUXMODE=0

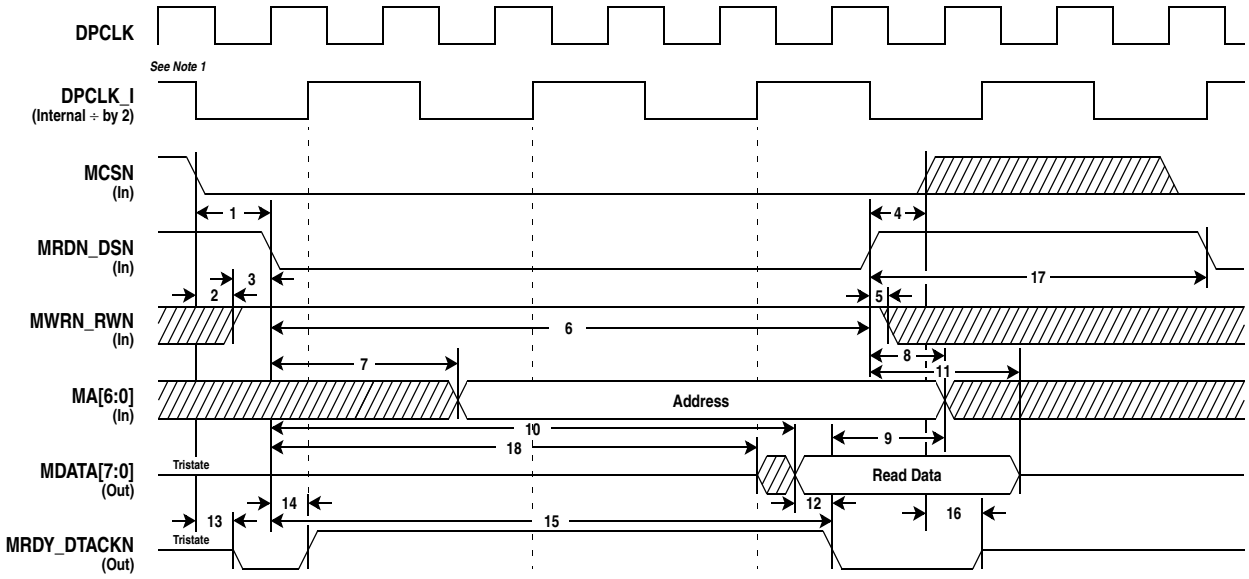


Table 18: Microprocessor Interface Timing (Read); PROCMODE=1, MUXMODE=0

NUMBER	PARAMETER	MIN	MAX	UNITS	NOTES
1	**MCSN** low to **MRDN_DSN** low	0		ns	2
2	**MCSN** low to **MWRN_RWN** high	0		ns	
3	**MWRN_RWN** high to **MRDN_DSN** low	3		ns	
4	**MRDN_DSN** high to **MCSN** high	0		ns	4
5	**MWRN_RWN** hold from **MRDN_DSN** high	0		ns	
6	**MRDN_DSN** low pulse width	3 Tcp+3ns			3, 5
7	**MRDN_DSN** low to **MA[6:0]** valid		1 Tcp–3ns		3
8	**MA[6:0]** hold from **MRDN_DSN** high	0		ns	
9	**MA[6:0]** hold from **MRDY_DTACKN** low	0		ns	
10	**MRDN_DSN** low to **MDATA[7:0]** valid		3 Tcp+10ns		3
11	**MDATA[7:0]** tristate from **MRDN_DSN** high	2	12	ns	6
12	**MDATA[7:0]** valid to **MRDY_DTACKN** high	0		ns	
13	**MRDY_DTACKN** driven from **MCSN** low		12	ns	
14	**MRDY_DTACKN** high from **MRDN_DSN** low		12	ns	
15	**MRDY_DTACKN** low from **MRDN_DSN** low		3 Tcp+12ns		3
16	**MRDY_DTACKN** tristate from **MCSN** high		12	ns	
17	**MRDN_DSN** high to next **MRDN_DSN** low	1 Tcp+3ns			3
18	**MDATA[7:0]** driven from **MRDN_DSN** low	2 Tcp–3ns			3, 6

Notes:

1) *The microprocessor interface signals are internally synchronized to DPCLK_I.*
2) *Read cycle begins when both MCSN and MRDN_DSN are low at the rising edge of DPCLK_I.*
3) *Tcp = DPCLK_I clock period = 2 × DPCLK clock period.*
4) *MCSN may be held low continuously for back to back accesses.*
5) *This timing assumes a fixed number of wait states per cycle. It can be ignored if MRDY_DTACKN is used to insert wait states.*
6) *Not tested in production.*

13.8 MISCELLANEOUS

Figure 48: Interrupt Timing

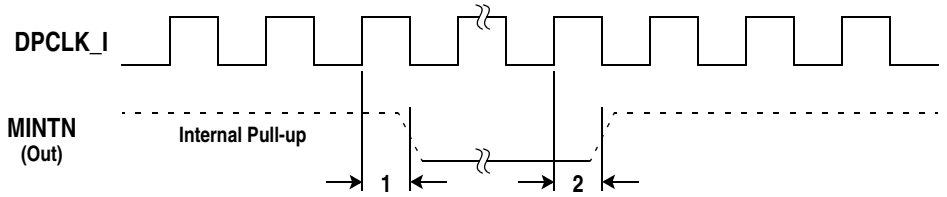


Table 19: Interrupt Timing

NUMBER	PARAMETER	MIN	MAX	UNITS	NOTES
1	**MINTN** delay time		12	ns	1
2	**MINTN** hold time	1.5		ns	2

Notes:

1) With 50 pF load.

2) It is recommended that an external pull-up is added to decrease the rise time.

Figure 49: RESETN Timing

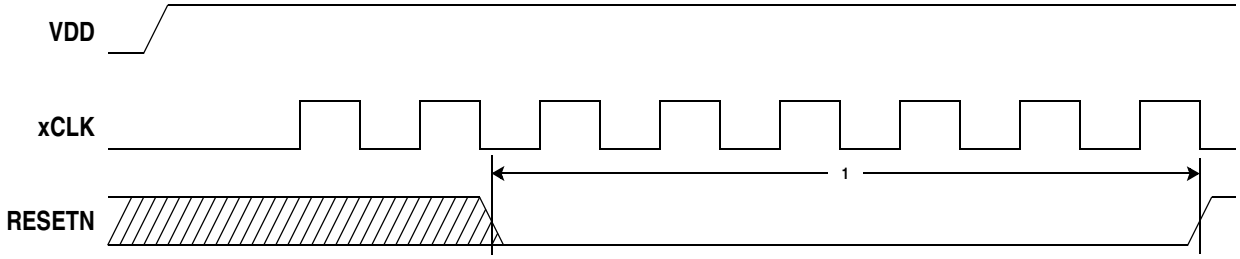


Table 20: RESETN Timing

NUMBER	PARAMETER	MIN	MAX	UNITS	NOTES
1	**RESETN** pulsewidth	8		T_{xCLK}	1

Notes:

1) T_{xCLK} is the period of the lowest frequency clock

Figure 50: GOUT, ROTATE Timing

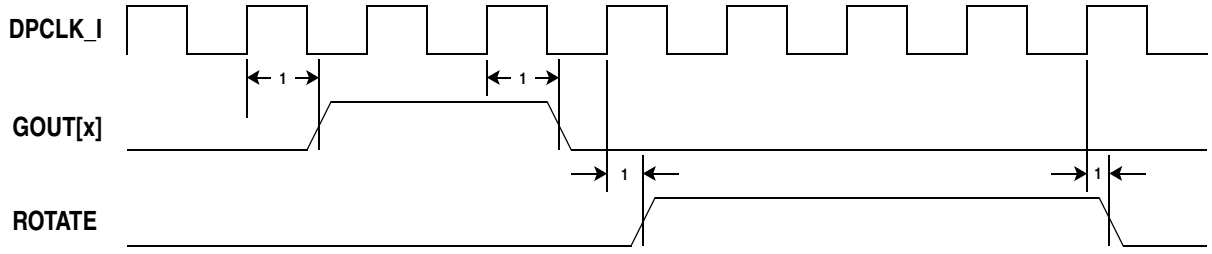


Table 21: GOUT, ROTATE Timing

NUMBER	PARAMETER	MIN	MAX	UNITS	NOTES
1	**GOUT[X]**, **ROTATE** output delay from **DPCLK_I**		10	ns	

Figure 51: Flow Control Timing (xCLKADJ Mode = 0)

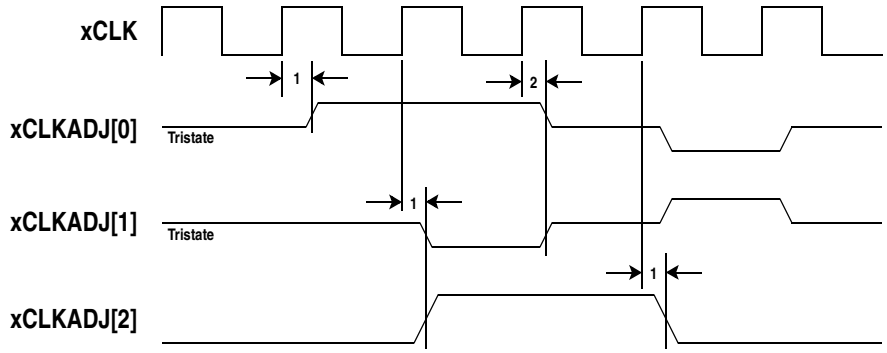


Table 22: Flow Control Timing (xCLKADJ Mode = 0)

NUMBER	PARAMETER	MIN	MAX	UNITS	NOTES
1	**xCLKADJ** drive delay from **XCLK**		15	ns	1
2	**xCLKADJ** tristate from **XCLK**	1	15	ns	2

Notes:

1) With 50 pF load.

2) Not tested in production.

Figure 52: Flow Control Timing (xCLKADJ Mode = 1)

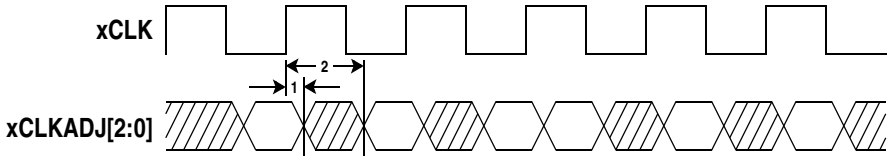


Table 23: Flow Control Timing (xCLKADJ Mode = 1)

NUMBER	PARAMETER	MIN	MAX	UNITS	NOTES
1	**xCLKADJ** output hold from **XCLK**	2		ns	1
2	**xCLKADJ** drive delay from **XCLK**		15	ns	1

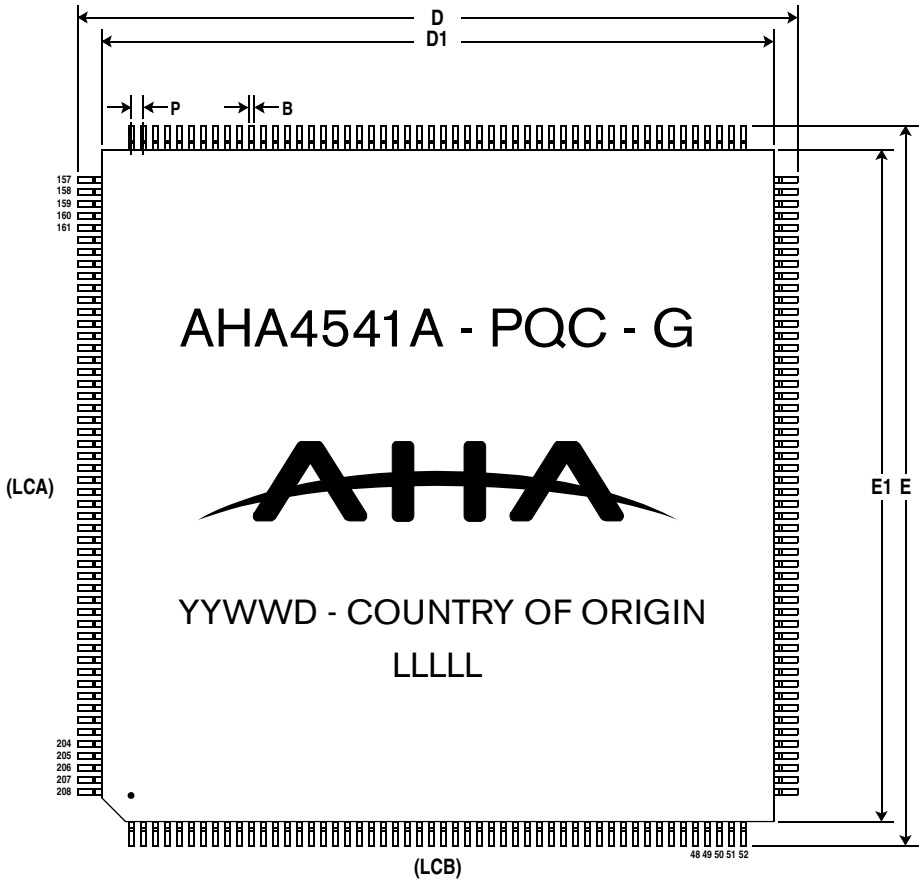
Notes:

1) With 50 pF load.

14.0 PACKAGE SPECIFICATIONS

14.1 PACKAGE DIMENSIONS

Figure 53: Package Dimensions - Top View



Note: *YYWWD = Date Code*
YY = Year
WW = Work Week
D = day
LLLLL = Lot Number

Figure 54: Package Dimensions - Cross Section View

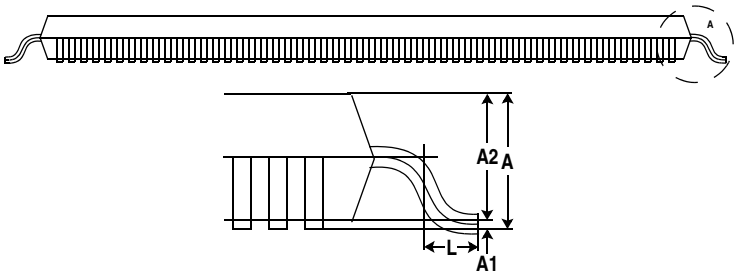


Table 24: PQ2 / MQFP (Power Quad 2 / Metric Quad Flat Pack) 28 x 28 mm Package Dimensions

(All dimensions are in mm)

SYMBOL	NUMBER OF PIN AND SPECIFICATION DIMENSION		
	208		
	SB		
	MIN	NOM	MAX
(LCA)	52 (Lead Count A)		
(LCB)	52 (Lead Count B)		
A		3.7	4.07
A1	.25	.33	0.50
A2	3.20	3.37	3.60
D		30.6	
D1		28.0	
E		30.6	
E1		28.0	
L	.46	.56	.66
P		.50	
B	.17	.22	.27

Notes: All dimensions are in millimeters

15.0 ORDERING INFORMATION

15.1 AVAILABLE PARTS

PART NUMBER	DESCRIPTION
AHA4541A-PQC	311 Mbits/sec Turbo Product Code Encoder/Decoder - Commercial Temp

15.2 PART NUMBERING

AHA	4541	A	P	Q	C	G
Manufacturer	Device Number	Revision Level	Package Material	Package Type	Temperature Specification	RoHS Compliant

Device Number:

4541

Revision Letter:

A

Package Material Codes:

P - Plastic

Package Type Codes:

Q - Quad Flat Pack

Test Specifications:

C - Commercial 0°C to +70°C
G - RoHS Compliant

16.0 ABOUT AHA

Comtech AHA Corporation (AHA) develops and markets superior integrated circuits, boards, and intellectual property core technology for communications systems architects worldwide. AHA has been setting the standard in Forward Error Correction and Lossless Data Compression technology for many years and provides flexible, cost-effective solutions for today's growing bandwidth and reliability challenges. Comtech AHA Corporation is a wholly owned subsidiary of Comtech Telecommuncations Corp. (NASDAQ: CMTL). For more information, visit www.aha.com.