Viterbi Decoder v9.1

LogiCORE IP Product Guide

Vivado Design Suite

PG027 February 4, 2021

XILINX

Table of Contents

Appendix A: Upgrading

Appendix B: Debugging

Appendix C: Additional Resources and Legal Notices

Introduction

The Viterbi Decoder is used in many Forward Error Correction (FEC) applications and in systems where data are transmitted and subject to errors before reception. The Viterbi Decoder is compatible with many common standards, such as DVB, 3GPP2, 3GPP LTE, IEEE 802.16, Hiperlan, and Intelsat IESS-308/309.

Features

- High-speed, compact Viterbi Decoder
- Fully synchronous design using a single clock
- Parameterizable constraint length from 7 to 9
- Parameterizable convolution codes
- Parameterizable traceback length
- Decoder rates from 1/2 to 1/7
- Very low latency option
- Minimal block RAM requirements; two block RAMs for a constraint length 7 decoder
- Serial architecture for small area
- Soft decision with parameterizable soft width
- Multichannel decoding
- Dual rate decoder
- Trellis mode
- Erasure for external puncturing
- BER monitor
- Normalization
- Best state option

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	UltraScale+™ Families UltraScale™ Architecture Versal™ ACAP Zynq®-7000 SoC, 7 Series
Supported User Interfaces	AXI4-Stream
Resources	Performance and Resource Utilization web page
Provided with Core	
Design Files	Encrypted RTL
Example Design	Not Provided
Test Bench	VHDL
Constraints File	Not Provided
Simulation Model	VHDL Behavioral VHDL or Verilog Structural
Supported S/W Driver	N/A
Tested Design Flows[2]	
Design Entry	Vivado® Design Suite System Generator for DSP
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Release Notes and Known Issues	Master Answer Record: 54512
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Xilinx Support web page	

Notes:

1. For a complete listing of supported devices, see the Vivado IP catalog.
2. For the supported versions of third-party tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. This document covers the following design processes:

- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, and evaluating the Vivado timing, resource and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - Port Descriptions
  - Clocking
  - Resets
  - Customizing and Generating the Core

Core Overview

This core implements a Viterbi Decoder for decoding convolutionally encoded data. For detailed information on the design, see Chapter 3, Designing with the Core.

Feature Summary

In modern communication systems, there is a requirement to transmit data and recover it, without error, in the presence of noise. This prevents having to retransmit the data, if there are errors, which would reduce the data rate in the system. One technique used is convolutional coding. A Convolutional Encoder [Ref 1] and Viterbi Decoder [Ref 3] are used together to provide the error correction. The Convolutional Encoder adds redundancy to the original data, and in the presence of noise the Viterbi Decoder uses maximum likelihood decoding to recover the data.

The Convolutional Encoder encodes the input data. A typical code rate for an encoder is 1/2, which signifies that for each input bit there are two output bits from the encoder. Similarly, for a code rate 1/3, each input bit has three output bits. Generator polynomials are used to encode each output bit from the Convolutional Encoder, thereby providing error protection for the input data. The encoder implementation consists of XOR gates and shift registers.

The Viterbi Decoder is configured to the same parameters as the encoder - code rate, constraint length, and the generator polynomials. The format of the input data to the Viterbi Decoder can be either hard or soft coding. A hard code is a binary value, whereas a soft code has several levels to reflect the strength, and hence confidence level, of the input data. This allows the Viterbi Decoder to know how strong '1' or '0' can be, which results in a better error protection. The output of the Viterbi Decoder is the original data that was input into the encoder.

The summary features for the Viterbi Decoder are as follows:

- Parameterizable decoder rates, constraint length, convolution codes and traceback lengths
- Choice of either parallel architecture for high data throughput, or serial for smaller area footprint
- Very low latency option
- Soft decision with parameterizable soft width
- Other architectural options such as multichannel decoding, dual rate decoder or trellis mode
- Erasure for external puncturing

Applications

The Viterbi Decoder is compatible with many common standards, such as DVB, 3GPP2, 3GPP LTE, IEEE 802.16, Hiperlan, and Intelsat IESS-308/309.

Licensing and Ordering

This Xilinx LogiCORE™ IP module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado® Design Suite. For full access to all core functionalities in simulation and in hardware, you must purchase a license for the core. To generate a full license, visit the product licensing web page. Evaluation licenses and hardware timeout licenses might be available for this core or subsystem. Contact your local Xilinx sales representative for information about pricing and availability.

For more information, visit the Viterbi Decoder product page.

Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

TIP: *To verify that you need a license, check the "License" column of the IP Catalog. "Included" means that a license is included with the Vivado Design Suite; "Purchase" means that you have to purchase a license to use the core.*

License Checkers

If the IP requires a license key, the key must be verified. The Vivado® design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with error. License checkpoints are enforced by the following tools:

- Vivado design tools: Vivado Synthesis
- Vivado Implementation
- write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Product Specification

The Viterbi Decoder is used in many Forward Error Correction (FEC) applications and in systems where data are transmitted and subject to errors before reception.

Standards

The Viterbi Decoder core adheres to the AMBA® AXI4-Stream standard [Ref 5].

Performance

For full details about performance and resource utilization, visit the Performance and Resource Utilization web page.

Latency

The latency of the core depends on the traceback length and the constraint length. If the reduced latency option is selected, then the latency of the core is approximately halved and the latency is only 2 times the traceback length. The latencies given in the following sections are a count of the number of symbol inputs between `s_axis_data_tdata` and the decoded data result on the output `m_axis_data_tdata`. The actual latency depends on the parameters selected for the core and the true value of the latency for a given set of parameters can be found through simulation. The `tvalid` signal indicates when there is valid data on the output of the core.

Without Reduced Latency

For a parallel core, the latency is of the order

$$\text{Latency} \cong 4*\text{traceback\_length} + \text{constraint\_length} + \text{output\_rate}$$

For a serial core, the latency is of the order shown. Note that the latency is in terms of valid inputs in the serial case.

$$\text{Latency} \cong 4*\text{traceback\_length} + \text{constraint\_length}$$

With Reduced Latency

Reduced latency is only available for the parallel core. The latency is given by

$$\text{Latency} \cong 2*\text{traceback\_length} + \text{constraint\_length} + \text{output\_rate}$$

Multichannel Latency

The multichannel core always uses the standard latency option. The latency in the multichannel case is given by

$$\text{Latency} \cong 4*\text{traceback\_length}*\text{channel\_count} + (\text{constraint\_length}*\text{channel\_count}) + \text{output\_rate}$$

This corresponds to the reduced latency single-channel case above with channel count set to 1.

Trellis Mode Latency

The latency of the Trellis Mode Decoder is as the equations above, but reduced by the `output_rate` as the branch metric costing unit is not present in the core.

BER Performance

BER performance curves were generated using a hardware-in-the-loop test framework, consisting of the Xilinx Convolution Encoder, an AWGN channel model, the Viterbi Decoder and logic to implement BPSK modulation, soft data mapping and data comparison and collection. Figure 2-1 shows the basic system data flow.

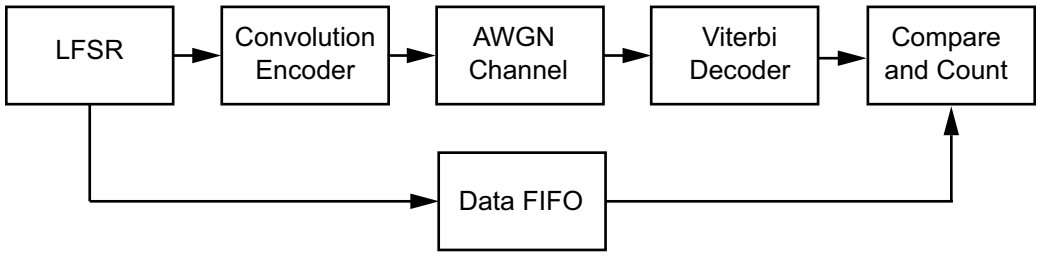

Figure 2-1: **BER Performance Test System Data Flow Diagram**

Figure 2-2 shows the BER performance of the core for constraint lengths 7 and 9 with soft width 4 and rate 1/2.

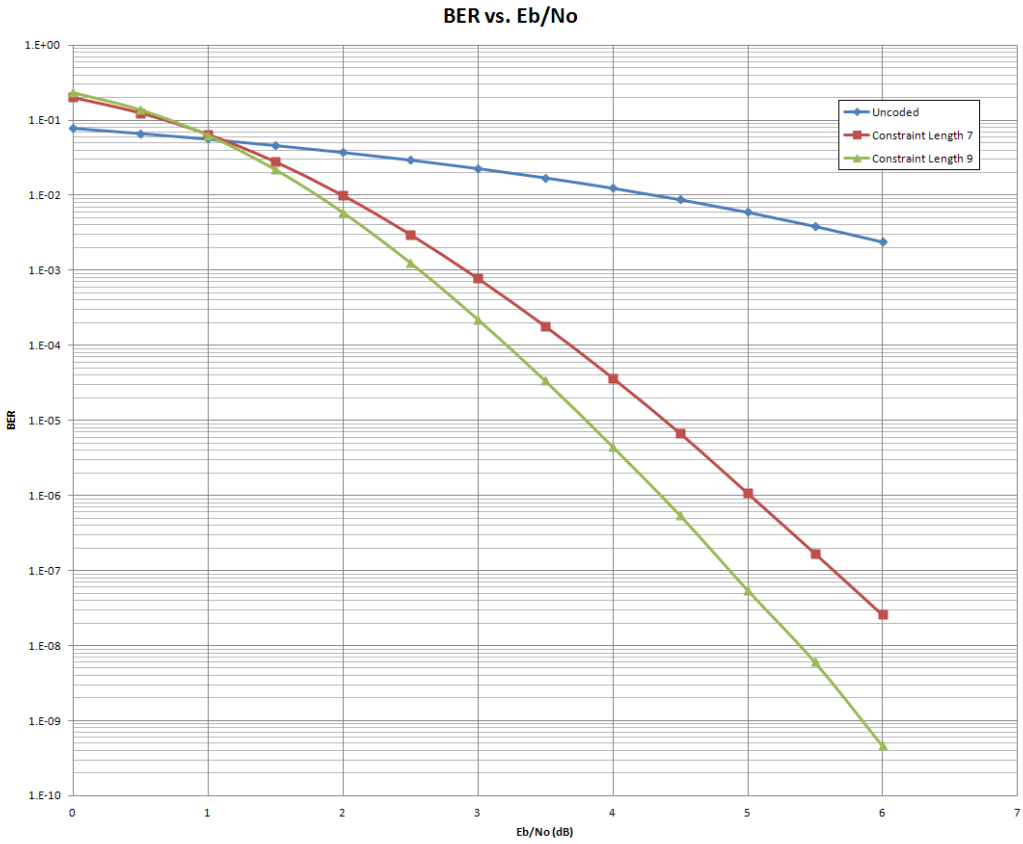

Figure 2-2: **BER Performance for Rate 1/2 System**

Resource Utilization

For full details about performance and resource utilization, visit the Performance and Resource Utilization web page.

Port Descriptions

A representative symbol of the Viterbi Decoder, with the signal names, is shown in Figure 2-3 and Figure 2-4 and described in Table 2-1. Some of the pins are optional. These should be selected only if they are genuinely required, as their inclusion might result in an increase in the core size. Timing diagrams for the signals are shown in the Functional Description in Chapter 3.

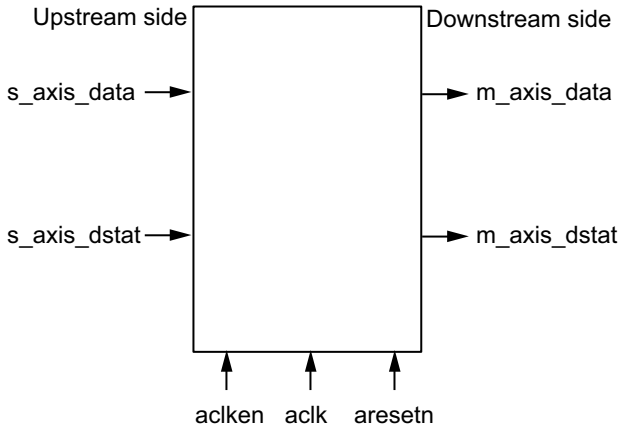

Figure 2-3: **Core AXI Channels**

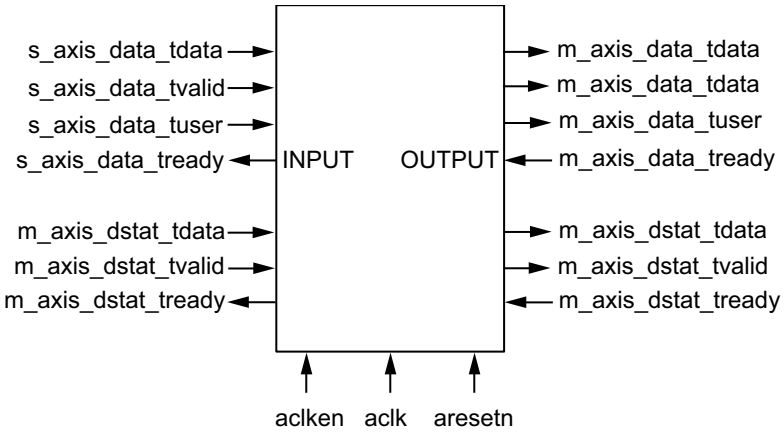

Figure 2-4: **Core Schematic Symbol**

Table 2-1: **Signal Descriptions**

Signal	I/O	Description
aclk	I	Rising edge clock
aclken	I	Active-High clock enable (optional)
aresetn	I	Active-Low synchronous clear (overrides aclken)
s_axis_data_tdata	I	Input data
s_axis_data_tvalid	I	tvalid for S_AXIS_DATA channel. See AXI4-Stream Protocol.
s_axis_data_tready	O	tready for S_AXIS_DATA. Indicates that the core is ready to accept data.
m_axis_data_tdata	O	tdata for the output data channel, decoded output data.
m_axis_data_tvalid	O	tvalid for M_AXIS_DATA channel.
m_axis_data_tready	I	tready for M_AXIS_DATA channel. Do not enable optional tready pins or tie port High if downstream slave is always able to accept data from M_AXIS_DATA.
s_axis_dstat_tdata	I	Input status data; for the Viterbi Decoder this is the BER count.

Table 2-1: **Signal Descriptions *(Cont'd)***

Signal	I/O	Description
s_axis_dstat_tvalid	I	tvalid for S_AXIS_DSTAT channel. See AXI4-Stream Protocol.
s_axis_dstat_tready	O	tready for S_AXIS_DSTAT. Indicates that the core is ready to accept data. Always High, except after a reset if there is not a tready on the output.
m_axis_dstat_tdata	O	tdata for the output DSTAT channel. Outputs the number of errors on the channel.
m_axis_dstat_tvalid	O	tvalid for M_AXIS_DSTAT channel.
m_axis_dstat_tready	I	tready for M_AXIS_DSTAT channel. Do not enable or tie High if downstream slave is always able to accept data from M_AXIS_DSTAT.

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

Functional Description

This core implements a Viterbi Decoder for decoding convolutionally encoded data. For details of the encoding process, see the *Convolutional Encoder Product Guide* (PG026) [Ref 1]. This is available in Xilinx Vivado® Design Suite. The decoder core consists of two basic architectures: a fully parallel implementation which gives fast data throughput at the expense of silicon area and a serial implementation which occupies a small area but requires a fixed number of clock cycles per decoded result.

Viterbi decoding decodes the data originally input to the Convolutional Encoder by finding an optimal path through all the possible states of the encoder. For a constraint length 7, there are 64 states and for a constraint length 9 there are 256. The basic decoder core consists of three main blocks as shown in Figure 3-1.

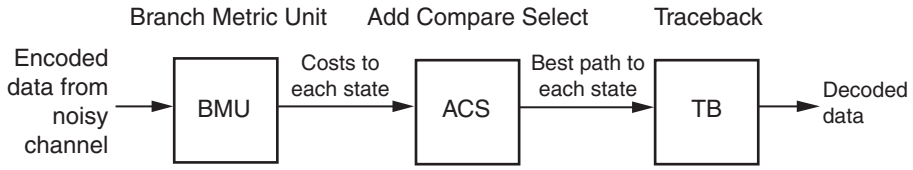

Figure 3-1: **Viterbi Decoder Block Diagram**

Costing

The first block is the branch-metric-unit (BMU). This module costs the incoming data. For the fully parallel decoder, the incoming data can be hard coded with bit width 1 or soft coded with a parameterizable bit width which can be set to any value from 3 to 5. Hard-coded data is decoded using the Hamming method of decoding, whereas soft data is decoded using an Euclidean metric. In hard coding, the demodulator makes a firm or hard decision on whether a one or zero is transmitted and provides no other information to the decoder on how reliable the decision is. For soft decoding, the demodulator provides the decoder with some side information together with the decision. The extra information provides the decoder with a measure of confidence for the decision. Soft-coded data gives a significantly better BER performance compared with hard-coded data. Soft decision offers

approximately a 3 dB increase in coding gain over hard-decision decoding. Hard coding is available only for the Standard parallel or Multichannel Viterbi. Erasure (external puncturing) is not available with hard coding.

There are two available data formats for soft coding: soft signed magnitude and offset binary. See Table 3-1 for the data formats for the case of soft width 3.

Table 3-1: **Data Format for Soft Width 3**

	Signed Magnitude	Offset-Binary
Strongest 1	111	111
	110	110
	101	101
Weakest 1	100	100
Weakest 0	000	011
	001	010
	010	001
Strongest 0	011	000

Decoding

The second block in the decoder is the add-compare-select (ACS) unit. This block selects the optimal path to each state in the Viterbi trellis. Figure 3-2 shows one stage in the Viterbi trellis for a constraint length 3 decoder. The ACS block uses the convolutional codes to extract the correct cost from the BMU for each branch in the trellis.

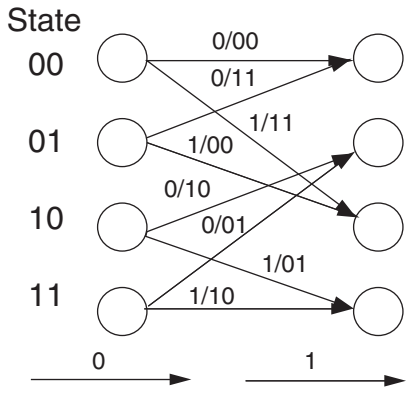

Figure 3-2: **State Transitions for Constraint Length 3**

The BER performance of the Viterbi algorithm varies greatly with different convolutional code sets; some of the standard convolutional codes are shown in Table 3-2.

Table 3-2: **Standard Convolution Codes**

Constraint Length	Output Rate = 2		Output Rate = 3	
	binary	octal	binary	octal
7	1111001 1011011	171 133	1001111 1010111 1101101	117 127 155
9	101110001 111101011	561 753	101101111 110110011 111001001	557 663 711

The ACS module decodes for each state in the trellis. Thus, for a constraint length 7 decoder which has 64 states, there are 64 sub-blocks in the ACS block.

If the core is implemented in serial mode, the amount of silicon required for each sub-block is very small. See Performance, page 8 for further characterization of the decoder.

Traceback

The final block in the decoder is the traceback block. The actual decoding of symbols into the original data is accomplished by tracing the maximum likelihood path backwards through the trellis. Up to a limit, a longer sequence of tracing results in a more accurate path through the trellis. After many symbols equal to at least six times the constraint length, the decoded data is output.Thus the Viterbi always requires at least twice the traceback length of data to be input subsequently to a given input for that input to be successfully decoded. The traceback starts from zero or best state; the best state is estimated from the ACS costs. The traceback length is the number of trellis states processed before the decoder makes a decision on a bit. The decoded data is output only after a traceback length number of bits has been traced through. In other words, the traceback length determines the length of the training sequence for the Viterbi Decoder.

The length of the traceback is parameterizable and can be set to any value between 12 and 128. For the reduced latency option, the traceback length can only be a multiple of 6 between 12 and 126. The recommended value for non-punctured decoding is at least 6 times the constraint length. For data that has been punctured, that is, symbols removed prior to transmission over the channel, a larger value traceback length is required; usually, it is at least 12 times the constraint length to obtain optimal BER performance.

A best state option is available to select the starting location for the traceback from the state with minimal cost. There is also a reduced latency option which reduces the latency on the core by approximately half. The latency is of the order of two times the traceback length for reduced latency; see Latency, page 8. The reduced latency option has a slight speed penalty and is not available with the multichannel or serial core. The traceback block is implemented in block RAM, and the larger the value of the traceback length, the greater the block RAM requirements. See Performance, page 8 for additional details.

Serial Decoder

Data can be processed a bit at a time if the serial option is selected. This results in a smaller design but also a significant increase in latency. The number of clock cycles needed to process each set of input symbols depends on the output rate and the soft width of the data. See Table 3-3 for the minimum number of clock cycles required for each set of input symbols.

number of clock cycles = soft_width + output_rate + 6

The enabling of the data through the core is controlled by the `tvalid` and `tready` signals on the `s_axis_data_tdata`. See Figure 3-3 for an example of the AXI handshaking on the serial core.

Table 3-3: **Minimum Required Clock Cycles Per Input Symbol Set for a Rate 1/2 Serial Decoder**

Soft Width	Minimum Clock Cycles
3	11
4	12
5	13

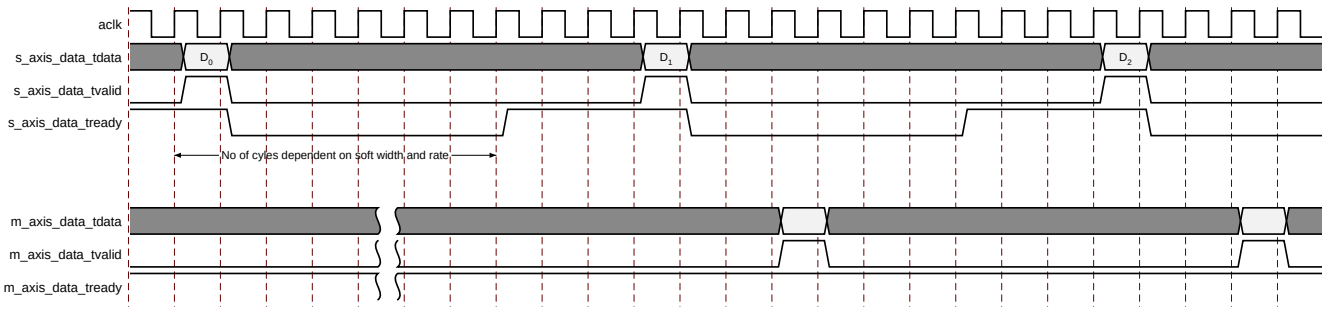

Figure 3-3: **Handshaking Example on the Serial Core**

Multichannel Decoder

The multichannel decoder decodes many interlaced channels using a single Viterbi Decoder. The input to the multichannel decoder is interlaced encoded data on the slave DATA channel. For a channel count of 3, channel 1 data is input followed by channel 2 and then channel 3 in a repeating sequence. The output is interlaced decoded data on the master DATA channel. See Figure 3-4. The multichannel decoder can decode from 2 to 32 channels. The larger the number of channels, the greater the block RAM requirements, as each channel requires its own traceback.

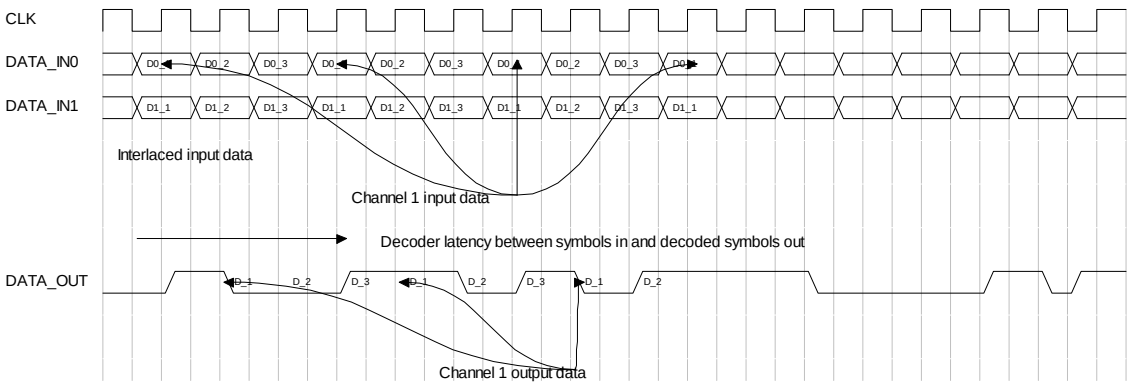

Figure 3-4: **Multichannel Decoder with Channel Count 3**

The BER from the multichannel decoder, if selected, gives the average number of errors present over all the input channels.

The multichannel decoder can decode at high speeds, but the true output rate of the decoder is equal to the speed divided by the number of channels. See Performance, page 8 for characterization results on the multichannel decoder.

Trellis Mode Decoder

For systems that are both power-limited and bandwidth-limited, Trellis Coded Modulation (TCM), or Pragmatic Trellis Coded Modulation (PTCM) as it can be known, is used. The modulation schemes are generally 8-PSK and 16-PSK. The LogiCORE™ IP Viterbi Decoder as used in a trellis decoder system is shown by the grayed out box in the trellis mode system diagram shown in Figure 3-5. In trellis mode, the BMU in the Viterbi is not used, but the costing is done externally using a Branch Metrics cost table. The ACS and traceback sections are used as normal. The output from the cost table is four TCM buses and a 4-bit Sector Bus. The address for the Branch Metrics cost table is provided by the I and Q outputs after symbol recovery from the PSK demodulator.

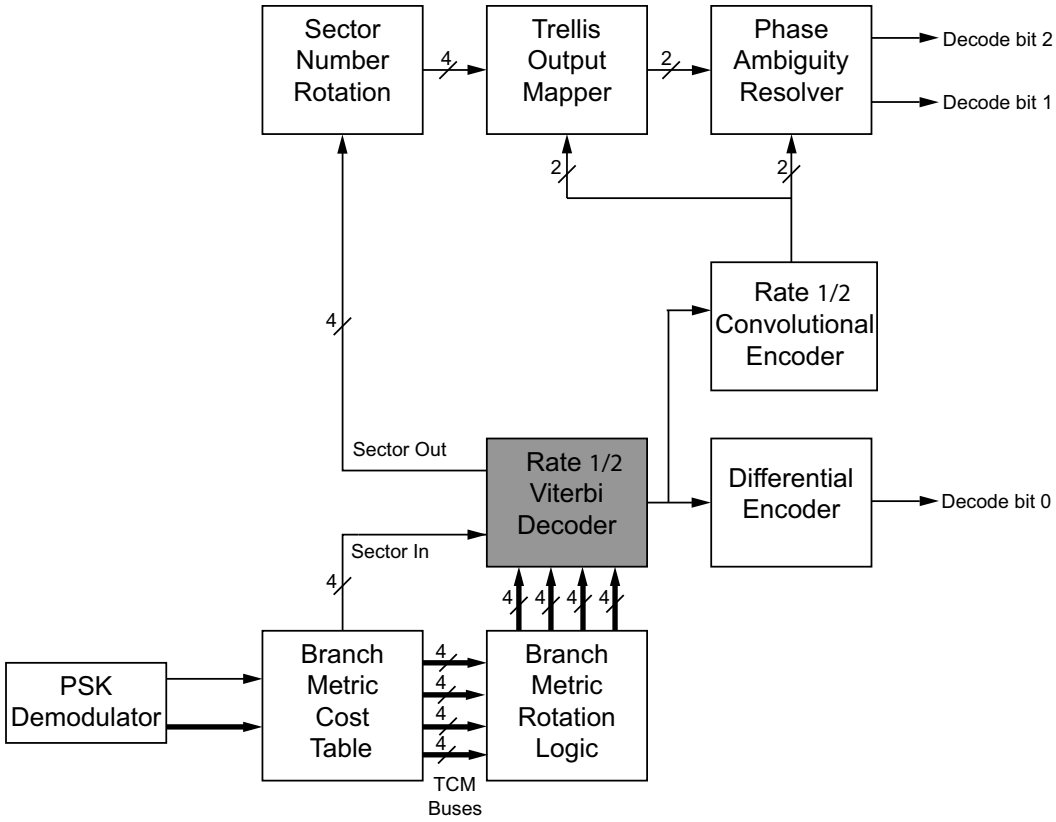

Figure 3-5: **Viterbi Decoder in Trellis Receiver**

Pragmatic Trellis Coded Modulation (PTCM) has the following setup:

- A standard rate 1/2 Viterbi Decoder is used.
- The data is costed externally to the decoder, bypasses the BMU, and sent to the ACS. Some points to note:
  - Each TCM bus can be 4 to 6 bits wide depending upon the TCM bus width selected on the Viterbi Decoder.
  - The 4 to 6 bits are unsigned cost values that are applied to the ACS module in the Viterbi Decoder. If the width of the generated costs is less than the (soft width+1), then the data should be tied to the lower bits and the remaining TCM input bits tied to zero.
- The received symbol or phase angle is converted to four branch metrics and a sector number externally to the decoder using an external lookup table.
- The sector number identifies the part of the I-Q plane where the symbol was received. The branch metrics are then processed by the Trellis Mode Decoder. The sector number is delayed by the Viterbi latency in the Trellis Mode Decoder.
- The costed data and sector are input to the core on the `s_axis_data_tdata` bus. The decoded data along with the output sector are output from the core on the `m_axis_data_tdata` bus.

Dual Rate Decoder

For a given constraint length and traceback-length, the core can function as a Dual Decoder, that is, two sets of convolutional codes and output rates can be used internally to the decoder. The dual-decoder offers significant device area savings when two different decoders with the same constraint length are required. For example, as a constraint length 7 decoder the core can decode as a rate 1/2 decoder and a rate 1/3 decoder. The implementation requires only a little additional logic for the extra costing involved in the BMU and some multiplexing in the ACS unit (see Figure 3-1). The Dual Decoder can be implemented as either parallel or serial architecture, and erasure pins can be present on the input. The selection of the decoder rate and codes is through the SEL pin (see Figure 3-6). When the SEL pin is Low, output rate0 and convolution0_codes are used in the decoding. When the SEL pin is High, then the rate is 1/output_rate1, and the convolution1_codes are used to decode the incoming data. The SEL_O pin shows the decoded data corresponding to the original SEL set of data points.The number of input data buses is equal to the max of the two output rates.

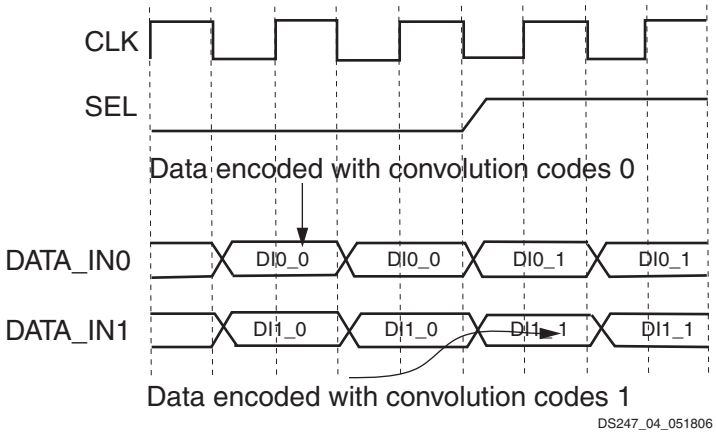

Figure 3-6: **SEL Input for Dual Decoder**

Erasure

If the data has been punctured prior to transmission, then de-puncturing is carried out externally to the Viterbi Decoder (see Figures 3-7 and 3-8). The presence of null-symbols (that is, symbols which have been deleted prior to transmission across the channel) is indicated using the erasure input ERASE. The decoder functions exactly as a non-punctured Viterbi Decoder, except the corresponding DATA_IN inputs are ignored when the erased input bit is High. If ERASE(0) is High, then the input on DATA_IN0 is viewed as a null-symbol. If ERASE(1) is High, then the input on DATA_IN1 is viewed as a null symbol. Although the normal usage of erasure is with a rate 1/2 decoder, the erasure pins can be present for output rates greater than 2. Erasure can be used with the Standard, Multichannel and Dual Decoder. Erasure cannot be present on the Trellis Mode Decoder. See the timing diagram in Figure 3-9 for a decoder working with external rate 3/4 erasure.

Note: DATA_IN0 and DATA_IN1 are the fields on the `s_axis_data_tdata` bus, and ERASE(0) and ERASE(1) are on the `s_axis_data_tuser` bus. The output data, DATA_OUT, is on the `m_axis_data_tdata` bus.

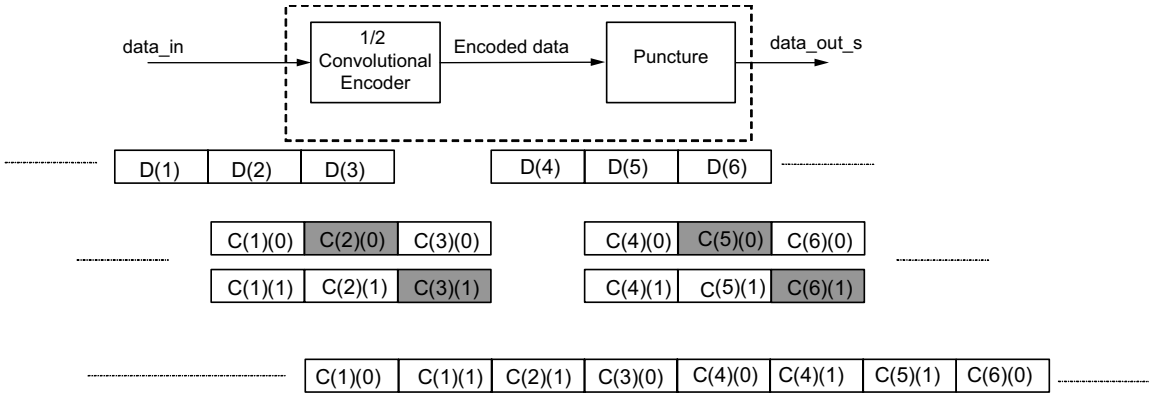

Figure 3-7: **Puncturing Encoded Data with 3/4 Puncture Rate with Single-Channel Output**

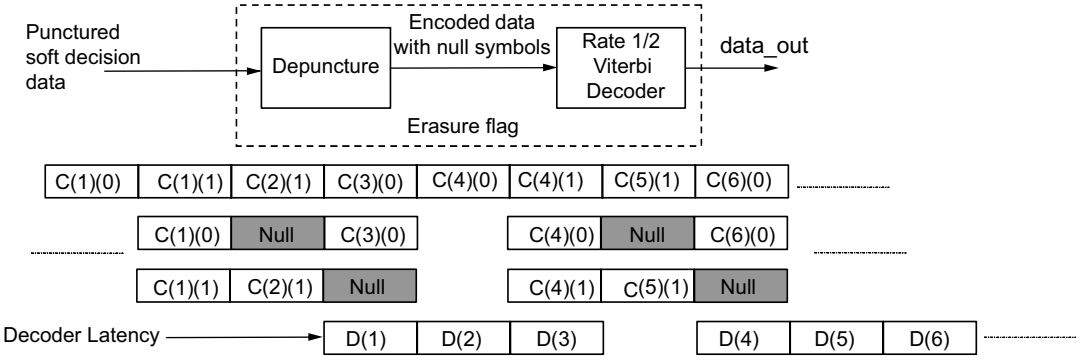

Figure 3-8: **De-puncturing Rate 3/4 Punctured Data with Single-Channel Soft Input and Erasure**

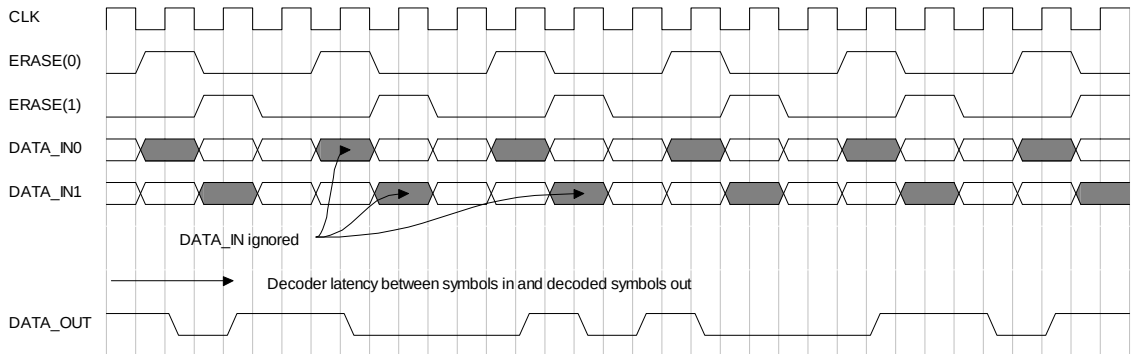

Figure 3-9: **Viterbi Decoder with Erasure Input Following a Rate 3/4 Pattern**

BER

The Bit Error Rate (BER) option on the decoder monitors the error rate on the transmission channel. Decoded data from the Viterbi Decoder is re-encoded using the Convolutional Encoder and compared with a delayed version of the data input to the decoder. An error is indicated if the delayed and encoded data differ (see Figure 3-10). The count is

incremented on a symbol-by-symbol basis. For example, if both the I & Q outputs differ from the expected I and Q for a rate 1/2 decoder, then this is only considered as one error on the BER count. The two sets of symbols can differ if there is an error on the channel or if the Viterbi Decoder has decoded incorrectly. The probability of the decoder incorrectly decoding is significantly smaller than the probability of a channel bit error; therefore the BER output gives a good estimate of the errors on the channel.

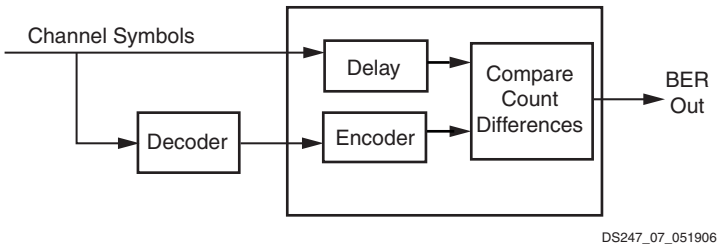

Figure 3-10: **Bit Error Rate Calculation**

The BER symbol count which is input on the DSTAT channel determines the number of input symbols over which the error count takes place.This value can be varied dynamically. The output error count BER is the number of errors that has been counted during the BER symbol count. `m_axis_dstat_valid` is asserted when BER symbol count input symbols have been processed and the bit error rate value is present on `m_axis_dstat_tdata`. The BER output always has a width of 16; therefore the maximum number of errors that can be counted is (2^{16}-1). If more errors occur than this upper limit, the maximum number of errors is output, that is, BER is set to all 1s.

Normalization

The NORM signal is an optional output that gives immediate monitoring of the errors on the channel. As the metrics grow in the ACS unit, they must be normalized to avoid overflow. When normalization occurs the decoder subtracts a fixed value from all metrics and asserts the normalization signal. If the ACS unit requires normalization, then there are uncertainties or errors on the channel. The more frequent the normalization, the higher the rate of errors present. The actual frequency of the normalization depends on many factors, in particular the soft width and the output rate of the decoder.

The normalization signal can be used to detect Viterbi synchronization. A high normalization rate (exceeding a certain threshold) indicates loss of synchronization. A low normalization rate (less than a certain threshold) indicates Viterbi synchronization has been achieved. In general, the normalization rate is inversely proportional to the Signal-to-Noise Ratio (SNR) at the decoder input.

Synchronization

If required the synchronization status of the core can be monitored externally to the core. The method involves the analysis of both the normalization output from the ACS modules

within the core and the BER performance of the core. A complete description of the method used is provided in [Ref 7].

The Normalization rate by itself gives an indication of the Viterbi Decoder synchronization status. A high normalization rate, exceeding a predetermined threshold, implies a loss of synchronization. Similarly, the BER rate of the core, by itself, can indicate a loss of synchronization. Thus, if the BER rate, or the normalization rate, was used in isolation, the threshold required would vary with the noise on the channel. The BER rate, like the normalization rate, would therefore be dependent on the signal to noise ratio Eb/No. To monitor synchronization, both the BER rate and the normalization rate are required to achieve a synchronization method that is independent of Eb/No.

Packet Handling

Viterbi decoding is a continuous operation, but the input to the decoder can be packet based rather than continuous streams. For data encoded in packets, it is necessary to terminate the encoder between the packets by the insertion of what is called zero tail bits. For a constraint length 7 decoder, there are 6 zero bits inserted into the encoder at the end of the packet. For a general Viterbi (constraint length -1), tail bits are required to return the encoder back to state zero. The effect of these zero tail bits is to return the Viterbi trellis to zero state and also the next packet starts from state zero. The Viterbi handles tail bits when working in any of the basic modes; no additional signals or control circuitry is required.

Although zero-tail bits or zero-tail termination is the standard method for handling packets within the Viterbi Decoder, there is a rate loss on the channel caused by constraint length -1 information bits being added to the original message. If the original packet contained m bits, the output code words are of length m + K -1, where K is the constraint length. Thus, the effective rate on the channel becomes:

$$\text{RateNew} = Rate X\left(1 - \frac{K-1}{m+K-1}\right)$$

For large packets, the rate loss becomes insignificant. For smaller packets, the method of tail-biting avoids the issue of the fractional rate loss by letting the last K-1 information bits define the starting state of the encoder. Only the data is encoded, that is, exactly m encoded bits are produced. In this case, no rate loss occurs and the encoding always starts and ends in the same state, but not necessarily the zero state. For a full description of the Viterbi Decoder and trellis termination and tail-biting, see [Ref 6].

General Design Guidelines

The following section provide guidelines on the best design practices for inclusion of the LogiCORE™ IP Viterbi Decoder core within a system.

Data Format

Use soft coding, as this gives a better BER performance. For the soft coding width, 3 or 4 bits should be sufficient. If the width is any larger, the size of the ACS is increased but does not give a large improvement in BER performance.

The format for soft width 3 is shown in Table 3-1. To extend the soft width input, for example, 4 bits, see the example shown:

	Signed Magnitude	Offset Binary
Strongest..1	1111	1111
Weakest..1	1000	1000
Weakest..0	0000	0111
Strongest..0	0111	0000

For the signed magnitude, it is the lower bits that are extended. The MSB is the sign bit. For offset binary, a lower bit is added to extend the range. In the example, this is a range from low, 0, to high, 15.

Data Input

Remove the DC bias in the soft input data before entering the data into the Viterbi Decoder. Use symmetric rounding when the soft data width is larger than the Viterbi input data width.

IMPORTANT: *The Viterbi input format is a balanced number system, while the 2s complement number system is unbalanced with one more negative number than positive number, which can lead to 1/2 LSB DC offset if mapping does not take this into consideration.*

Best State

This is on by default because it tends to give a better BER performance, especially for punctured data. The improvement is in the order of 0.25 dB. For non-punctured data, the option is not really needed. However, there is a penalty in the area of the core if this option is used. There is the flexibility to change the width if there are issues with noise.

Control Signals

Use the `aresetn` so as to put the decoder in a good start-up state. Make use of the DCM Lock signal to control the `aresetn`. If there is a need to save power, and not use the core, use the `aclken` to stop, or start the core operations. However, do not use `aclken` or `aresetn` to deal with packets of data. If dealing with blocks of data, use the BlockIn/Out signals. To qualify the output data AND the `tvalid` and BlockOut signals together.

TVALID

Use the `tvalid` signal to validate the data. This signal could be used as a clock enable to store the output from the Viterbi Decoder, for example, in a FIFO or memory.

NORM

Use the NORM signal to get an idea of the errors in the Viterbi Decoder and how it is dealing with the internal path metrics. If it is a noisy system, there is normalizing of the metrics and a NORM pulse appears.

Keep it Registered

To simplify timing and increase system performance in an FPGA design, keep everything registered, that is, all inputs and outputs from the user application should come from, or connect to a flip-flop. While registering signals might not be possible for all paths, it simplifies timing analysis.

Viterbi Decoder Non-features Summary

This section outlines the features that might be external to the core, depending upon the system being implemented.

Multichannel BER

If there is a requirement for individual BER circuits, these have to be done externally to the core.

Log Likelihood Ratio

The Log Likelihood Ratio (LLR) values that create the soft code inputs have to be generated externally to the core. See resources on the internet for information on how to do this.

De-puncturing

The Viterbi Decoder does not perform any kind of internal de-puncturing. Any de-puncturing has to be performed externally.

Trellis Mode

In this mode, the external costing table has to be created. See Trellis Mode Decoder.

Factors Affecting BER Performance

When viewing the Viterbi Decoder as part of a whole communication system, there are a few parameters that could affect BER performance. For the core, parameters such as data format, constraint length, and traceback length, use of best state have an effect. However, these are generally determined by the standard being implemented. After these have been taken care of, then outside factors need to be taken into consideration, for example modulation type, the channel model used, Eb/No value, convolutional codes and DC bias. For factors that affect packets of data, see [Ref 6].

Clocking

The core has a single clock, `aclk`, and all signals are synchronous to the `aclk` input.

Resets

The Viterbi Decoder core uses a single, optional, reset input called `aresetn`. `aresetn` is active-Low. See aresetn.

Protocol Description

AXI4-Stream Protocol

The use of AXI4-Stream interfaces brings standardization and enhances interoperability of Xilinx® IP LogiCORE™ solutions. Other than general control signals such as `aclk`, `aclken` and `aresetn`, and event outputs, all inputs and outputs to the core are conveyed using AXI4-Stream channels. A channel consists of `tvalid` and `tdata` always, plus several optional ports and fields. In the Viterbi Decoder, the additional ports used are `tuser` and `tready`. Together, `tvalid` and `tready` perform a handshake to transfer a value, where the payload is `tdata`. The payload is indeterminate when `tvalid` is deasserted.

The Viterbi Decoder operates on the values contained in the S_AXIS_DATA channel `tdata` fields and outputs the results in the `tdata` fields of the M_AXIS_DATA channel. For further details on AXI4-Stream Interfaces see [Ref 4] and [Ref 5].

Basic Handshake

Figure 3-11 shows the transfer of data in an AXI4-Stream channel. `tvalid` is driven by the source (master) side of the channel and `tready` is driven by the receiver (slave). `tvalid` indicates that the value in the payload fields (`tdata` and `tuser`) is valid. `tready` indicates that the slave is ready to receive data. When both `tvalid` and `tready` are TRUE in a cycle, a transfer occurs. The master and slave set `tvalid` and `tready` respectively for the next transfer appropriately.

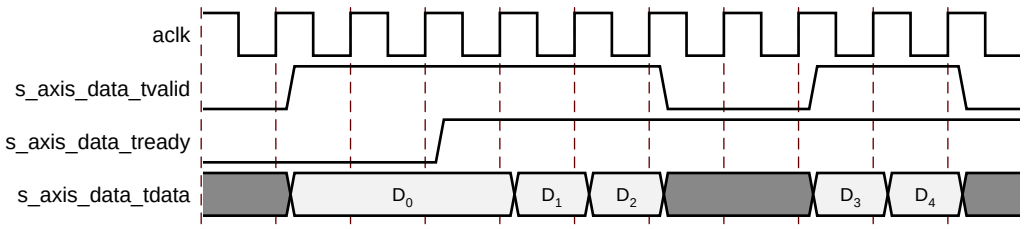

Figure 3-11: **Data Transfer in an AXI4-Stream Channel**

The full flow control of AXI4-Stream aids system design because the flow of data is self-regulating. Data loss is prevented by the presence of back pressure (`tready`), so that data is only propagated when the downstream datapath is ready to process it.

For the main output channel, M_AXIS_DATA, if the output is prevented from off-loading data because `m_axis_data_tready` is Low then data accumulates in the core. When the core internal buffers are full the core stops further operations. When the internal buffers fill, the `tready` (`s_axis_data_tready`) is deasserted to prevent further input. This is the normal action of back pressure.

For the status output channel, M_AXIS_DSTAT, if `m_axis_dstat_tready` is held Low, the core does not overwrite the internal BER value until the current value within the core has been read because there is no internal buffering on the DSTAT channel. The core holds `m_axis_dstat_tdata` static after `m_axis_dstat_tvalid` goes High until `tready` is asserted, see Figure 3-12.

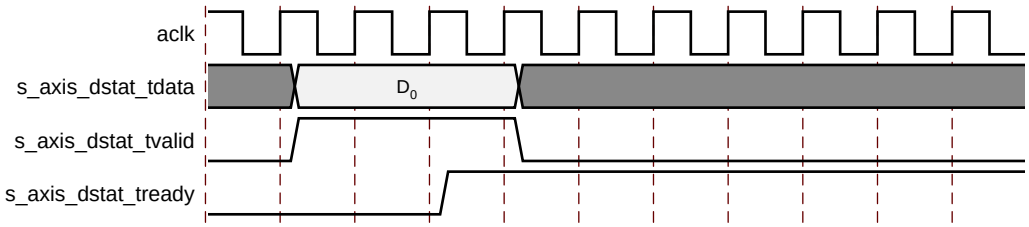

Figure 3-12: **DSTAT Handshaking**

aclken

The clock enable input (`aclken`) is an optional pin. When `aclken` is deasserted (Low), all the other synchronous inputs are ignored, except `aresetn`, and the core remains in its current state. This pin should be used only if it is genuinely required because it has a high fanout within the core and can result in lower performance. `aclken` is a true clock enable and causes the entire core to freeze state when it is Low.

An example of `aclken` operation is shown in Figure 3-13. In this case, the core ignores symbol D_4 as input to the block, and the current `m_axis_data_tdata` value remains unchanged.

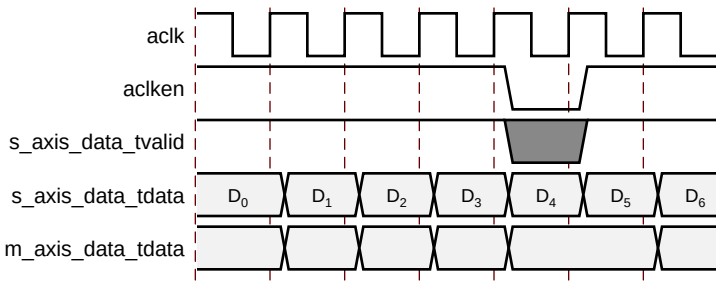

Figure 3-13: **Clock Enable Timing**

aresetn

The synchronous reset (`aresetn`) input can be used to re-initialize the core at any time, regardless of the state of `aclken`. `aresetn` needs to be asserted Low for at least two clock cycles to initialize the circuit. The core becomes ready for normal operation two cycles after `aresetn` goes High, if `aclken` is asserted. Note that the block RAM is not cleared with the `aresetn` signal and there is a block of previously decoded data output from the traceback prior to correct decoding resuming. The `tvalid` signal on `m_axis_data` is only asserted when valid data is available on `m_axis_data_tdata`. The timing for the `aresetn` input is shown in Figure 3-14.

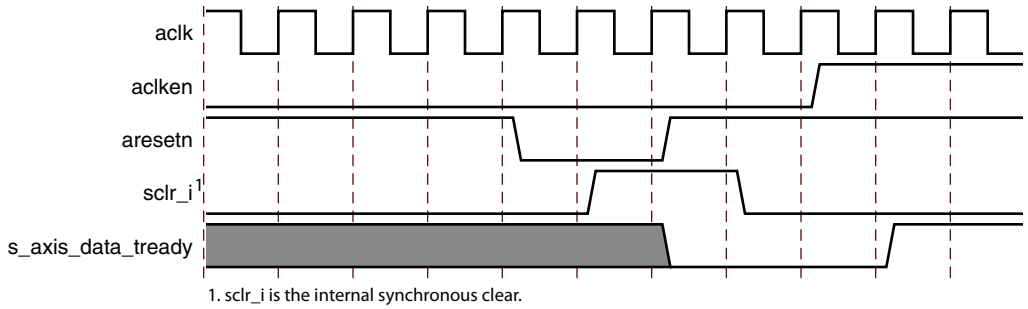

Figure 3-14: **Synchronous Reset Timing**

S_AXIS_DATA Channel

s_axis_data_tdata

Data to be processed is passed into the core on this port. To ease interoperability with byte-oriented buses, `tdata` is padded with zeros because the Viterbi Decoder bus width can vary, depending on the type of Viterbi and the output rate. The padding bits are ignored by the core and do not result in additional resource use. The structure is shown in Figure 3-15 for an input rate 2 decoder. The `tdata` carries the data to be decoded. The encoded bits on each of the `data_in` inputs can be hard coded (bus width 1) or soft coded (bus width 3 to 5). The width of the input for a non-trellis decoder is always 8 times the output rate. If the Dual Decoder is selected, the number of `data_in` inputs is equal to the maximum output rate.

The input format for the trellis decoder is shown in Figure 3-16. The width of the trellis mode inputs can range from 4 to 6 corresponding to a data width of 3 to 5. The trellis mode inputs are the outputs from an external costing of the data. There are always four inputs to the decoder, and the decoder always functions as a rate 1/2 decoder when trellis mode is selected. The width for the trellis decoder is always 40 bits with the sector input residing in the top byte.The SECTOR bus has width 4. The SECTOR input is delayed by the decoder delay and output to the SECTOR bus in `m_axis_data_tdata`. See Trellis Mode Decoder. The following buses are available on `s_axis_data_tdata` for all decoders except the Trellis Mode Decoder:

DATA_IN0	input data which can be 1 bit for hard decoding or 3 to 5 bits wide for soft coding
DATA_IN1	input data which can be 1 bit for hard decoding or 3 to 5 bits wide for soft coding
DATA_IN2	input data which can be 1 bit for hard decoding or 3 to 5 bits wide for soft coding
	.... up to the output rate

The following buses are available on `s_axis_data_tdata` for the Trellis Mode Decoder:

TCM00, TCM01... TCM11	Input trellis data with width of 4 to 6 bits
SECTOR	Input sector of width 4 bits

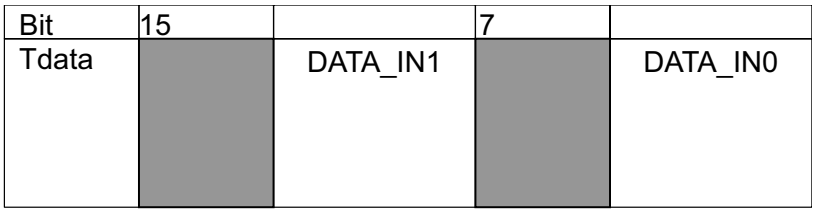

Figure 3-15: **Input tdata for Standard Output Rate 2 Decoder with Soft Input**

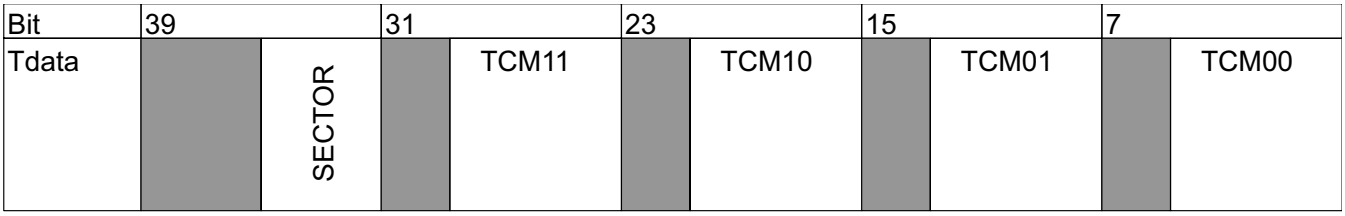

Figure 3-16: **Input tdata for Trellis Mode Decoder**

s_axis_data_tuser

This port is only present if the core is punctured, or is a Dual Decoder or the block valid signal is used with the core.

ERASE — This erase input bus is only required where data on the channel has been punctured. The inputs are used to indicate the presence of a null-symbol on the corresponding `data_in` buses. ERASE(0) corresponds to DATA_IN0, ERASE(1) corresponds to DATA_IN1, ... If an erase pin is High, the data on the corresponding `data_in` bus is treated as a null-symbol internally to the decoder. The width of the erase bus is equal to the output rate of the decoder with a maximum value of 7.

SEL — Optional, controlled by the Dual Decoder option. This is used to select the correct set of convolutional codes for the decoding of the input data symbols in the Dual Decoder case. When SEL is Low, the input data is decoded using the first set of convolutional codes. When it is High, the second set of convolutional codes is applied. See Figure 3-6.

BLOCK_IN — Optional, controlled by the BLOCK VALID option. Marker signal used to tie output to input. The BLOCK_IN pin is delayed by the latency of the decoder and output as BLOCK_OUT on the M_AXIS_DATA_TUSER bus. The BLOCK_OUT pin shows the decoded data corresponding to the original BLOCK_IN set of data points.

The width of `s_axis_data_tuser` is always a multiple of 8 bits and is determined by the presence or absence of these three signals.

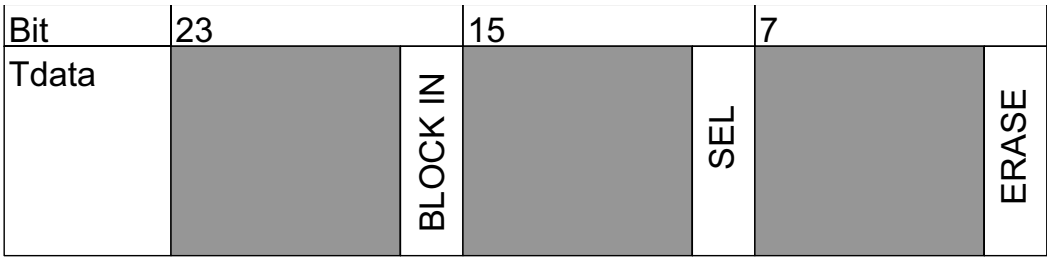

Figure 3-17: **Input TUSER**

M_AXIS_DATA Channel

m_axis_data_tdata

This output bus is composed of multiple fields:

DATA	Decoded output data always 1 bit.
SECTOR	This output is present if the decoder is a Trellis Mode Decoder and is always 4 bits. The output SECTOR is a delayed version of the input SECTOR bus. Both buses have a fixed width of 4 bits. The delay equals the delay through the Trellis Mode Decoder.

The width of the `m_axis_data_tdata` is 8 bits if the core is not a trellis decoder and 16 bits otherwise. See Figure 3-18.

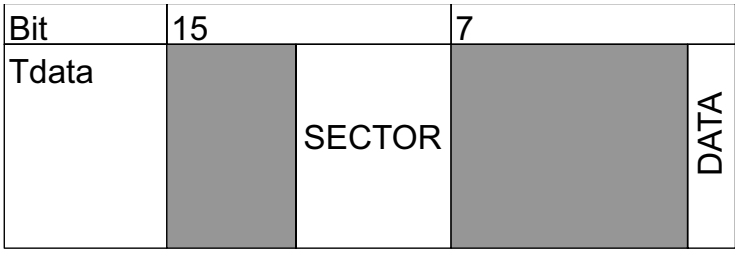

Figure 3-18: **tdata Output**

m_axis_data_tuser

This port is only present if the core is a Dual Decoder or has a normalization signal or block valid is present.

SEL	This signal is a delayed version of the SEL signal. The delay equals the delay through the Dual decoder.
BLOCK_OUT	This signal is a delayed version of the BLOCK_IN signal. The BLOCK_OUT signal shows the decoded data corresponding to the original BLOCK_IN set of data points. The delay equals the delay through the decoder.
NORM	The NORM output indicates when normalization has occurred within the core. It gives an immediate indication of the rate of errors in the channel. See Normalization, page 21 for additional details.

The width of `m_axis_data_tuser` is always a multiple of 8 bits and is determined by the presence or absence of these three signals.

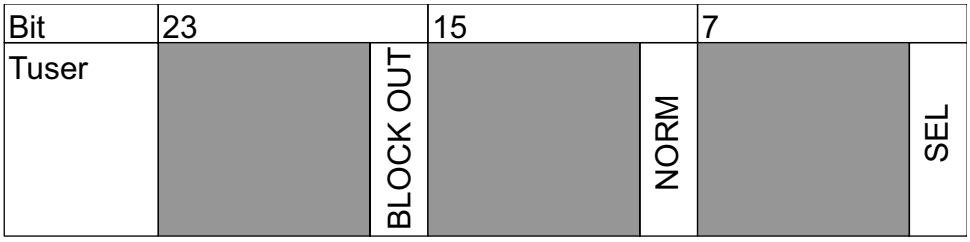

Figure 3-19: **TUSER Output**

S_AXIS_DSTAT Channel

s_axis_dstat_tdata

BER_RANGE This is the number of symbols over which errors are counted in the BER block, see BER, page 20.

The width of `s_axis_dstat_tdata` is always 16 bits.

M_AXIS_DSTAT Channel

m_axis_dstat_tdata

BER The Bit Error Rate (BER) bus output (fixed width 16) gives a measurement of the channel bit error rate by counting the difference between the re-encoded DATA_OUT and the delayed DATA_IN to the decoder. For a full description of BER, see BER, page 20. Trellis mode does not support a BER output.

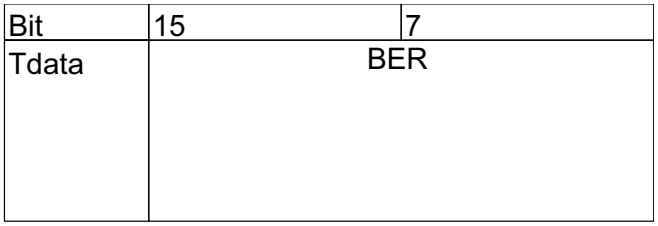

Figure 3-20: **DSTAT Output**

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 8]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 10]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 11]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 12]

Customizing and Generating the Core

This section includes information about using Xilinx tools to customize and generate the core in the Vivado Design Suite.

If you are customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 8] for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value you can run the `validate_bd_design` command in the Tcl Console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 10] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 11].

This section defines the Viterbi Decoder core parameters, according to the layout in the Vivado Integrated Design Environment.

Note: The controls in the System Generator for DSP GUI work identically to those in the Vivado Design Suite, although the layout differs.

Viterbi Type Tab

The core customization parameters found in this tab are as follows.

Component Name

The component name is used as the base name of the output files generated for the core. Names must begin with a letter and must be composed of the following characters: a to z, 0 to 9, and "_".

Viterbi Type

Select one of the four Viterbi Decoder types:

- **Standard**: This type is the basic Viterbi Decoder.
- **Multichannel**: This type allows many interlaced channels of data to be decoded using a single Viterbi Decoder. The number of channels to be decoded can be any value between 2 and 32. See Multichannel Decoder, page 16.
- **Trellis Mode**: This type is a Trellis Mode Decoder using the costed data and SECTOR inputs. See Trellis Mode Decoder, page 17.
- **Dual Decoder**: The type is the Dual Decoder that can be operated in dual mode with two sets of convolutional codes. The SEL pin is present on the slave and master TUSER buses when the decoder operates in this mode. See Dual Rate Decoder, page 19.

Decoder Options

- **Constraint Length**: This is the length of the constraint register in the encoder plus 1. This value can be any integer in the range 7 to 9 inclusive.
- **Traceback Length**: This is the length of the survivor or training sequence in the traceback through the Viterbi trellis. Optimal length for the traceback is considered to be at least 6 times the constraint length for non-punctured data. For the multichannel Viterbi, the traceback length is the length of the traceback for each channel in the decoder. For the reduced latency option, the traceback length must always be divisible by 6. For punctured data, the length should be at least 12 times the constraint length. Increasing traceback length might increase RAM requirements. Increasing traceback length also increases the latency of the core. See Latency, page 8 for additional details.
- **Use Reduced Latency**: This option reduces the latency on the core by approximately half. The reduced latency option has a slight speed penalty and is not available with the multichannel Viterbi. See Latency, page 8 section for additional details.

Architecture

- **Parallel**: Large but fast Viterbi Decoder.
- **Serial**: Small but serial processing of the input data (see Serial Decoder).

Architecture and Data Format Tab

The parameters found in this tab are as follows.

Best State

The best state option starts the traceback of the core from the optimal state.

- **Use Best State**: The best state selection gives improved BER performance for highly punctured data.
- **Best State Width**: The best state selects the best state from the costs for each state. Most of the lower bits in the cost are redundant in the cost comparison, and the best state area requirements can be reduced by selecting a smaller width than the full ACS width. If the width is set to 6, then the full cost is used in the best state selection for a soft width of 3. If the width is set to 3, then the lower three bits are ignored in the best state calculations.

Puncturing Options

- **None**: This indicates there is no external puncturing on the core.
- **External (Erased Symbols)**: This indicates the presence of the erased bus ERASE on the TUSER input and allows the core to be de-punctured externally prior to decoding. ERASE(0) High indicates that the sample on DATA_IN0 is a null symbol; ERASE(1) High indicates that the data on DATA_IN1 is a null symbol;... The size of the erase bus is equal to the output rate of the decoder, or the maximum output rate if the Dual Decoder is selected.

Coding

There are two types of coding available: Soft Coding and Hard Coding.

- **Soft Coding**: Uses the Euclidean metric to cost the incoming data against the branches of the Viterbi trellis.
- **Hard Coding**: Uses the Hamming difference between the input data bits and the branches of the Viterbi trellis. Hard coding is only available for the standard parallel core.

Soft Width

The input width of soft-coded data is in the range 3 to 5. Larger widths require more logic. If the core is implemented in serial mode, larger soft widths also increase the serial processing time. See Table 3-3 for the minimum number of clock cycles required for a rate 1/2 decoder in the serial case.

Data Format

There are two data formats available for Soft Coding: Signed Magnitude and Offset Binary Table 3-1 shows the required format for the data for the case of soft width 3 for each of the data types. Soft width 4 and 5 follow a similar format.

Output Rate and Convolution Code Tab

The parameters found in this tab are as follows.

Convolution Code0 Radix

The convolutional codes can be input and viewed in binary, octal, or decimal.

Output Rate0

Output Rate is the symbol output rate at the Encoder. Output Rate0 can be any value from 2 to 7. Output Rate0 is the output rate used if the decoder is non-dual. If the decoder is dual, then Output Rate0 is the first output rate and the rate used by the decoder when the SEL input is Low.

Convolution0 Codes

These codes are the convolutional codes used in the encoder. The codes can be entered (and viewed) in binary, octal, and decimal. If the decoder is dual, then Convolution0 Codes are the codes applied in the decoder when the SEL input is Low.

If the Dual Decoder type is selected, then a second output rate and convolution code selection tab is shown:

Output Rate1

Output Rate1 can be any value from 2 to 7. This is the second output rate used if the decoder is dual. The incoming data is decoded at this rate when the SEL input is High. Output Rate1 is not used for the non-Dual Decoder and the tab is only available if Dual Decoder is selected.

Convolution Code1 Radix

The convolutional codes can be input and viewed in binary, octal, or decimal.

Convolution1 Codes

The convolutional codes are used in the decoder when the decoder is dual and the SEL input is High. The codes are entered in binary, octal, or decimal.

Puncturing and BER Options Tabs

The parameters found in this tab are as follows.

BER Options

Use BER Symbol Count: Check this box if a Bit Error Rate (BER) monitor is required. The core compares the delayed incoming data with an encoded version of the outgoing decoded data to obtain an estimate of the BER on the channel. See BER, page 20 for additional details. The Number of BER symbols over which the estimate is given is dynamically variable and is input on the `s_axis_dstat_tdata` bus and can range from 3 to (2^{16}-1).

Optional Pins

Check the boxes of the optional pins that are required: NORM, BLOCK VALID, TREADY and ACLKEN. Select only pins that are genuinely required, because each selected pin results in more FPGA resources being used and can result in a reduced maximum operating frequency.

NORM

Check this box if a normalization output is required. For additional details, see Normalization, page 21.

Block Valid

Check this box if BLOCK_IN and BLOCK_OUT signals are required. These signals track the movement of a block of data through the decoder. BLOCK_OUT corresponds to BLOCK_IN delayed by the decoder latency.

User Parameters

Table 4-1 shows the relationship between the fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl Console).

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
Viterbi Type	viterbi_type	Standard
Number of Channels	channels	1
Constraint Length	constraint_length	7
Use Reduced Latency	reduced_latency	False
Traceback Length	traceback_length	42
Architecture	architecture	Parallel
Use Best State	best_state	True
Base State Width	best_state_width	3
Coding	coding	Soft_Coding
Soft Width	soft_width	3
Data Format	data_format	Signed_magnitude
Output Rate 0	output_rate0	2
Convolution Code 0 Radix	convolution_code_0_radix	Binary
Convolution 0 Code 0	convolution0_code0	1111001
Convolution 0 Code 1	convolution0_code1	1011011
Convolution 0 Code 2	convolution0_code2	0
Convolution 0 Code 3	convolution0_code3	0
Convolution 0 Code 4	convolution0_code4	0
Convolution 0 Code 5	convolution0_code5	0
Convolution 0 Code 6	convolution0_code6	0
Output Rate 1	output_rate1	2
Convolution Code 1 Radix	convolution_code_1_radix	Binary
Convolution 1 Code 0	convolution1_code0	1111001
Convolution 1 Code 1	convolution1_code1	1011011
Convolution 1 Code 2	convolution1_code2	0
Convolution 1 Code 3	convolution1_code3	0
Convolution 1 Code 4	convolution1_code4	0
Convolution 1 Code 5	convolution1_code5	0
Convolution 1 Code 6	convolution1_code6	0
Puncturing	puncturing	None
Use BER Symbol Count	ber_symbol_count	True

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
NORM	norm	False
Block Valid	block_valid	False
TREADY	tready	True
ACLKEN	aclken	false

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 10].

Constraining the Core

There are no constraints associated with this core.

Simulation

For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 12].

IMPORTANT: *For cores targeting 7 series or Zynq-7000 devices, UNIFAST libraries are not supported. Xilinx IP is tested and qualified with UNISIM libraries only.*

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 10].

Test Bench

This chapter contains information about the test bench provided in the Vivado® Design Suite environment.

Demonstration Test Bench

When the core is generated in the Vivado® IP catalog, a demonstration test bench is created. This is a simple VHDL test bench that exercises the core.

The demonstration test bench source code is one VHDL file in the IP catalog output directory:

```
<component_name>/demo_tb/tb_<component_name>.vhd
```

The source code is comprehensively commented.

Using the Demonstration Test Bench

The demonstration test bench instantiates the generated Viterbi Decoder core. If the Vivado Design Suite project options are set to generate a structural model, a VHDL or Verilog netlist named `<component_name>.vhd` or `<component_name>.v` is generated.

The general steps for using the demonstration test bench are:

1. Compile the netlist and the demonstration test bench into the work library (see your simulator documentation for more information on how to do this).
2. Simulate the demonstration test bench.
3. View the test bench signals in a waveform viewer to see the operations of the test bench.

Demonstration Test Bench in Detail

The demonstration test bench performs the following tasks:

- Instantiates the core
- Generates a clock signal

- Generates a source data table consisting of a sinusoid
- Serializes and convolution encodes the source data to create input data for the Viterbi Decoder core
- Inserts errors into the input data to demonstrate error correction
- If external erasure is supported, erase inputs using a predefined puncture pattern
- Drives the core input signals to demonstrate core features
- Checks that the core output signals obey AXI4 protocol rules (data values are not checked to keep the test bench simple)
- Provides signals showing the separate fields of AXI4-Stream `tdata` and `tuser` signals
- Provides signals showing the source data before serialization and encoding, and the deserialized decoded output data

The demonstration test bench drives the core input signals to demonstrate the features and modes of operation of the core. The operations performed by the demonstration test bench are appropriate for the configuration of the generated core and are a subset of the following operations:

1. An initial phase where the core is initialized and no operations are performed.
2. Decode data containing no errors.
3. Decode and correct data containing some errors.
4. Try to decode and correct data containing many errors, sometimes failing to correct the errors.
5. If BER statistics are supported, set up BER statistics over various ranges.
6. Demonstrate the use of AXI4-Stream handshaking signals `tvalid` and `tready`.
7. Demonstrate the effect of asserting `aresetn`.
8. If `aclken` is present: demonstrate the effect of toggling `aclken`.

Customizing the Demonstration Test Bench

It is possible to modify the demonstration test bench to use different source data or different control information.

Source data is pre-generated in the `create_src_table` function and stored in the SRC_DATA constant. Data from this constant is serialized by the `s_data_stimuli` process, convolution encoded by the `conv_data` procedure and driven into the core by the `encode_data` procedure, which also inserts errors and controls erasures. Data is driven continuously throughout the operation of the test bench: new input data is required for the Viterbi Decoder core to produce output data.

Source data before serialization is shown on the `s_axis_data_tdata_src_des` signal, which is synchronized to the corresponding serialized and encoded data being driven into the Viterbi Decoder core. Output data is deserialized by the `deserialize_output` process and shown on the `m_axis_data_tdata_des` signal. These signals can be viewed in a simulator to compare source data and decoded, corrected output data.

If external erasure is supported, the test bench models the effect of puncturing by a convolution encoder, by de-puncturing and inserting null symbols in the input data. The puncture rate is defined by the PUNC_INPUT_RATE and PUNC_OUTPUT_RATE constants, and the puncture pattern is defined in the PUNC_CODES constant. These constants can be modified to change the puncture rate and pattern.

BER statistics are controlled by the `s_dstat_stimuli` process: this selects the BER range and drives transactions on the S_AXIS_DSTAT channel to set up BER statistics generation.

Upgrading

This appendix contains information about migrating a design from ISE® to the Vivado® Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Migrating to the Vivado Design Suite

For information about migrating to the Vivado Design Suite, *see the ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 13].

The Vivado Design Suite IP update functionality can be used to update a core from v7.0 and v8.0 to v9.0, but the update mechanism alone does not create a core compatible with v7.0 and v8.0.

Parameter Changes

Table A-1 shows the changes to parameters from v7.0 and v8.0 to v9.0.

Table A-1: **Parameter Changes**

Version 7.0	Version 8.0 and 9.0	Notes
architecture	Unchanged	
ber_symbol_count	Ber_symbol_count	Different functionality as the BER count is now input on dynamically on the DSTAT channel
ber_thresh	Removed	Synchronization not supported in v8.0
best_state	Unchanged	
best_state_width	Unchanged	
block_valid	Unchanged	
ce	aclken	Renamed from ce to aclken for AXI4 standardization
channels	Unchanged	
coding	Unchanged	

Table A-1: **Parameter Changes *(Cont'd)***

Version 7.0	Version 8.0 and 9.0	Notes
component_name	Unchanged	
constraint_length	Unchanged	Support constraint length reduced to 7 to 9
convolution0_code0	Unchanged	
convolution0_code1	Unchanged	
convolution0_code2	Unchanged	
convolution0_code3	Unchanged	
convolution0_code4	Unchanged	
convolution0_code5	Unchanged	
convolution0_code6	Unchanged	
convolution1_code0	Unchanged	
convolution1_code1	Unchanged	
convolution1_code2	Unchanged	
convolution1_code3	Unchanged	
convolution1_code4	Unchanged	
convolution1_code5	Unchanged	
convolution1_code6	Unchanged	
convolution_code_0_radix	Unchanged	
convolution_code_1_radix	Unchanged	
data_format	Unchanged	
direct_traceback	Removed	Packet options not supported in v8.0
dynamic_thresholds	Removed	Synchronization not supported in v8.0
maximum_direct	Removed	Packet options not supported in v8.0
norm	Unchanged	
norm_thresh	Removed	Synchronization not supported in v8.0
number_of_ber_symbols	Removed	BER symbol count is input dynamically
output_rate0	Unchanged	
output_rate1	Unchanged	
puncturing	Unchanged	
rdy		Replaced with AXI4 control signals
reduced_latency	Unchanged	
soft_width	Unchanged	
synchronization	Removed	Synchronization not supported in v8.0
synchronous_clear	aresetn	aresetn always present on core

Table A-1: **Parameter Changes *(Cont'd)***

Version 7.0	Version 8.0 and 9.0	Notes
traceback_length	Unchanged	
trellis_initialization	Removed	Packet options not supported in v8.0
viterbi_type	Unchanged	

Port Changes

Table A-2 shows the changes to parameters from v7.0 and v8.0 to v9.0.

Table A-2: **Port Changes**

Version 7.0	Version 8.0 and 9.0	Notes
DATA_IN0	s_axis_data_tdata	Now exists as a field within s_axis_data_tdata
DATA_IN1	s_axis_data_tdata	
DATA_IN2	s_axis_data_tdata	
DATA_IN3	s_axis_data_tdata	
DATA_IN4	s_axis_data_tdata	
DATA_IN5	s_axis_data_tdata	
DATA_IN6	s_axis_data_tdata	
TCM00	s_axis_data_tdata	Now exists as a field within s_axis_data_tdata when the core is in trellis mode
TCM01	s_axis_data_tdata	
TCM10	s_axis_data_tdata	
TCM11	s_axis_data_tdata	
SECTOR_IN	s_axis_data_tdata	Occupies the top byte of the input tdata when the core is in trellis mode
BLOCK_IN	s_axis_data_tuser	Now exists as a field within s_axis_data_tuser
PACKET_START	Removed	Packet processing removed in v8.0 of the core
TB_BLOCK	Removed	
PS_STATE	Removed	
TB_STATE	Removed	
ber_thresh	Removed	Synchronization removed in v8.0 of the core
NORM_THRESH	Removed	
ERASE	s_axis_data_tuser	Now exists as a field within s_axis_data_tuser
DATA_OUT	m_axis_data_tdata	Now exists as a field within m_axis_data_tdata
DATA_OUT_DIRECT	Removed	Packet processing removed in v8.0 of the core
DATA_OUT_REVERSE	Removed	

Table A-2: **Port Changes *(Cont'd)***

Version 7.0	Version 8.0 and 9.0	Notes
PACKET_START_O	Removed	
TB_BLOCK_O	Removed	
DIRECT_RDY	Removed	
REVERSE_RDY	Removed	
BER	m_axis_dstat_tdata	The dstat input and output channel now handle the BER count which can be varied dynamically
BER_DONE	m_axis_dstat_tvalid	
NORM	m_axis_data_tuser	Now exists as a field within m_axis_data_tuser
SECTOR_OUT	m_axis_data_tdata	Occupies the top byte of the output data if the core is in trellis mode
BLOCK_OUT	m_axis_data_tuser	Now exists as a field within m_axis_data_tuser
OUT_OF_SYNC	Removed	Synchronization removed in v8.0 of the core
OOS_FLAG	Removed	
SEL	s_axis_data_tuser	Now exists as a field within s_axis_data_tuser
SEL_O	m_axis_data_tuser	Now exists as a field within m_axis_data_tuser
ND	s_axis_data_tvalid	
RFD	s_axis_data_tready	
RDY	m_axis_data_tready	
CE	aclken	Rename only
SCLR	aresetn	Rename and change on sense (now active-Low). Must now be asserted for at least 2 cycles
CLK	aclk	Rename only

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

Parameter Changes

No change.

Port Changes

No change.

Other Changes

No change.

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

TIP: *If the IP generation halts with an error, there might be a license issue. See License Checkers in Chapter 1 for more details.*

Finding Help on Xilinx.com

To help in the design and debug process when using the Viterbi Decoder, the Xilinx Support web page (Xilinx Support web page) contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the Viterbi Decoder. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool messages
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for the Viterbi Decoder

AR: 54512

Technical Support

Xilinx provides technical support in the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which files to include with the WebCase.

Debug Tools

There are many tools available to address Viterbi Decoder design issues. It is important to know which tools are useful for debugging various situations.

Vivado Design Suite Debug Feature

Vivado® lab tools insert logic analyzer (ILA) and virtual I/O (VIO) cores directly into your design. Vivado lab tools allow you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature represents the functionality in the Vivado I DE that is used for logic debugging and validation of a design running in Xilinx devices in hardware.

The Vivado lab tools logic analyzer is used to interact with the logic debug LogiCORE IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 14].

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado lab tools are a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the Vivado lab tools for debugging the specific problems.

Many of these common issues can also be applied to debugging design simulations.

General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `locked` port.
- If your outputs go to 0, check your licensing.

Interface Debug

AXI4-Stream Interfaces

If data is not being transmitted or received, check the following conditions:

- If the transmit `s_axis_data` is stuck Low, the core cannot send data.
- If the receive `m_axis_data` is stuck Low, the core is not receiving data.
- Check that the `ACLK` inputs are connected and toggling.
- Check that the AXI4-Stream waveforms are being followed Figure 3-11.

- Check core configuration.
- Add appropriate core specific checks.

Appendix C

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado® IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this product guide:

1. *Convolutional Encoder Product Guide* (PG026)
2. Convolutional Encoder Product Page
3. Viterbi Decoder Product Page
4. *Xilinx Vivado AXI Reference Guide* (UG1037)
5. *AMBA® AXI4-Stream Protocol Specification* (ARM IHI 0051A)
6. *Viterbi Decoder Block Decoding - Trellis Termination and Tail-Biting* (XAPP551)
7. *Viterbi Synchronization Data Sheet* (DS205)
8. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
9. *Vivado Design Suite User Guide: Implementation* (UG904)
10. *Vivado Design Suite User Guide: Designing with IP* (UG896)
11. *Vivado Design Suite User Guide: Getting Started* (UG910)
12. *Vivado Design Suite User Guide - Logic Simulation* (UG900)
13. *ISE® to ISE to Vivado Design Suite Migration Guide* (UG911)
14. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
02/04/2021	9.1	Added Versal ACAP support.
11/18/2015	9.1	Added support for UltraScale+ families.
10/01/2014	9.1	Version number changed to align with core version number.
04/02/2014	9.0	Updated Resource Utilization and Performance information.
12/18/2013	9.0	• Revision number advanced to 9.0 to align with core version number. • Template updated • Added UltraScale™ architecture support.