XAPP551 (v2.0) July 30, 2010

Viterbi Decoder Block Decoding - Trellis Termination and Tail Biting

Author: Michael Francis

Summary

Many digital communication standards employ convolution coding as a means of forward error correction (FEC). Data encoded in this way generally is decoded with a Viterbi decoder, which operates by constructing a *trellis* of state probabilities and branch metrics. The transmitted data is often terminated with a number of zeros to force the encoder back to the zero state. This allows the decoder to start decoding from a known state, however, extra symbols have to be transmitted over the channel.

Another termination technique is to ensure the trellis start and end states are identical. This technique is referred to as *tail biting* and has the advantage of not requiring any extra symbols to be transmitted. Tail biting is used in several popular communications standards, such as the IEEE Standard for Local and Metropolitan Area Networks - Part 16 (IEEE 802.16e) and the 3GPP Long Term Evolution (LTE) standard. This application note gives some background on the different termination techniques used in these standards, explains how to implement them using the Xilinx® Viterbi decoder LogiCORE™ module (version 6.2 or later), and includes a number of simulation reference designs to illustrate operation of the Viterbi decoder.

Viterbi Decoding

A description of the general Viterbi decoding algorithm is beyond the scope of this document. Refer to the Internet for available tutorial notes. The important concept to understand is that a trellis is constructed by computing the cost of being in each possible convolution encoder state at every symbol period. The data bit (0 or 1) most likely to have caused entry to each state is stored in a table. For example, if the encoder has a six register delay line (that is, constraint length 7), 2^6 bits are stored for each cycle. After a number of clock cycles, defined by the traceback length, the decoder traces back through the trellis, outputting the data bits for the most likely survivor path. This operation is referred to as *traceback*. Then these bits are passed through a last in, first out (LIFO) structure, so they are output in the order originally received. Usually the traceback length is set to be greater than six times the constraint length or greater than 12 times if the data is punctured.

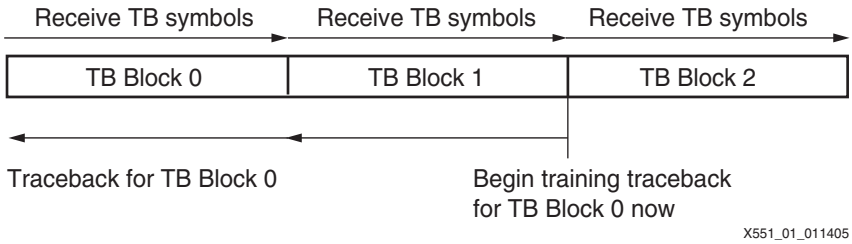

Figure 1: **Viterbi Decoder Traceback**

Figure 1 shows an example where the data bits causing entry to each state in the trellis are stored while TB Block 0 is received (TB is the traceback length). As more symbols are received, the data bits causing entry to each state in the trellis for Block 1 are stored. At this point, there are two traceback lengths' worth of data stored. The decoder now performs the traceback operation on Block 1, finding the most likely survivor path. The data bits for Block 1 are not output at this time. This operation determines the correct state to start the traceback through Block 0. In other words, Block 1 is used for training to start the traceback of Block 0 from the

correct state. In reality, the upper and lower arrows in Figure 1 are skewed in time. The traceback for a block occurs some time after the last symbol in the block is sampled. The arrows are positioned to show on which data the operation is acting. The blocks are sometimes referred to as *windows*, and the technique of processing one window at a time is referred to as the *sliding window* technique. In the same way, Block 2 is used as a training sequence to start the traceback of Block 1 from the correct state.

Viterbi Block Decoding

Convolution codes are not strictly block codes. The decoder operates on a continuous stream of incoming encoded data, splitting it into traceback lengths for processing. However, it is convenient to split the data into packets and regard each packet as a self-contained, independent block. Each packet can contain many traceback lengths of data. Generally, the encoder is forced to a known starting state and then forced to a known ending state after the last data bit is shifted in. The most frequently used methods to terminate the trellis in a Viterbi decoder are:

- Trellis truncation
- Trellis termination
- Tail biting

Trellis Truncation

Trellis truncation is the simplest of all the methods (see Figure 2). The encoder is reset to zero at the beginning of each packet. Then the data is shifted in. Nothing is done to force the encoder into a special termination state.

This method has these advantages:

- It is simple to implement.
- The code rate is not affected, that is, the ratio of original data bits to transmitted bits remains the same. If the encoder code rate is R and N bits are input, then a total of N/R bits are transmitted.

This method has these disadvantages:

- The bit error rate (BER) performance of the code is degraded because the decoder does not know from which state to start the last traceback.
- There is no data to use as a training sequence to find the correct state. The distance properties of the convolution code are lost at the end of the packet.

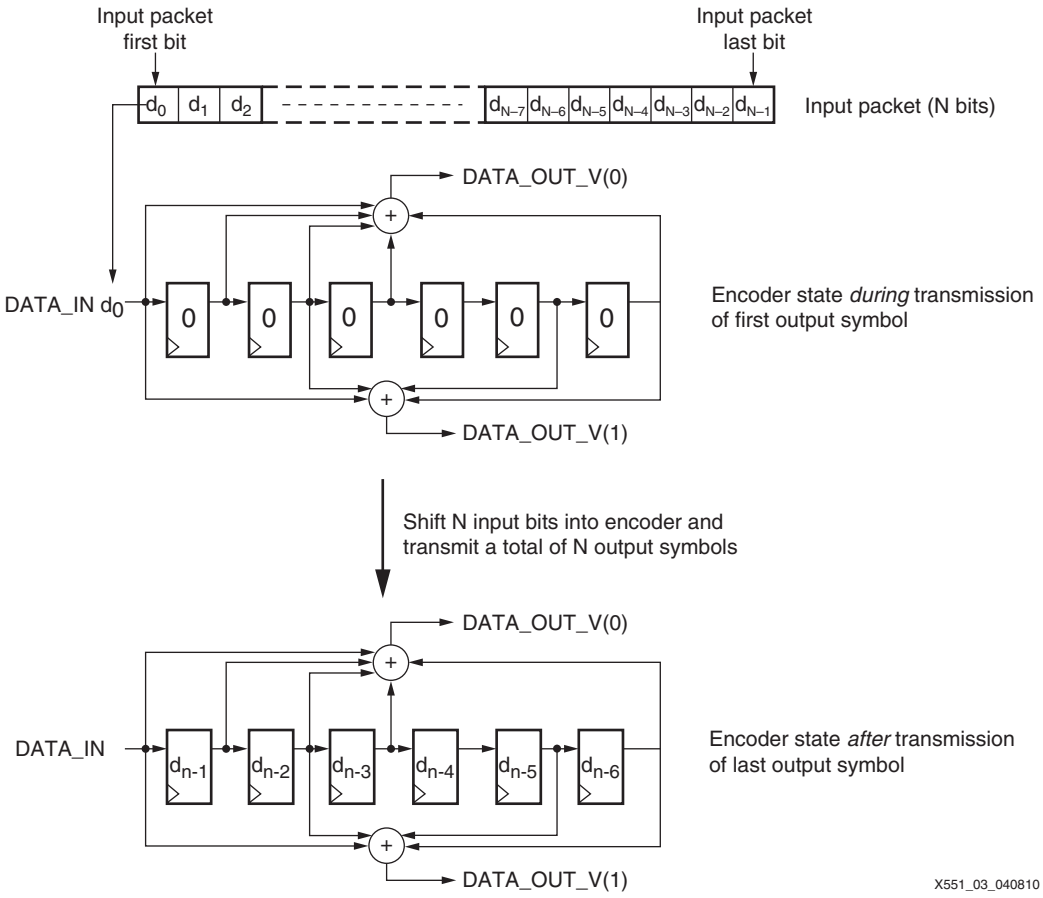

Figure 2: **Convolutional Encoding for Trellis Truncation Termination**

Trellis Termination (Tail Bits)

Trellis termination is the most common technique. The encoder is reset to zero at the beginning of each packet as in trellis truncation. This operation generally happens automatically, as the previous packet ends in state 0. After all data bits are shifted into the encoder, another Z zero bits are shifted in, where Z is the number of register stages in the encoder shift register. This resets the encoder state back to zero. The final output symbols are generated as the Zth zero is on the input to the encoder shift register. On the next clock cycle, the encoder enters state 0 and no more symbols are transmitted for this packet.

Figure 3 shows this process for a constraint length 7 example. The encoder is initialized to all zeros, and the first symbol is formed when DATA_IN = d_0. The last symbol of the packet is formed when d_{N-1} is in the right-most register and the final zero is on DATA_IN. After one more shift, the encoder is returned to the all zero state. Note that the DATA_OUT symbols are not transmitted for this state. The decoder can begin traceback for the packet in the certain knowledge that state 0 is the correct state from which to start.

In trellis termination, both the start state and end state are known by the decoder. This method has these advantages:

- It is fairly simple to implement.
- Unlike simple trellis truncation, the termination does not affect the error correction properties of the convolution code.

This method has these disadvantages:

- Extra bits have to be transmitted, reducing the code rate. If there are N bits in the packet, (N + Z)/R bits are transmitted, where Z is the number of zeros set by the constraint length - 1 and R is the encoder code rate.
- The extra bits consume additional transmission time and slightly reduce the energy-per-bit to noise-power-spectral-density ratio (Eb/No) for a given probability of error. Typically, the overall effect is insignificant except when N is very small.

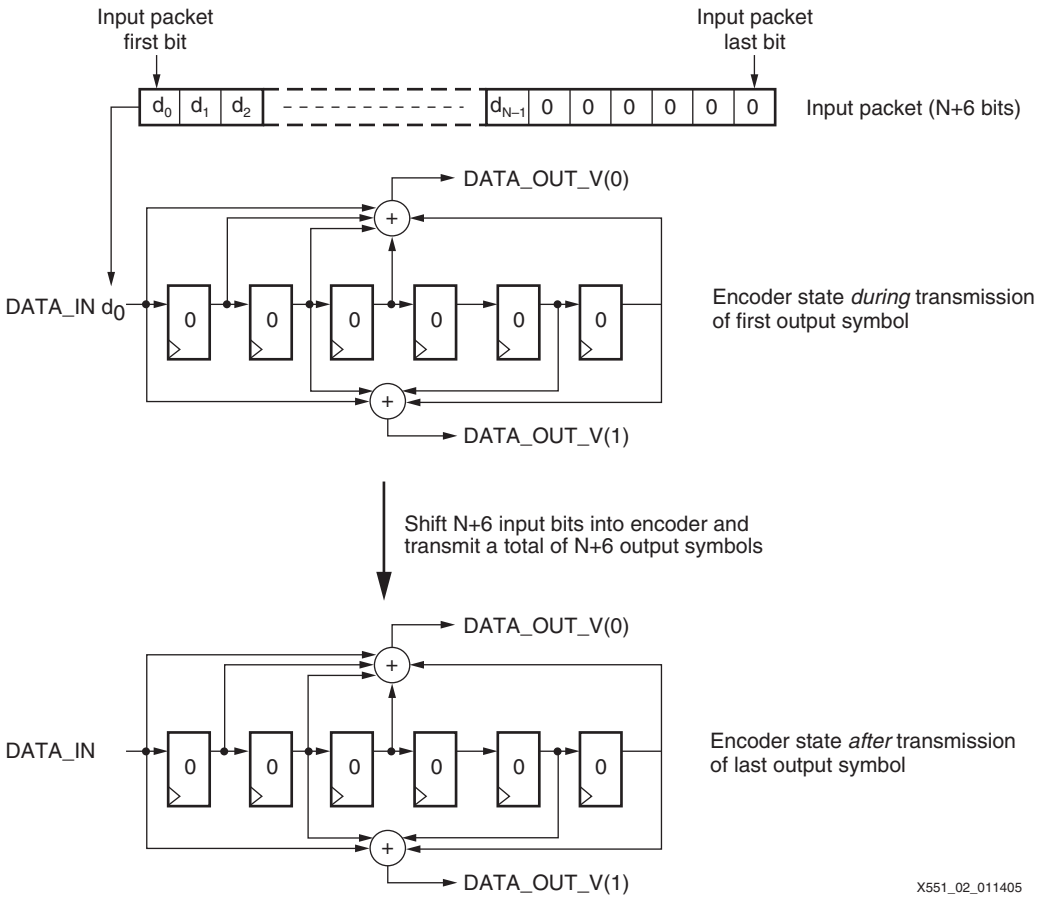

Figure 3: **Convolution Encoding for Trellis Zero Termination**

Tail Biting

Tail biting attempts to overcome the problem of transmitting extra termination bits experienced by trellis termination. The tail biting technique has these advantages:

- The code rate is not affected. N/R bits are transmitted.
- The error correction properties of the convolution code are not affected.

The disadvantages are:

- Because training is required to determine the correct start state and initial traceback state, decoding latency is increased over trellis termination.
- Receiver complexity is slightly increased.

Tail biting can be done by:

1. Using the last Z data bits of the packet to initialize the encoder shift register prior to transmission of the packet (number of bits Z = constraint length - 1). No output symbols are transmitted during encoder initialization. This means that the encoder start state and end state for the packet are identical. It also implies that the entire packet must be available at

the encoder before the first symbol is transmitted. Figure 4 illustrates this method for an example with constraint length 7. The encoder is initialized to the last six bits of the packet, and the first symbol is formed when DATA_IN = d0. The last symbol of the packet is formed when d_{N-1} is on DATA_IN. One more shift puts the encoder back into the initial state. The decoder can then begin traceback of the packet from that state.

2. Initializing the encoder with the first Z data bits of the packet—again not transmitting any output symbols during this time. The remaining (N - Z) data bits are then encoded and transmitted, followed by the first Z bits. This has the same effect of causing the start and end states to be identical. The advantage of this method is that it does not require the entire packet before encoding starts. However, the bits are out of sequence at the receiver.

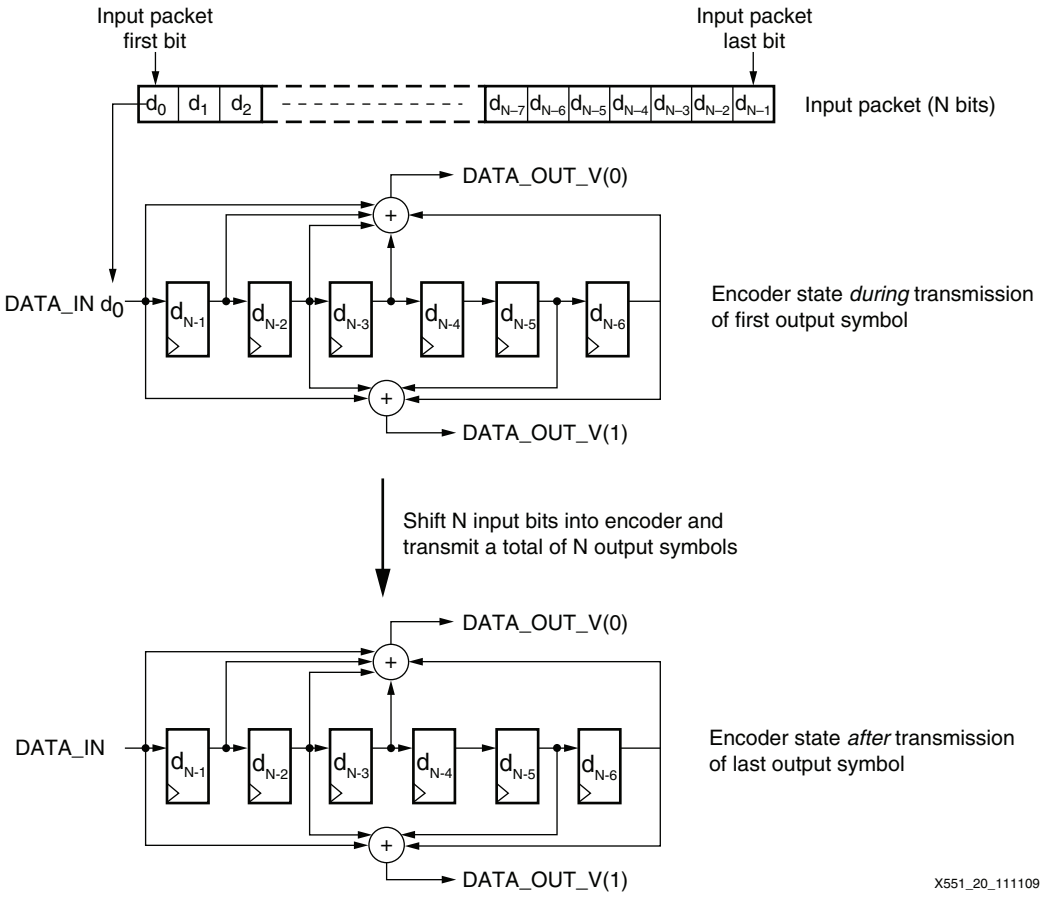

Figure 4: **Convolution Encoding for Tail Biting Termination**

Implementation

All termination methods presented can be implemented with the Convolutional Encoder [Ref 1] and Viterbi decoder [Ref 2].

Trellis Truncation Encoding

Convolutional encoding can be implemented using the Convolutional Encoder (version 6.2 or later). However, because the encoder is relatively simple, another option is to write RTL code for the encoder. In the example shown in Figure 5, a 270-bit packet is input into the convolutional encoder, resulting in 270 symbols. Each symbol is 2 bits for a 1/2 code rate.

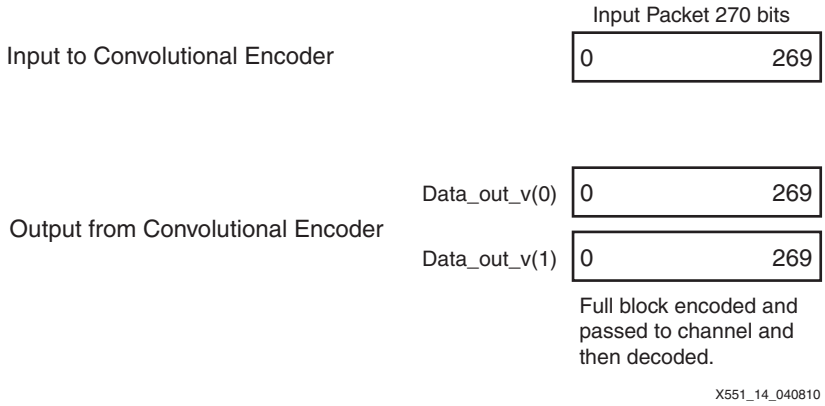

Figure 5: **Convolutional Encoding with Trellis Truncation**

Trellis Termination Encoding

Convolutional encoding can be implemented using either the Convolutional Encoder IP, or inferred from RTL code. In the example shown in Figure 6, the traceback length is 54 for a constraint length equal to 9. The packet length is 108 bits, and 8 zeros are appended to the original packet (constraint length – 1). In Figure 6, the code rate is 1/3, and the number of zeros appended is 8.

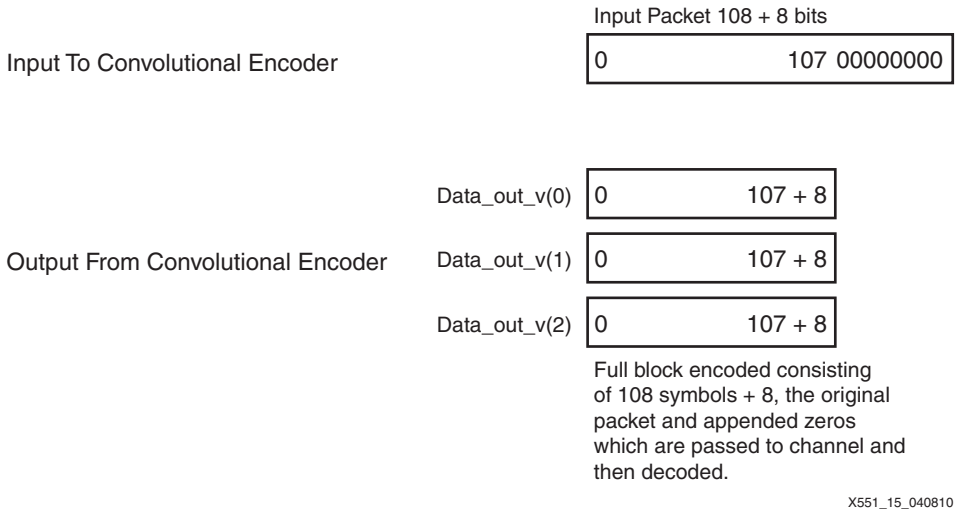

Figure 6: **Convolutional Encoding with Trellis Termination**

The output from the encoder consumes a total of 108 + 8 cycles. This method can be used with any length packet. A number of zeros (constraint length - 1) still needs to be added, even if the packet length is less than the traceback length.

Tail Biting Encoding

In an example where the constraint length is 7 and the packet length is 270 bits, a circular buffer stores the packet being transmitted. The last Z bits of the packet are loaded first, followed by the whole packet. However, the first Z bits are ignored and are not required for transmitting. Thus, the enable for the encoded data is 270 cycles. In the example shown in Figure 7, the circular buffer can be implemented using block RAM. As outlined in Trellis Termination Encoding, another approach is to add the first Z bits of the packet prior to encoding.

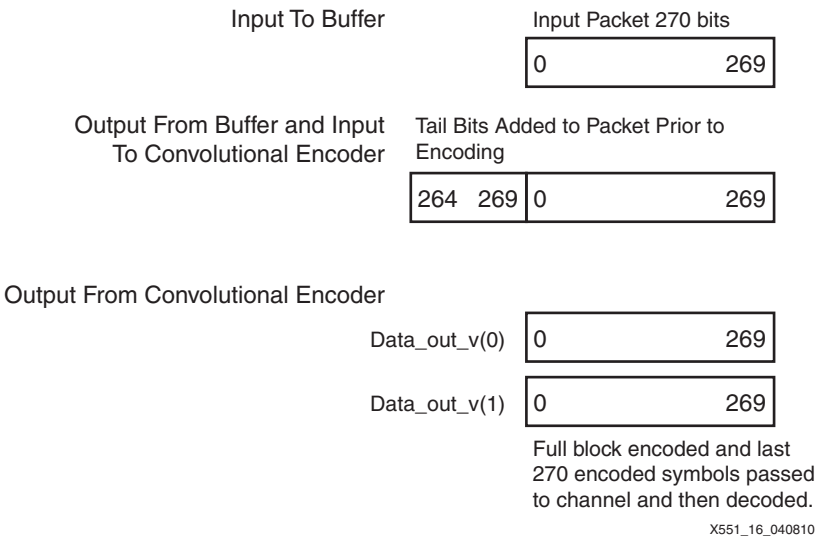

Figure 7: **Convolutional Encoding with Tail Bits**

Channel Model

A noisy channel model can be implemented using the Additive White Gaussian Noise (AWGN) core. See Additive White Gaussian Noise (AWGN) Core v1.0 [Ref 3] for details on testing the encoder/decoder combination.

Trellis Truncation Decoding Implementation

The Viterbi decoder decodes in the normal way, using a sliding window, as shown in Figure 1, page 1. The trellis construction should start from state 0. The decoder can be forced to give state 0 the lowest cost at the start of trellis construction (see Trellis Termination Decoder Implementation, page 9 for details).

Dummy data or data for the next packet is clocked in after the end of the packet to provide a training sequence for the last block. This training sequence causes the traceback for the last block to start in a state that might not be the correct state.

One alternative is to not bother with a training sequence for the last block and start the traceback immediately from the state with the lowest cost. The core can be configured in this way by selecting *Direct Traceback* in the core GUI. Select the *Best State* option to ensure the traceback begins from the lowest cost state. This option gives the best chance for correct decoding, however, the results are not as good as when a full training sequence is available. If traceback is started from the wrong state, some BER degradation occurs at the end of a packet. Note that the Best State decoding logic increases the core size.

When using the Direct Traceback technique (see Figure 8), the TB_BLOCK input is used to signal the position of the final TB block of data within a packet. This block is traced back without a prior training sequence to determine the traceback start state. Traceback of the final TB block begins immediately after TB_BLOCK is deasserted.

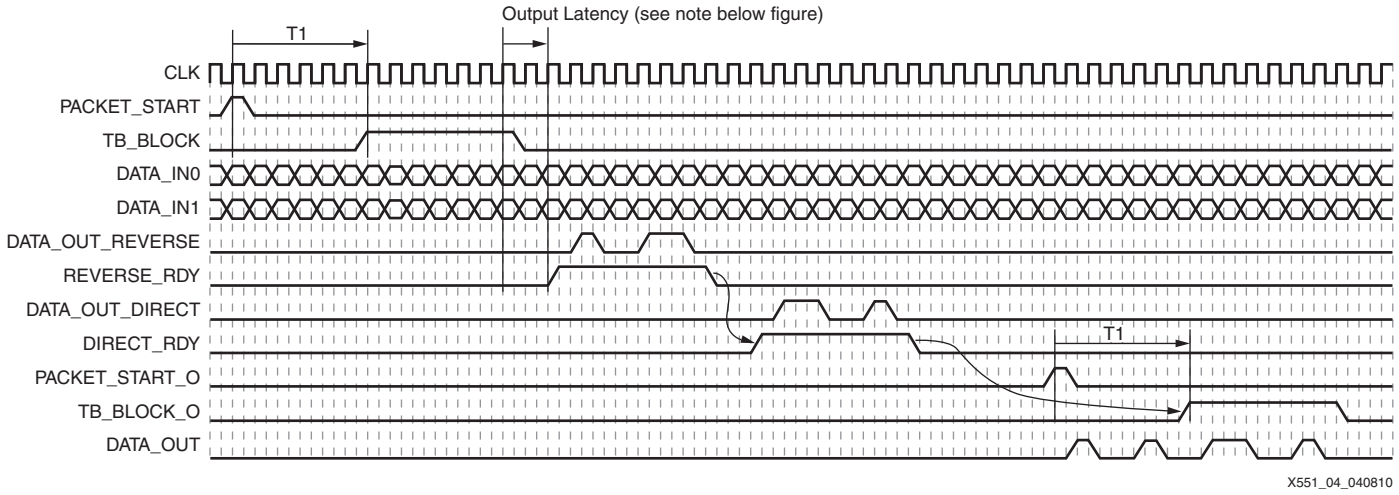

Figure 8: **Packet Handling**

Note relevant to Figure 8:

1. Output latency is shown as two clock cycles for clarity. Actual latency = (8 + output rate) clock cycles.

In this example, PACKET_START is used to signal the start of the packet. PACKET_START is used to force the trellis construction to start from a state other than the one the previous block ended in, as is the case with trellis truncation. See Trellis Termination Decoder Implementation, page 9 for more details on PACKET_START.

TB_BLOCK can be asserted at any time. The T1 delay in Figure 8 can be any number of clock cycles and does not have to be an integer number of traceback lengths. The T1 delay is 0 clock cycles when there is only one traceback block in the packet. Typically the TB_BLOCK pulse has a duration of the number of remaining clock cycles when the packet length is divided by the traceback length. If this remainder is zero, then TB_BLOCK should be the same number of cycles as the traceback length. The TB_BLOCK pulse can be any duration, within the limits defined in the Viterbi decoder data sheet [Ref 2]. It does not matter if the pulse is less than the normally required traceback length, because a training sequence is not used to define the start state of the traceback in this case. TB_BLOCK is used in the same way with trellis termination and tail biting.

The decoder continues to output decoded data with the normal latency on DATA_OUT. If Direct Traceback is enabled, the decoder also outputs the data for the last block of the packet on DATA_OUT_REVERSE. This last block of data is output a small number of clock cycles after TB_BLOCK is deasserted, and provides data with the lowest possible latency. This is useful if there is only a single block in a packet because DATA_OUT can be ignored and output data can be taken from DATA_OUT_REVERSE or DATA_OUT_DIRECT.

Overall latency can be reduced using DATA_OUT_REVERSE even if there are several blocks in a packet. In this case, DATA_OUT_REVERSE can be buffered in the appropriate locations of a dual-port RAM while the data for the earlier blocks (on DATA_OUT) is being written to the same RAM. This means that all decoding has completed as soon as the last symbol for the second last block on DATA_OUT is written to the RAM. Thus the overall latency can be reduced by TB cycles.

Because data on DATA_OUT_REVERSE is not passed through the LIFO, it comes out in reverse order. It is immediately followed by the correctly ordered data on DATA_OUT_DIRECT. In some applications, it might not matter that the data comes out in reverse and DATA_OUT_REVERSE can be used to give extremely low latency. The data for the final block still comes out on DATA_OUT after the normal latency, indicated by TB_BLOCK_O. This data is identical to the final block data output earlier on DATA_OUT_DIRECT. If latency of the final

block is not an issue, then it is simplest to always read the decoded data from the DATA_OUT port and ignore DATA_OUT_REVERSE and DATA_OUT_DIRECT.

Trellis Termination Decoder Implementation

The packet starts and finishes in state 0. It is desirable to force the decoder to begin the trellis construction from state 0. Nothing special needs to be done if the previous packet received by the decoder left state 0 as the most likely state. However, this cannot be guaranteed. Figure 9 shows the BER degradation that occurs when the trellis construction relies on the previous packet ending in state 0 versus *forcing* the new packet to start from state 0. This constraint length 7 example has 32-symbol packets and uses Direct Traceback. The degradation at lower signal-to-noise ratios (SNRs) is approximately 0.5 dB.

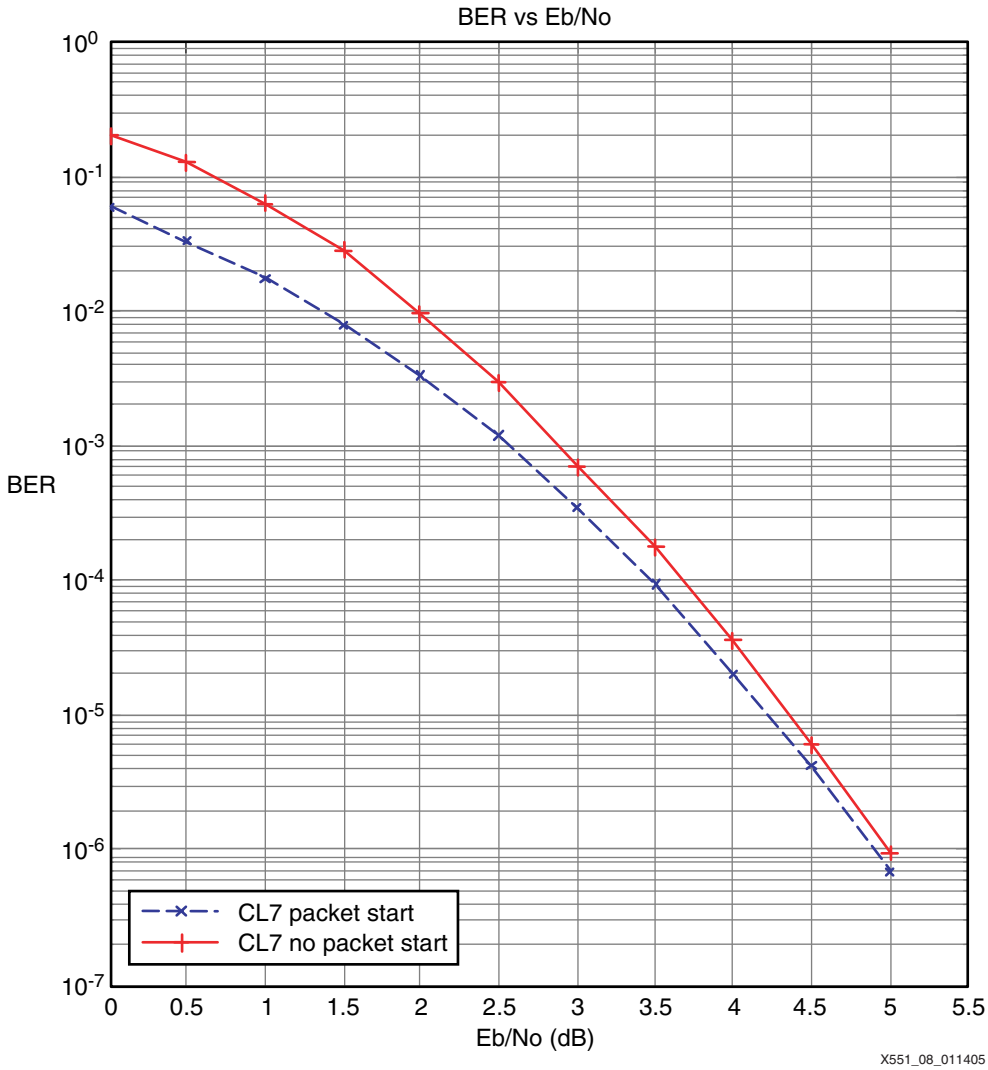

Figure 9: **Effect of Incorrect Trellis Construction in Trellis Termination**

Trellis Construction

The decoder can be forced to begin trellis construction from state 0 in one of two ways:

1. Feed in a number of strong zeros prior to the first real data in the packet. A minimum of Z zero symbols needs to be sampled to force state 0 to have the lowest cost in the decoder as illustrated in Figure 10. It does not matter where the decoder is in a traceback block when the zeros are fed in. The zeros between the packets still force the decoder to the correct state prior to the start of the real packet data. When these zeros are sampled, the

core's BLOCK_IN input can be asserted. BLOCK_OUT then goes to a logic High when the data corresponding to the input zeros is output. This signal can be used as a flag to ignore the output data. BLOCK_OUT is simply BLOCK_IN delayed by the decoding latency.

It is not necessary to insert zeros prior to the first packet after a reset as construction of the first trellis always begins from state 0.

One advantage of this method is that it also takes care of forcing the traceback to state 0 at the end of a packet. However, there are other ways of doing this task, as described in Traceback Start.

2. Use the PACKET_START input to force the trellis construction to start at state 0 or any other state as defined by the PS_STATE input. This input forces the cost for the start state to be very low and the cost of all the other states to be very high. The advantage of this method is that it does not waste any clock cycles while dummy zeros are sampled, as in Method 1. It also avoids the awkward timing and control logic that might be required to insert dummy zeros. Refer to Figure 8, page 8 for the PACKET_START timing.

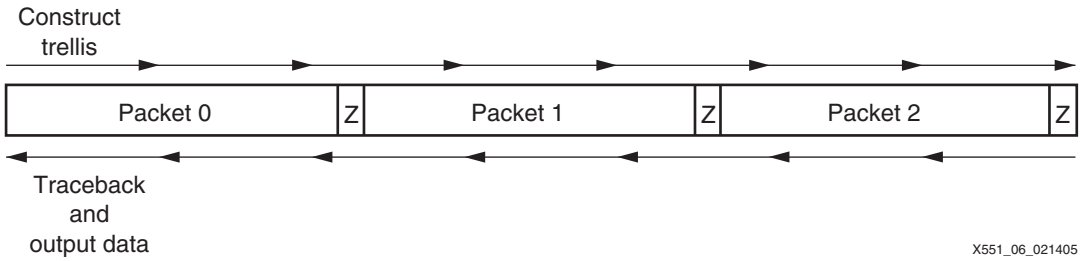

Figure 10: **Forcing Trellis Construction Start State to Zero by Zero Insertion**

As mentioned at the beginning of this section, it is possible to decode normally without forcing state 0 to have the lowest cost at the start of a packet. This might cause a BER degradation due to the reduced likelihood of correcting an error at the start of a packet. The degradation is noticeable if the channel is noisy when the tail bits for the previous packet are received and contain errors. Then it is likely that state 0 does not have a significantly lower cost than the other states as trellis construction for the next packet begins.

Traceback Start

Having taken care of the zero forcing at the start of trellis construction, the user must ensure that the traceback begins at state 0. The decoder begins traceback from the state with the lowest cost at the end of the training sequence traceback. If this state can be guaranteed to be zero, then nothing needs to be done. This will be the case when the next packet begins from state 0 and is decoded without error, because the first traceback block of the next packet forms the training sequence for the last traceback block of the current packet. For example, if the last block of Packet 0 in Figure 10 is being decoded, the first block of Packet 1 is used as the training sequence. If the traceback through this first block ends in state 0, then the starting state for the traceback through the last block of Packet 0 will be state 0. Typically this sequence cannot be guaranteed and the traceback must be forced to start from state 0.

Figure 11 shows the BER degradation that occurs when the traceback relies on the next packet trellis starting in state 0 versus *forcing* the traceback to start from state 0. This constraint length 7 example has 96-symbol packets. The degradation at lower SNRs is approximately 0.25 dB.

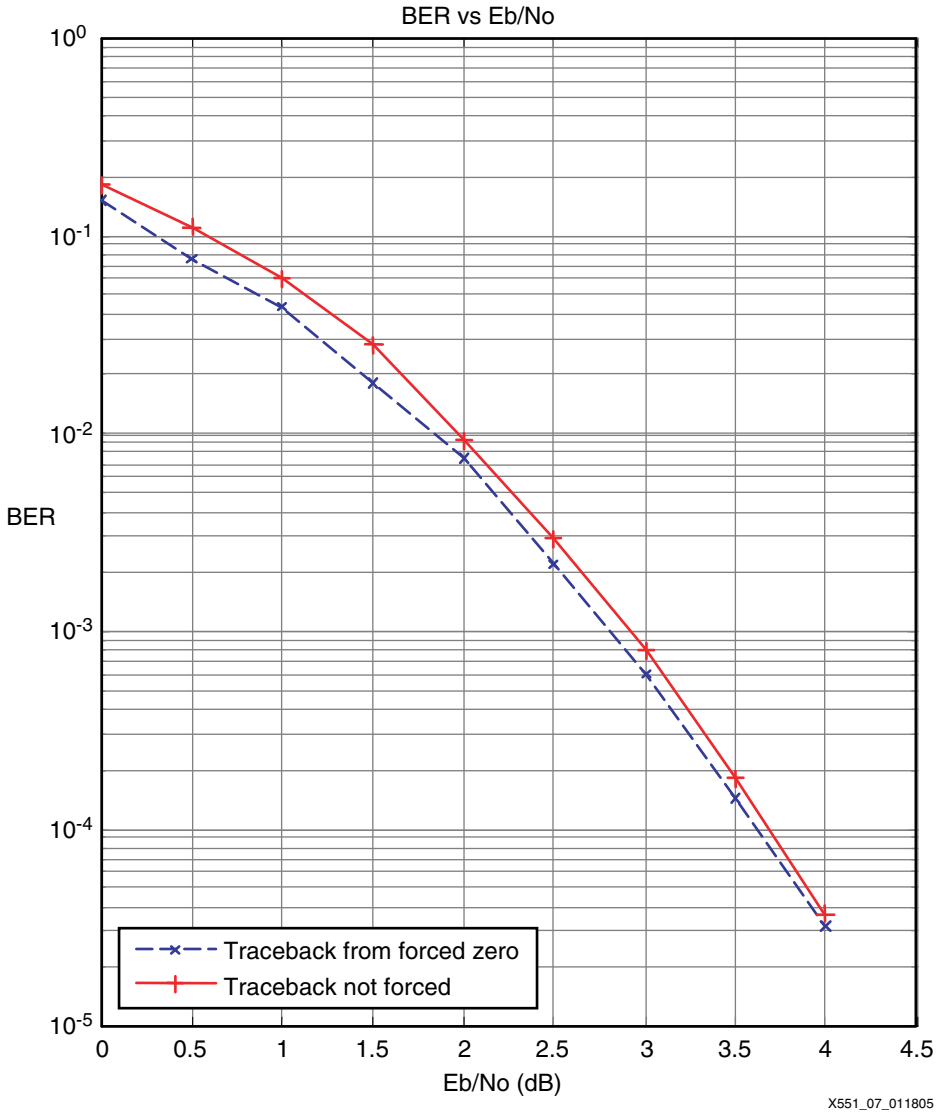

Figure 11: **Effect of Incorrect Traceback Start State in Trellis Termination**

The traceback of a packet can be forced to start from state 0 in one of two ways:

1. Strong zeros can be inserted between packets in exactly the same way as in the trellis construction method described in Method 1 on page 9. Again, as long as Z or more zeros are inserted, traceback can be guaranteed to begin from state 0. As shown in Figure 10, the zeros do not have to align with the start of the traceback block. All that matters is that they immediately follow a packet.

   This technique assumes that the dummy zeros are immediately followed by the next packet. If this is the last packet and there is no more data, more dummy input symbols (any value) should be sampled to flush out the decoder traceback pipeline.

2. Direct Traceback from state 0 can be used for the last block in the packet.

   If latency is important, then the processing delay on the last block can be reduced by selecting this feature. This does away with the training sequence for the last block and forces the traceback to begin from state 0, saving traceback length clock cycles.

   This timing is described in Trellis Truncation Decoding Implementation, page 7 and is shown in Figure 8, page 8. The Direct Traceback is identical in this case, with the exception that we are tracing back from state 0 rather than the Best State.

For punctured data, the best possible BER results are obtained if the training sequences for all the other blocks in the packet are started from the Best State. See the Viterbi decoder data sheet [Ref 2] for more details on when to use the Best State logic. It is possible to trace back from the best state for these blocks while still tracing back from state 0 for the last block.

There is no BER degradation with Direct Traceback compared to using a training sequence, because we know for certain that the traceback starts from state 0.

It is also possible to perform Direct Traceback from a non-zero, user-defined state. In this case, the state value is sampled on the TB_STATE input at the same time as the last data symbol of the last TB block is sampled.

Figure 12 shows an example with a single small packet, as can be found in the IEEE 802.11a standard. In this example, the packet contains only a single traceback length, and the PACKET_START signal is used to force the trellis construction to begin from state 0. As mentioned above, this can also be achieved by padding with strong zeros prior to the packet. While TB_BLOCK is High, 24 symbols are sampled. When TB_BLOCK is deasserted, the decoder begins traceback from state 0 and outputs the packet data in reverse order on DATA_OUT_REVERSE. This standard requires that the 24-bit header packet is decoded as quickly as possible. Using DATA_OUT_REVERSE is the fastest way to meet this requirement. Data follows in non-reversed order on DATA_OUT_DIRECT and DATA_OUT, as described in the example in Figure 8, page 8.

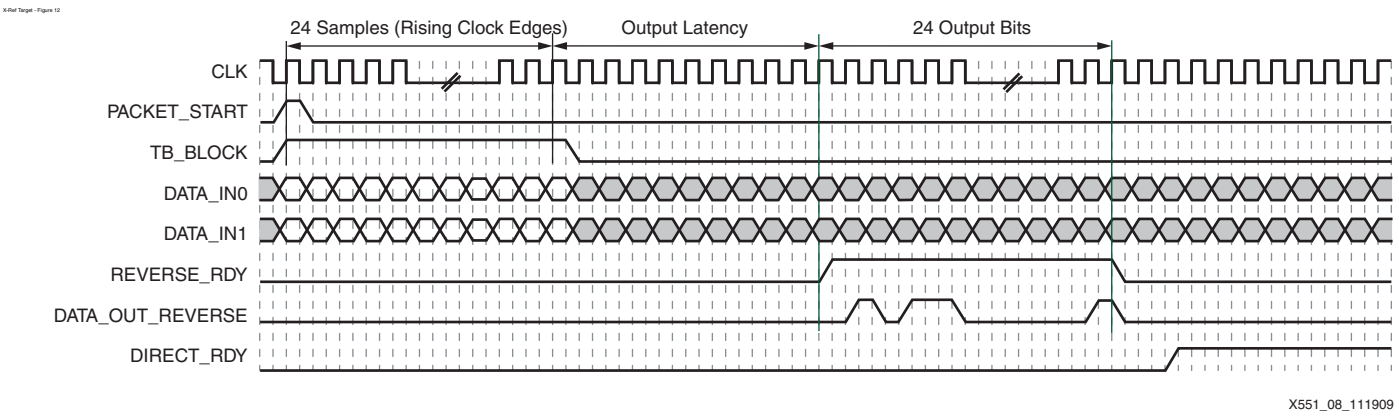

Figure 12: **Trellis Termination with a Single Small Packet**

Tail Biting Decoder Implementation

The implementation of tail biting can depend upon the length of the traceback length. There are three possible scenarios:

- The packet contains multiple traceback lengths.
- The packet contains a single traceback length.
- The packet is less than the traceback length.

In most tail biting applications, the encoder state is initialized to the last Z bits of the data packet prior to any transmission. Thus the encoder has the same starting and ending states for a packet. For optimal decoding, the decoder needs to start the trellis construction from this state. If not done, some BER degradation occurs due to the reduced likelihood of correcting an error at the start of a packet. The IEEE 802.16d standard uses this form of tail biting. In the LTE Uplink Channel Decoder [Ref 4], some of the tail biting techniques discussed are implemented for the Physical Downlink Control Channel.

Packet Contains Multiple Traceback Lengths

This section describes three techniques for decoding packets that contain multiple traceback lengths: decoding using the first and second traceback blocks, decoding using the last traceback block first, and decoding for tail biting using the first traceback block first.

Decoding Using First and Second Traceback Blocks

If there are several traceback lengths within a packet, one technique is to decode the first TB block at the end of the packet. Assuming the packet terminates in the correct state, then the trellis construction for TB block 0 begins in the correct start state, as illustrated in Figure 13. This example assumes a packet contains (N + 1) traceback lengths.

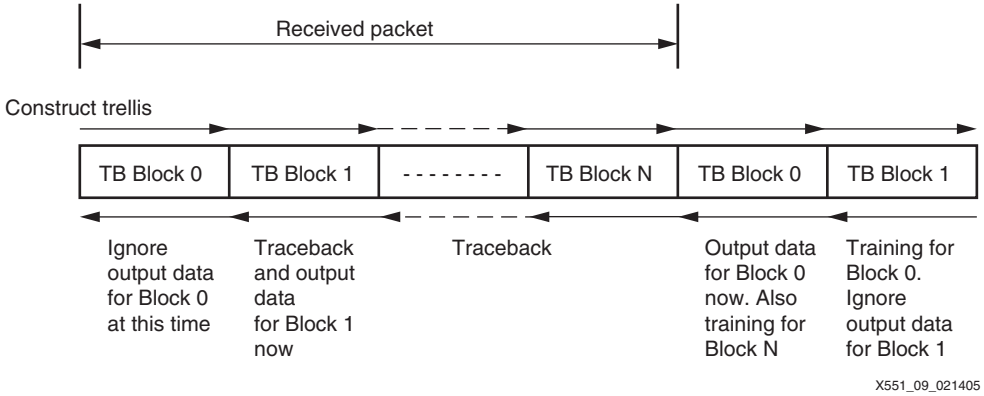

Figure 13: **Decoding for Tail Bits Using First and Second Traceback Blocks**

TB Block 0 is used at the start of the packet only to ensure the trellis construction for TB Block 1 starts from the correct state. Ignore the output data for TB Block 0 at this time because it might be incorrect due to not knowing from which state to begin the trellis construction. There is no point using the PACKET_START input. If we did know from which state to begin trellis construction, we would use PACKET_START, input the state on PS_STATE, and use the data output for TB Block 0 at this time without re-inserting TB Block 0 at the end of the packet. The advantages are:

- The re-insertion of Block 0 at the end also provides the correct training sequence to start the traceback of Block N from the correct state.
- This method clearly adds extra decoding latency for Block 0.
- The blocks other than the start and end blocks are decoded normally. The end state of one block automatically provides the start state for the next.

The disadvantages are:

- This method adds extra decoding latency (determined by the packet length) for Block 0.
- The end state of one block automatically provides the start state for the next. Thus, some buffering is required.

Decoding Using Last Traceback Block First

Another technique is to input TB Block N first, ignoring the output data (see Figure 14). This technique gives the correct start state for the trellis construction of TB Block 0. This block is followed by blocks 0 to N. TB Block 0 is input again at the end to provide a training sequence for decoding TB Block N. The advantages of this technique are:

- All the data is output from the decoder in the correct order.
- There is no delay for the first traceback block.

The disadvantage of this technique is:

- The entire packet must be received before decoding can begin.

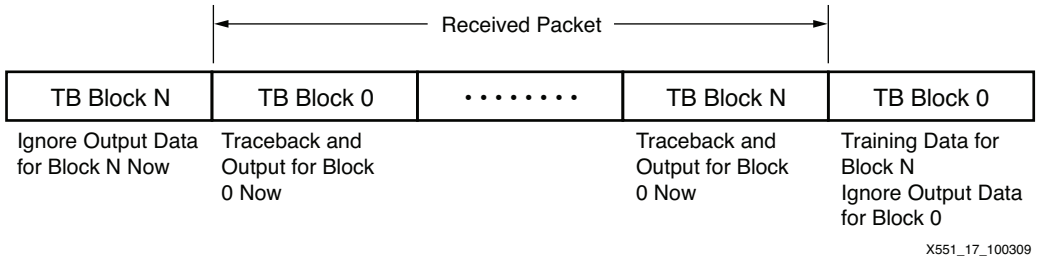

Figure 14: **Decoding for Tail Biting Using Last Traceback Blocks**

Decoding for Tail Biting Using First Traceback Block First

The core's Direct Traceback feature can be used to perform the decoding without the need to re-insert TB Block 1 at the end, as in Figure 13. This technique is shown in Figure 15.

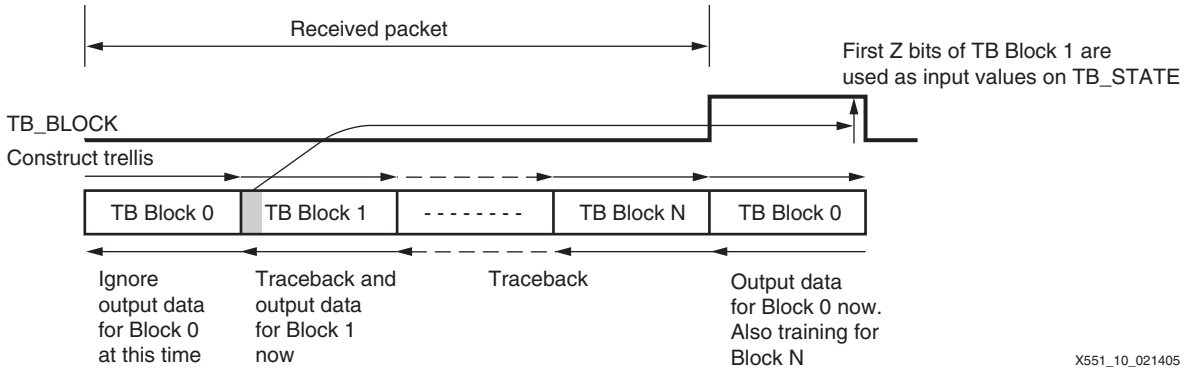

Figure 15: **Decoding for Tail Biting using Direct Traceback**

In this case, TB Block 0 is still used initially to obtain the correct starting state for the trellis construction during TB Block 1 and the first output data for TB Block 0 is ignored. When the data for TB Block 1 appears on DATA_OUT, the first Z bits can be saved in a register, giving the correct traceback start state to begin the decoding of TB Block 0. Thus TB Block 1 does not need to be re-inserted at the end of the packet to determine the start state for the TB Block 0 traceback. Simply apply the saved start state on the TB_STATE input at the same time that the last symbol of TB BLOCK 0 is sampled on DATA_IN (the timing is shown in Figure 16). In this example, the constraint length is 7 and Z is 6. The first six bits of TB Block 1 are decoded as `110000` binary, giving the state the encoder shift register was in immediately after TB Block 0 and hence the correct state with which to begin the Direct Traceback of TB Block 0. Input a value of 3 (`000011` binary) on TB_STATE at the end of the TB_BLOCK pulse. The traceback of TB Block 0 then begins from the correct state (3) without requiring a training sequence. Not all clock cycles are shown in Figure 16. In reality, the blocks and latencies are longer than shown.

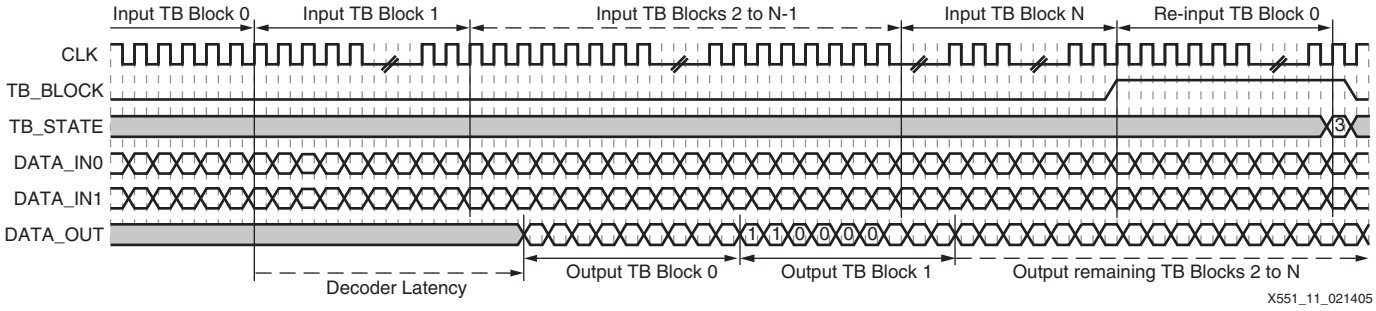

Figure 16: **Timing for Direct Traceback Tail Biting Technique**

This technique relies on being able to decode the first Z bits of TB Block 1 before the state value is required for the TB_STATE input. There might not be sufficient time to perform this operation if there is only a small number of TB Blocks in a packet. In this case, the method shown in Figure 13, page 13 can be used.

Packet Contains Single Traceback Length

If the packet contains only a single traceback length, then the technique is effectively the same as passing Block 0 through the decoder three times: once to ensure the correct start state for the trellis construction, once to construct the trellis, and once to perform the correct training so the traceback begins from the correct state (see Figure 17). The Direct Traceback technique cannot be used in this case, because the traceback start state cannot be obtained to apply to TB_STATE.

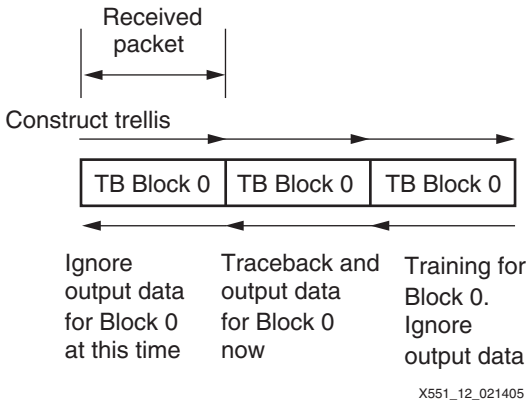

Figure 17: **Single Block Packet**

Block 0 *must* be passed through three times. If it passes through only twice, BER degradation is substantial, resulting in almost all packets decoding incorrectly. Figure 18 shows a small 32-symbol, constraint length 3 example. The curves for tail biting, implemented by passing the block through the decoder three times and a standard continuous stream (non-tail biting) decoder are almost identical. The curve obtained when the block is only passed through twice shows considerably worse BER performance. For larger constraint lengths, this performance is even worse because the likelihood of choosing the correct starting state by chance is reduced.

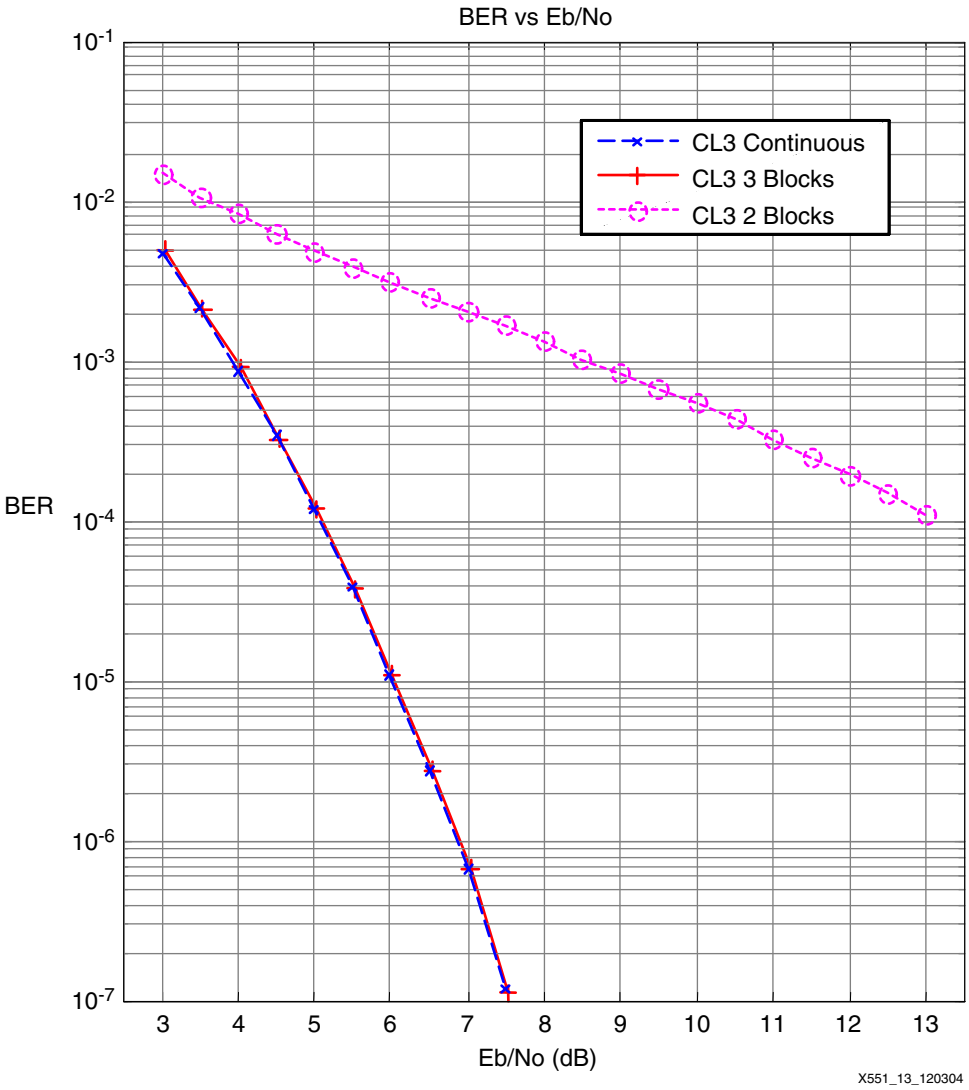

Figure 18: **Effect of Incorrect Trellis Construction in Tail Biting**

Packet Contains Less Than One Traceback Length

In this case the packet of data is less than the traceback length. Thus, the decoder does not have enough data to construct the trellis, start at the correct point in the trellis, and perform the correct training. Therefore, copies of the packet data are repeated before and after the actual packet block. The repeated packet data is traceback length to allow construction and training of the data.

In the example shown in Figure 19, the packet length is 27 bits and the traceback length is 42 bits. Because tail biting is being performed, the last Z bits of the packet are used to initialize the decoder. A traceback-length block of data is the first input to the Viterbi decoder from the circular buffer. The block consists of the last 15 bits of the packet followed by the 27 packet bits. The 27-bit block being decoded is then input, indicated by assertion of the BLOCK_IN signal, followed by another 27-bit block and the first 15 bits of the next block, which is used for traceback training. The Viterbi decoder outputs the 27-bit packet indicated by the RDY signal and the BLOCK_OUT signal.

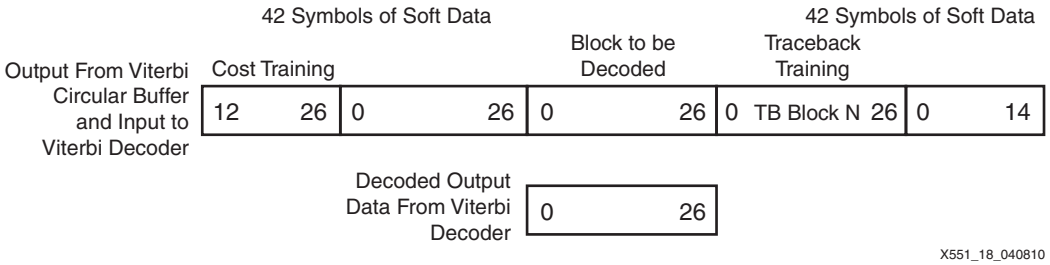

Figure 19: **Decoding for Tail Bits for Small Packet Lengths**

Refer to Tail Biting Example in the design section.

Recommendations

When implementing a termination system, these recommendations should be kept in mind:

- Use trellis termination or tail biting methods.
- Only use tail biting if it is specified in the standard.
- Use direct traceback if latency is important.
- Set the soft width of the Viterbi decoder to four.
- Set the traceback length to 6 times the constraint length for normal data and 12 times for punctured data.

Reference Designs

The reference design files can be downloaded at https://secure.xilinx.com/webreg/clickthrough.do?cid=146331. Table 1 shows the reference design checklist for this application note.

Table 1: **Reference Design Checklist**

Parameter	Description
General	
Target devices (stepping level, ES, production, speed grades)	Virtex®-6 FPGAs
Source code provided	Y
Source code format	VHDL
Design uses IP from CORE Generator software	Y
Simulation	
Functional simulation performed	Y
Timing simulation performed	N
Testbench used for functional and timing simulations	Y
Testbench format	VHDL
Simulator software/version used	ModelSim 6.5c
SPICE/IBIS simulations	N
Implementation	
Synthesis software tools/version used	XST 12.1
Implementation software tools/versions used	ISE 12.1
Static timing analysis performed	Y

Table 1: **Reference Design Checklist *(Cont'd)***

Parameter	Description
Hardware Verification	
Hardware verified	N
Hardware platform used for verification	NA

The `xapp551.zip` file associated with this application note contains three termination scheme examples: tail biting, trellis termination training sequence, and trellis termination direct traceback. Each example contains an XISE file that can be opened with the Xilinx ISE® design suite (version 11.3 or later), a Viterbi decoder implementation, testbench files, and DO scripts for ModelSim simulations. The example files are not synthesizable, but they illustrate how these techniques aid the development of hardware solutions.

The examples share a similar format, which is outlined in Table 2, Table 3, and Table 4. Also provided is a data source, which can be put into packets and sent through a convolutional encoder. A channel model is used to add noise to the encoded signal before decoding. The decoded signal is then compared with the delayed version of the data source.

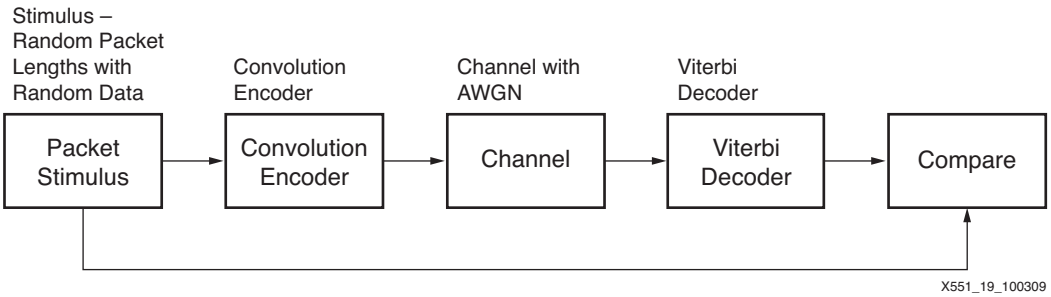

Figure 20: **Testbench Configuration**

Tail Biting Example

This example shows tail biting techniques. The example files are described in Table 2.

Table 2: **Description of Files for Tail Biting Example**

Directory	File Name	Description
`tail_biting/sim/test`	`tb.vhd`	This file instantiates the blocks shown in Figure 20, but for tail biting. It shows how the testbench processes the input packet through the various buffering, encoding, and decoding stages.
`tail_biting/sim/globals`	`fifo_wrap.vhd`	This file is the wrapper for the FIFO in the CORE Generator™ software.
	`glb_dpm.vhd`	This file is the wrapper for the dual port block RAM in the CORE Generator software.
	`global_types_pkg.vhd`	This file contains global type and constant definitions.
`tail_biting/sim/enc`	`circ_buffer_enc.vhd`	This file contains the circular buffer used for reading the six tail biting bits.
	`conv_enc.vhd`	This file contains the convolutional encoder with circular buffer instantiation. This is a constraint length 7 encoder with a rate of 1/3.
`tail_biting/sim/channel`	`awgn_gen.vhd`	This file contains the behavioral model for the AWGN channel.

Table 2: **Description of Files for Tail Biting Example *(Cont'd)***

Directory	File Name	Description
tail_biting/sim/dec	circ_buffer_dec.vhd	This file contains the circular buffer, which is used for generating the training sequence that is wrapped around the Viterbi packet to be decoded.
	vit_dec.vhd	This file contains the top level of the decoder. It instantiates the circular buffer and Viterbi decoder. It handles block valid and data flow signals.
tail_biting/	vit.do	This file is the simulation script for use with ModelSim.
	wave_vit.do	This file is the simulation script for use with ModelSim that puts the relevant signals in the wave window.
	tail_biting.xise	This file contains the ISE software project. To run the implementation, open tail_biting.xise with the ISE tool and run the flow.

To run the simulation, the vit.do file is executed. This file compiles the design files, opens a wave window, and simulates the design. The packet size can be modified by changing these lines in the tb.vhd file:

```
constant PACKET_ULIMIT : positive := 100; -- upper limit on packet size
constant PACKET_LLIMIT : positive := 20;  -- lower limit on packet size
```

Trellis Termination Training Sequence Example

This example illustrates the Viterbi decoder used in a trellis termination scheme. To run the vit.do file in the ISE software project, select simulation and double-click.

The example files are described in Table 3.

Table 3: **Description of Files for Trellis Termination Training Sequence Example**

Directory	File Name	Description
training_seq/sim/test	tb.vhd	This file instantiates the blocks shown Figure 20, but for trellis termination. It shows how the testbench processes the input packet through the various buffering, encoding, and decoding stages.
training_seq/sim/globals	conversion_pkg.vhd	This file is the package file for converting signals.
training_seq/sim/enc	conv_enc.vhd	This file contains the convolutional encoder with circular buffer instantiation. This is a constraint length 7 encoder with rate 1/2.
training_seq/channel	awgn_gen.vhd	This file contains the behavioral model for the AWGN channel.
training_seq/	training_seq.do	This file is the simulation script for use with ModelSim.
	wave_vit.do	This file is the simulation script for use with ModelSim that puts the relevant signals in the wave window.
	training_seq.xise	This file contains the ISE software project. To run the implementation, open training_seq.xise with the ISE tool and run the flow.

To run the simulation, open the ISE software project and run the behavioral simulation. This file compiles the design files, opens a wave window, and simulates the design. The packet size can be modified by changing these lines in the tb.vhd file.

```
-- patterned or random input
  constant RANDOM_INPUT : boolean := TRUE;
  -- limit size of random input packets >= 1
  constant PACKET_LIMIT : positive := 40; -- can be set to any value
```

Trellis Termination Direct Traceback Example

This example illustrates the trellis termination using direct traceback. The example files are described in Table 4.

Table 4: **Description of Files for Trellis Termination Direct Traceback Example**

Directory	File Name	Description
direct_traceback/sim/test	tb.vhd	This file instantiates the blocks shown in Figure 20, but for direct traceback. It shows how the testbench processes the input packet through the various buffering, encoding, and decoding stages.
direct_traceback/sim/globals	conversion_pkg.vhd	This file is the package file for converting signals.
direct_traceback/enc	conv_enc.vhd	This file contains the convolutional encoder with circular buffer instantiation. This is a constraint length 9 encoder with rate 1/2.
direct_traceback/sim/channel	awgn_gen.vhd	This file contains the behavioral model for the AWGN channel.
direct_traceback/	direct_traceback.do	This file is the simulation script for use with ModelSim.
	wave_vit.do	This file is the simulation script for use with ModelSim that puts the relevant signals in the wave window.
	direct_traceback.xise	This file contains the ISE software project. To run the implementation, open direct_traceback.xise with the ISE tool and run the flow.

The simulation is run by opening the ISE software project and running the behavioral simulation.

Conclusion

In the latest communication standards, some control information is required to be sent via packets. These packets can vary in length. This application note describes how, using trellis termination and tail biting techniques, these packets can be transmitted and decoded successfully using the Xilinx Viterbi decoder.

References

These product pages provide additional information and links useful to this application note:

1. Convolutional Encoder
   http://www.xilinx.com/products/ipcenter/Convolutional_Encoder.htm
2. Viterbi Decoder
   http://www.xilinx.com/products/ipcenter/Viterbi_Decoder.htm
3. Additive White Gaussian Noise (AWGN) Core
   http://www.xilinx.com/products/ipcenter/DO-DI-AWGN.htm
4. 3GPP LTE UL Channel Decoder
   http://www.xilinx.com/products/ipcenter/DO-DI-CHDEC-LTE.htm

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
02/14/05	1.0	Initial Xilinx release.
07/30/10	2.0	Added Figure 2, Figure 5, Figure 6, Figure 7, Figure 14, Figure 19, and Figure 20. Added Trellis Truncation Encoding, Trellis Termination Encoding, Tail Biting Encoding, Channel Model, Traceback Start, Packet Contains Multiple Traceback Lengths, Packet Contains Single Traceback Length, Packet Contains Less Than One Traceback Length, Recommendations, Reference Designs, and References sections. Clarified that reference designs are simulation reference designs in Summary. Updated traceback length setting for data punctures in Viterbi Decoding. Reorganized text in Tail Biting. Added possible scenarios to the beginning of Tail Biting Decoder Implementation. Added Table 1.

Notice of Disclaimer