Lattice Semiconductor Corporation

LatticeCORE™

Block Viterbi Decoder User’s Guide

June 2010
IPUG32_02.7

Table of Contents

Chapter 1:

Introduction

The Block Viterbi Decoder IP core is a parameterizable Viterbi Decoder for decoding different combinations of convolutionally encoded sequences. The decoder supports various code rates, constraint lengths, and generator polynomials. It also allows soft-decision decoding and is capable of decoding punctured codes. The core can operate in continuous or block modes, whichever is required by the channel. Either Tail Biting or Zero Flushing convolutional codes can be decoded in the block mode. All the configurable parameters, including operation mode, generator polynomials, punctured block size, and puncture pattern can be defined by the user to suit the needs of their application. The code rate and puncture pattern can also be changed dynamically through input ports during the operation of the decoder. Lattice's Block Viterbi Decoder IP is compatible with many networking and wireless standards that use different methods of convolutional encoding at the encoder.

Quick Facts

Table 1-1 through Table 1-4 give quick facts about the Block Viterbi Decoder IP core for LatticeEC™, LatticeECP™, LatticeECP2™, LatticeECP2M™, LatticeECP3™, LattticeSC™, LatticeSCM™, LatticeXP™, and LatticeXP2™, devices.

Table 1-1. Block Viterbi Decoder IP Core for LatticeEC/ECP/XP Devices Quick Facts

		Block Viterbi IP Configuration				
		IEEE 802.16 2004- SC PHY	**3GPP**	**DVB-S IEEE 802.11A**	**IEEE 802.16-2004-OFDM PHY (dynamic puncturing)**	**IEEE 802.16-2004-OFDM PHY (fixed puncturing)**
Core Requirements	FPGA Families Supported	LatticeEC/ECP/XP				
	Minimal Device Needed	LFEC1E LFECP6E LFXP3C	LFEC10E LFECP10E LFXP10C	LFEC3E LFECP6E LFXP3C	LFEC3E LFECP6E LFXP3C	LFEC6E LFECP6E LFXP6C
Resource Utilization	Targeted Device	LFEC20E-5F672C/ LFECP20E-5F672C/ LFXP20E-5F256C				
	LUTs	500	9950	2600	2750	3300
	sysMEM EBRs	2	16	4	4	4
	Registers	250	3200	900	1050	1200
Design Tool Support	Lattice Implementation	Diamond® 1.0 or ispLEVER® 8.1				
	Synthesis	Synopsys® Synplify® Pro for Lattice D-2009.12L-1				
	Simulation	Aldec® Active-HDL® 8.2 Lattice Edition				
		Mentor Graphics® ModelSim® SE 6.3F				

Table 1-2. Block Viterbi Decoder IP Core for LatticeECP2/ECP2M/XP2 Devices Quick Facts

		Block Viterbi IP Configuration				
		IEEE 802.16 2004- SC PHY	**3GPP**	**DVB-S IEEE 802.11A**	**IEEE 802.16-2004-OFDM PHY (dynamic puncturing)**	**IEEE 802.16-2004-OFDM PHY (fixed puncturing)**
Core Requirements	FPGA Families Supported	LatticeECP2/ECP2M/XP2				
	Minimal Device Needed	LFE2-6E LFE2M20E LFXP2-5E	LFE2-12E LFE2M20E LFXP2-17E	LFE2-6E LFE2M20E LFXP2-5E	LFE2-6E LFE2M20E LFXP2-5E	LFE2-6E LFE2M20E LFXP2-5E
Resource Utilization	Targeted Device	LFE2-50E-7F484C/ LFE2M35E-7F672C/ LFXP2-30E-7F484C				
	LUTs	500	11800	3050	3250	3500
	sysMEM EBRs	2	16	4	4	4
	Registers	250	3200	900	1050	1200
Design Tool Support	Lattice Implementation	Diamond 1.0 or ispLEVER 8.1				
	Synthesis	Synopsys Synplify Pro for Lattice D-2009.12L-1				
	Simulation	Aldec Active-HDL 8.2 Lattice Edition				
		Mentor Graphics ModelSim SE 6.3F				

Table 1-3. Block Viterbi Decoder IP Core for LatticeSC/SCM Devices Quick Facts

		Block Viterbi IP Configuration				
		IEEE 802.16 2004- SC PHY	**3GPP**	**DVB-S IEEE 802.11A**	**IEEE 802.16-2004-OFDM PHY (dynamic puncturing)**	**IEEE 802.16-2004-OFDM PHY (fixed puncturing)**
Core Requirements	FPGA Families Supported	LatticeSC/SCM				
	Minimal Device Needed	LFSC3GA15E/LFSCM3GA15EP1				
Resource Utilization	Targeted Device	LFSC3GA25E-7F900C/ LFSCM3GA25EP1-7F900C				
	LUTs	450	9450	2450	2650	3250
	sysMEM EBRs	2	16	4	4	4
	Registers	250	3400	900	1050	1200
Design Tool Support	Lattice Implementation	Diamond 1.0 or ispLEVER 8.1				
	Synthesis	Synopsys Synplify Pro for Lattice D-2009.12L-1				
	Simulation	Aldec Active-HDL 8.2 Lattice Edition				
		Mentor Graphics ModelSim SE 6.3F				

Table 1-4. Block Viterbi Decoder IP Core for LatticeECP3 Devices Quick Facts

		Block Viterbi IP Configuration				
		IEEE 802.16 2004- SC PHY	**3GPP**	**DVB-S IEEE 802.11A**	**IEEE 802.16-2004-OFDM PHY (dynamic puncturing)**	**IEEE 802.16-2004-OFDM PHY (fixed puncturing)**
Core Requirements	FPGA Families Supported	LatticeECP3				
	Minimal Device Needed	LFE3-35EA				
Resource Utilization	Targeted Device	LFE3-95E-8FN672CES				
	LUTs	500	11750	3050	3200	3500
	sysMEM EBRs	2	16	4	4	4
	Registers	250	3200	900	1050	1200
Design Tool Support	Lattice Implementation	Diamond 1.0 or ispLEVER 8.1				
	Synthesis	Synopsys Synplify Pro for Lattice D-2009.12L-1				
	Simulation	Aldec Active-HDL 8.2 Lattice Edition				
		Mentor Graphics ModelSim SE 6.3F				

Features

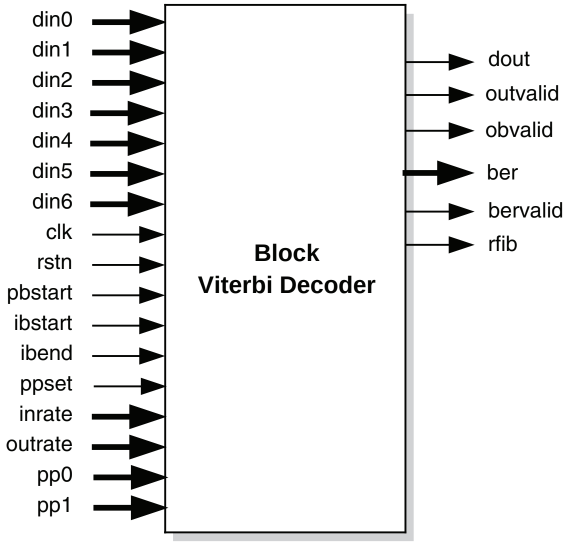
- Compatible with IEEE 802.16-2004 SC PHY/ OFDM PHY, IEEEE802.11a, 3GPP, 3GPP2, and DVB standards
- Supports multiple code rates: 1/2, 1/3, ... 1/7 for non-punctured codes, 2/3, 3/4, ..., 12/13 for punctured codes, and from m/(m+1) to m/(2m-1), where m is from 1 to 12, for dynamic punctured codes
- Variable constraint length from 3 to 9
- Supports dynamically variable code rates and puncture patterns
- Dynamic BER estimation option
- One-clock synchronous design
- Hard or parameterizable soft decision decoding. Hard and soft decision for non-punctured codes and soft decision for punctured codes
- Fully parallel or hybrid implementations. For a hybrid implementation, the degree of parallelism is parameterizable
- Parameterizable trace-back length
- Signed and unsigned representations for soft decision data
- Supports parameterized puncturing patterns
- Supports both continuous and block data input
- Supports both Tail Biting and Zero Flushing block convolutional codes
- Supports both one and two traceback schemes to cater to different coding scenarios

Functional Description

This chapter provides a functional description of the Block Viterbi Decoder IP core.

Figure 2-1 shows the interface diagram for Block Viterbi Decoder. The diagram shows all of the available ports for the IP. It should be noted that not all the I/O ports are available for all configurations.

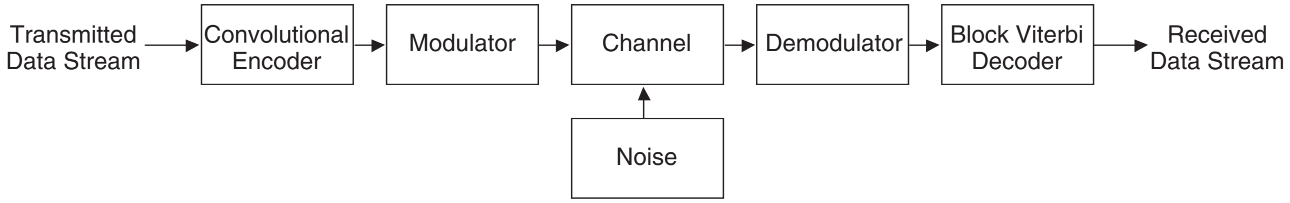
Figure 2-1. Block Viterbi Decoder Interface Diagram



General Description

Viterbi decoding is an efficient algorithm for decoding convolutionally encoded sequences corrupted by channel noise back to the original sequence. A digital transmit-receive system shown in Figure 2-2 uses a Viterbi decoder for decoding the convolutionally encoded data. The digital data stream (e.g., voice, image, or any packetized data) is encoded, modulated, and transmitted through a wired or wireless channel. A "noise" block connected to the channel symbolically denotes the channel noise. The data received from the channel at the receiver side is first demodulated and then decoded using the Viterbi decoder. The decoded output is equivalent to the transmitted digital data stream.

Figure 2-2. Digital Transmit-Receive System

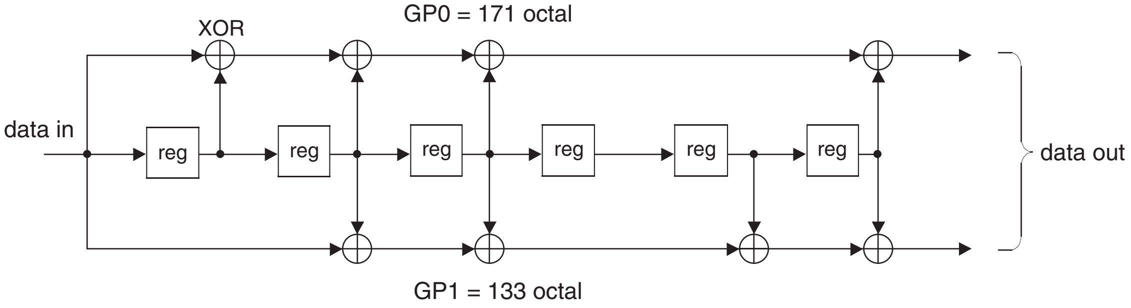


Convolutional Encoding

Figure 2-3 shows an example of convolutional encoding. In this example, each input symbol has two corresponding output symbols; hence the encoding is called 1/2 rate convolutional encoding. To generate the output, the encoder uses seven values of the input signal, one present and six past. The set of past values of input data is called the "state" of the encoder. The number of input data values used to generate the code is called the constraint length (K). In this case, the constraint length is 7. Each set of outputs is generated by XOR-ing a pattern of current and shifted values of input data. The patterns used to generate the coded output value can be expressed as binary strings called generator polynomials (GP). In this example, the generator polynomials are 171 and 133 (in octal).

The MSB of the generator polynomial corresponds to the input and the LSBs correspond to the state as shown in Figure 2-3. A bit value of ‘1’ in the generator polynomial represents a used bit and a value of ‘0’ signifies an unused bit.

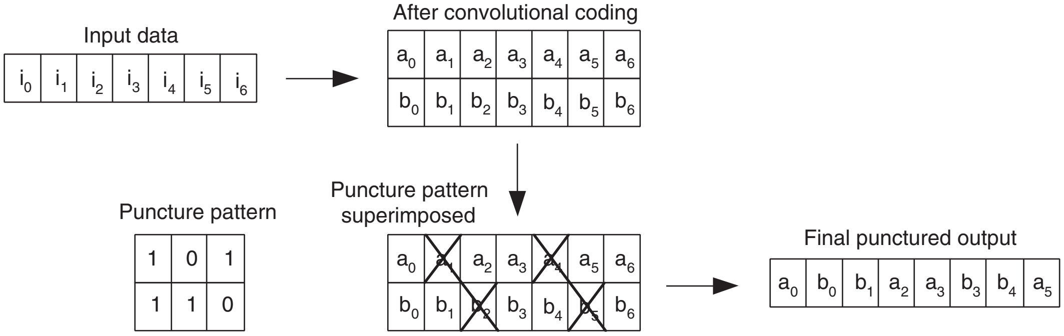
Figure 2-3. Convolutional Encoding



Punctured Codes and Depuncturing

After convolutional encoding, some of the encoded symbols can be selectively removed before transmission. This process, called “puncturing,” is a data compression method used to reduce the number of bits transmitted. Figure 2-4 shows an example of the puncturing process.

Figure 2-4. Puncturing Process

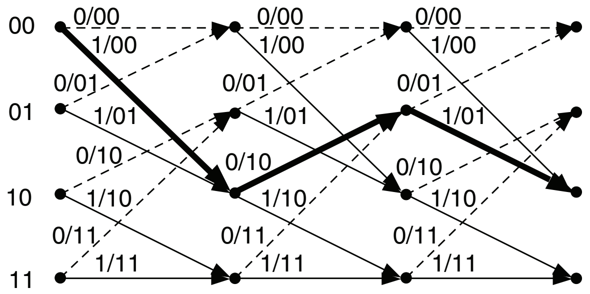

Input data
i_0 i_1 i_2 i_3 i_4 i_5 i_6
After convolutional coding
a_0 a_1 a_2 a_3 a_4 a_5 a_6
b_0 b_1 b_2 b_3 b_4 b_5 b_6
Puncture pattern
1 0 1
1 1 0
Puncture pattern superimposed
a_0 a_1 a_2 a_3 a_4 a_5 a_6
b_0 b_1 b_2 b_3 b_4 b_5 b_6
Final punctured output
a_0 b_0 b_1 a_2 a_3 b_3 b_4 a_5

If puncturing is employed in the encoder, the decoder will have to “depuncture” the data before decoding. Depuncturing is done by inserting NULL symbols for the punctured symbols. NULL symbols are equidistant from both ‘0’ and ‘1’. A pair of binary strings, called a “puncture pattern,” is used to identify punctured symbols. A “1” in a pattern means the corresponding symbol was not punctured in the encoder, while a “0” means the symbol has been punctured.

Viterbi Decoding

The convolutional encoding mentioned above can be considered as a series of state transitions for every input symbol. The input and the resulting state transitions can be shown in a special state transition diagram called a “trellis tree” or simply a “trellis.” A sample trellis tree is shown in Figure 2-5.

Figure 2-5. Trellis Tree



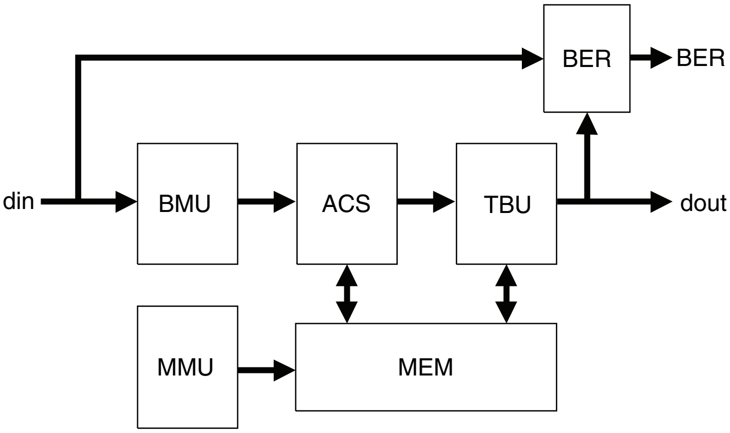
Trellis for 3 stages and constraint length = 3
Branches corresponding to input seq. 101 is highlighted

In the above trellis, the branches for three transitions are drawn. The path of the trellis for a typical input sequence, 101, is highlighted in the figure. Any transmission error alters the path traversed in the trellis. In Viterbi decoding, such a trellis is formed in memory, where the metrics corresponding to all paths are recorded. After constructing the trellis for a sufficient length (called the traceback length, L), the traceback process starts from node 0 in the last state. During the traceback process, the original sequence is reconstructed from the trellis. In error-prone applications, however, a trellis of length 2L is constructed and two traceback processes are employed. The first traceback starts from node 0, traces back L stages of the trellis, and ends up in a node which is more likely to be the right starting point for the second traceback. The second traceback starts from this reliable starting point and traces back another L nodes. The data corresponding to the second traceback are decoded to result in the original data stream.

Functional Description

A simplified implementation of the Lattice Block Viterbi Decoder IP is shown in Figure 2-6. A brief description of the modules is given below.

Figure 2-6. Internal Architecture of the Viterbi Decoder



Branch Metric Unit (BMU)

This module takes in the input data from the channel and computes a metric for each state and input combination. The metric is the Hamming distance for hard-decision encoded data and l1 norm (sum of absolute values) for soft decision encoded data. The BMU also includes a depuncturing unit for punctured codes. This module has three major sub-modules: state encoder, metric computer, and de-puncture unit.

Add, Compare, and Select Unit (ACS)

The ACS unit adds the current metric to the accumulated metric for each path and also determines the least metric for each state of the trellis. The accumulated metric is fetched from register files and stored back there, after adding the current metric. ACS also writes the survivor trellis path (the previous state information) in memory.

Traceback Unit (TBU)

The TBU performs decoding of the received data by tracing back the trellis from an appropriate starting node. Traceback and decoding is performed on a block of sequential nodes whose length is equal to the parameter Traceback Length. The Viterbi Decoder IP supports both one and two traceback schemes. In the one traceback scheme, the traceback starts from node 0 and happens for length L, where L is the traceback length. In the two traceback scheme, the first traceback starts from node 0 and happens for length L. This traceback determines a reliable starting node for the second traceback process. The second traceback starts from this reliable start node and happens for another length L.

The number of tracebacks employed and the traceback length are mostly set by the user, but the choice is restricted by other parameters and rules, as imposed by the Block Viterbi Decoder IP GUI.

Memory (MEM)

The memory stores the accumulated metric and the previous state information (traceback information).

Memory Management Unit (MMU)

The MMU generates addresses and read write signals for the memory during different phases of operation.

Bit Error Rate Monitor (BER)

This optional module is used to estimate the bit error rate of the channel. This is achieved by encoding the decoded output symbols using the same generator polynomials and comparing them with delayed input to the Viterbi decoder. Assuming the error in decoding is zero or negligible, the error determined by BER is equal to the channel error.

Other Modules

In Zero Flushing block decoding, an additional module called “Zero Padding Unit” is used. When the block length is not a multiple of the traceback length, the Zero Padding Unit automatically adds zero samples at the end of each block of input data.

Configuring the Block Viterbi Decoder

Puncture Settings

The Viterbi Decoder can be configured as a punctured or non-punctured decoder. A punctured decoder actually decodes convolutional codes that have been punctured after encoding. The puncture settings consist of the puncture block size (this is derived from code rate) and puncture patterns, `PP0` and `PP1`. The puncture settings are either fixed using the parameters in the IP GUI or can be dynamically set using input the ports, `inrate`, `outrate`, `pp0`, `pp1` and `ppset`. The values in `inrate` and `outrate` correspond to the rate factors `k` and `n`, respectively and they result in a code rate of `k`/`n`. The numerator of the code rate representation, `k` or the `inrate` is also called as the puncture block size in this document.

Continuous and Block Decoding

The decoding process can be applied on either continuous stream or blocks of input data. The main difference between these modes lies in the way the decoder performs the traceback operation. When the decoder is configured in continuous mode, it always performs two length-L tracebacks. The actual traceback length is set by the user through the IP GUI.

On the other hand, if the decoder is configured in block mode, the number of tracebacks and traceback length depends on the parameters of the decoder. The user has to specify the termination method that was used for the convolutional coding to enable the decoder to start from the correct initial state.

In dynamic puncturing mode, only block decoding is permitted.

Termination Modes

Convolutional encoders employ two block terminations methods: Zero Flushing and Tail Biting. In Zero Flushing mode, a series of zeros are added to the end of each block at the input of the convolutional encoder. In Tail Biting mode, the last few bits of each block are used to initialize the state of the encoder, before encoding that block. Both modes are widely used in various telecommunication standards.

Lattice's Block Viterbi decoder IP supports both of these termination methods. The choice of termination method is decided by the user and it must be exactly the same as what was used in the convolutional encoder.

Number of Tracebacks and Traceback Length

The accuracy of decoding depends to some extent on the starting node of a traceback operation. Usually, if the data was encoded using the Zero Flushing scheme and if the traceback length is equal to block length, the traceback can start at state 0. For all other schemes or for a continuous decoder, starting the traceback from zero state may not lead to right results. A reliable starting state can be determined by performing an additional traceback operation. The Block Viterbi Decoder can be configured to perform either 1 or 2 tracebacks by setting the parameter `Number of Tracebacks` in the IP GUI. For some configurations, the number of tracebacks can be selected by the user and for others, it is set automatically inside the decoder.

If `Number of Tracebacks` is equal to 1, the decoder performs length-L traceback starting from state 0 and does decoding. If the `Number of Tracebacks` is equal to 2, the decoder performs a length-L traceback from state 0 to determine a reliable starting point for second traceback. From that starting point, it performs a second length-L traceback and does decoding. For continuous decoders and block decoders with Tail Biting termination mode, `Number of Tracebacks` is internally set to 2. For block decoders with Zero Flushing termination mode, `Number of Tracebacks` can be set to either 1 or 2 by the user.

The traceback length is typically close to 7 to 9 times the constraint length (`K`) in most applications. Lattice's Viterbi Decoder IP allows the user to specify any traceback length between 3`K` and 14`K` for most configurations; however, the `Traceback Length` is restricted to be a multiple of puncture block size for fixed puncturing decoders. When the `Termination Mode` is set to "Tail Biting", the traceback length is internally set by the core to `Block Length*k/n`. When the decoder operates in dynamic puncture mode and `Number of Tracebacks` is set to 1, the `Traceback Length` should be a common multiple of all possible input rates and between 8. and 128. For example, if `Max Input Rate` is 4, the possible input rates are 1, 2, 3 and 4. Therefore, the `Traceback Length` can only be in the set {12, 24, 36, ..., 116, 128}.

Block Length

For block decoders, the block length is implicitly specified using the input signals `ibstart` and `ibend`. All the data between `ibstart` and `ibend` pulses, including both the ends, are taken to be part of the block. When `ibstart` is pulled high for one clock cycle the input data is read in as the first data of the block. The decoder continues to read the data in consecutive clock cycles into a block until it encounters a one clock cycle pulse in the `ibend` port. The block size has to be one of the legal values as given in Table 2-1, for the decoder to function correctly.

Table 2-1. Legal Values for Block Size

Termination Mode	Number of Tracebacks	Puncturing		
		None	Fixed	Dynamic
Zero Flushing	1	8 to 128	8 to 128*`k/n`, multiples of `n`	> 8, `Traceback Length*outrate/inrate`
Zero Flushing	2	> 8	> 8, multiples of `n`	> 8, multiples of `outrate`
Tail Biting	2	8 to 128	8 to 128*`k/n`, multiples of `n`	Not Applicable

Data Type

The Viterbi Decoder IP supports two commonly used binary representations, namely, sign-magnitude and unsigned offset, for soft decision data. In sign-magnitude representation, the most significant bit is a sign bit and the rest of the bits represent the magnitude. The most positive number corresponds to strong logic zero and other positive numbers are weak logic zeros. The most negative number corresponds to strong logic one and other negative numbers are weak logic ones. In unsigned offset representation, there is no sign bit in the number and all numbers are treated positive. The smallest number (all zeros) corresponds to strong logic zero and the biggest number (all ones) corresponds to strong logic one. The smaller numbers counting up from zero are progressively weaker logic zeros and bigger numbers counting down from the biggest number are progressively weaker logic ones.

Table 2-2 shows the data values and their interpretation in "Signed" and "Unsigned" data type configurations when `Soft Width` is 3.

Table 2-2. Interpretation of Signed and Unsigned Data

Signed Binary			Unsigned Offset		
Data	**Interpretation**		**Data**	**Interpretation**	
111	-3	(strong logic 1)	111	7	(strong logic 1)
110	-2	(weaker logic 1s)	110	6	(weaker logic 1s)
101	-1		101	5	
100	-0		100	4	
000	0	(weaker logic 0s)	011	3	(weaker logic 0s)
001	1		010	2	
010	2		001	1	
011	3	(strong logic 0)	000	0	(strong logic 0)

Signal Descriptions

The top level interface diagram of the Viterbi Decoder is shown in Figure 2-1. The details of the I/O ports are summarized in Table 2-3.

Table 2-3. Top Level I/O Interface

Port	Bits	I/O	Description
`clk`	1	I	System clock
`rstn`	1	I	System wide asynchronous active-low reset signal
`pbstart`	1	I	"Punctured block start" signal to indicate the start of a new block of punctured data. This signal is not available while decoding non-punctured codes.
`ibstart`	1	I	Input block start signal. This must be pulled high when the first data of a block is applied on the input port. This port is available for block decoding only.
`ibend`	1	I	Input block end signal. This signal must be pulled high to indicate the last data of a block being applied on the input port. This port is available for block decoding only.
`din0, din1, din2, din3, din4, din5, din6`	1 or 3 to 8 (each)	I	Data input buses – The buses become one bit inputs for hard decision decoding and equals to the soft width for soft decision decoding. The number of buses is 1 for punctured codes and *n* for non-punctured codes, where *n* is the code rate factor, from 2 to 7.
`inrate`	1-4	I	Input rate of the convolutional code for next block. This port is available only in dynamic puncturing mode.
`outrate`	2-5	I	Output rate of the convolutional code for next block. This port is available only in dynamic puncturing mode.

Table 2-3. Top Level I/O Interface (Continued)

Port	Bits	I/O	Description
pp0	1-12	I	Puncture pattern 0 of the convolutional code for next block. This port is available only in dynamic puncturing mode.
pp1	1-12	I	Puncture pattern 1 of the convolutional code for next block. This port is available only in dynamic puncturing mode.
ppset	1	I	Puncture rate and puncture pattern set signal. The new input rate, output rate and puncture patterns are set when ppset goes high. This port is available only in dynamic puncturing mode.
dout	1	O	Output decoded data.
outvalid	1	O	Output valid signal. This indicates the output on dout is a valid decoded value.
obvalid	1	O	Output block valid signal. This signal remains high for the entire duration of the output block. This signal is present only for punctured and block decoding.
ber	16	O	Bit-error rate output. This port is available for continuous decoding only.
bervalid	1	O	Identifies that a new Bit Error Rate (BER) value is available at the ber output port. This signal goes high once every *B* clock cycles, where B=2^(BER Period), is the duration over which BER is computed. This port is available for continuous decoding only.
rfib	1	O	"Ready for input block" signal. 1. This port is not available for non-punctured decoders. 2. For fixed puncturing, this signal goes high every L*(2^c) cycles periodically counting from ibstart for each input block, where L is the traceback length and c is the hybrid index. After applying an input block (after ibend going active), the user has to wait for the next rfib pulse before he can start giving the next input block. In fixed puncturing mode, this port is available only for zero flushing block decoding and Number of Tracebacks is 2. 3. For dynamic puncturing, this port is always available. It goes low one cycle after an input block starts (after ibstart signal going high). It goes high a few cycles after an input block ends (after ibend going low).

Interfacing with the Block Viterbi Decoder

Lattice's Block Viterbi Decoder provides several handshake signals for interfacing the decoder with other sub-systems.

In non-punctured, continuous modes, the input and output data rates are the same and it is straightforward to connect the decoder in a system. The only control output in these modes, `outvalid`, indicates when the output data is ready. Initially at reset, `outvalid` is low and it goes high after several clock cycles depending on the output latency for the chosen configuration. The latency depends on different decoder parameters, but mainly on traceback length. A sample timing diagram for this configuration is shown in Figure 2-7. The hybrid version of the non-punctured, continuous Viterbi decoder uses similar handshake mechanism as the parallel version. The main difference is that the data rate is a fraction of the clock rate for hybrid implementations. Figure 2-9 shows the timing diagram for a sample hybrid decoder.

A punctured, continuous mode Viterbi decoder has an additional input signal, `pbstart`, which is used to specify the start of each punctured block. This signal is required to synchronize the punctured blocks correctly for depuncturing inside the decoder. As in non-punctured mode, the input is assumed to be continuous. The output will have one gap per puncture block, which is indicated by `outvalid` going low. This gap is required to account for the data rate differences between the input and the output of the decoder. Figure 2-8 shows the timing diagram for a sample punctured, continuous decoder. The hybrid mode for this implementation has similar timing characteristics, except that the data rate is a fraction of the clock rate and hence the data and output control signals accordingly span multiple clock cycles. However the input control signals are all single clock cycle pulses as they are scanned only for one cycle. A sample timing diagram for a hybrid, punctured decoder is shown in Figure 2-10.

When a Viterbi Decoder is configured for block modes, the signals `ibstart` and `ibend` are used to specify the start and end of input blocks. For fixed puncturing decoders, when the decoder is configured for zero flushing termination mode with two tracebacks, an additional output control signal `rfib` is provided. After an `ibend` signal is applied signifying the end of a block, the next `ibstart` can only be applied, after the `rfib` goes high. To ensure processing of blocks without discontinuity, the `rfib` signal goes high at the end of every L cycles, where L is the traceback length. So if a block ends exactly at a traceback length boundary, `rfib` will go high while `ibend` goes high, allowing `ibstart` to be applied in the next clock cycle. This way continuous blocks can be applied to the decoder. For dynamic puncturing decoders, the `rfib` port is always present. The signal `rfib` goes low one cycle after `ibstart` is received. It remains low during the time an input block is received. It goes high a few cycles after `ibend` comes through. Refer to Figure 2-11 for a sample timing diagram for a block decoder.

When it is required to change the code rate or puncture pattern dynamically during the operation of the decoder, the Block Viterbi Decoder can be configured as a dynamic puncturing decoder. In this mode, the code rate and puncture patterns are set through input ports. The code rate is set using the input ports `inrate` and `outrate`. Care should be taken to ensure the following rule is followed for the rates: `inrate` < `outrate` < 2*inrate. Otherwise the decoder will not function correctly. An exception to this rule is when `inrate` = 1, at which time, the `outrate` has to be 2. Each of the puncture patterns must be `inrate` bits wide and the total number of '1's in PP0 and `PP1` must be equal to `outrate`. The values of `inrate`, `outrate`, `PP0` and `PP1` are read-in only when `ppset` goes high. The new puncture settings are set when `ppset` goes high and they are effective from the next input block. Before the decoder is applied with the first block, the puncture settings have to be set. Figure 2-12 shows the timing diagram for a typical dynamic punctured decoder.

Timing Diagrams

The top-level timing diagrams for several cases are given in the Figure 2-7 through Figure 2-12.

Figure 2-7. Timing Diagram for a Continuous, Parallel, Non-Punctured Decoder

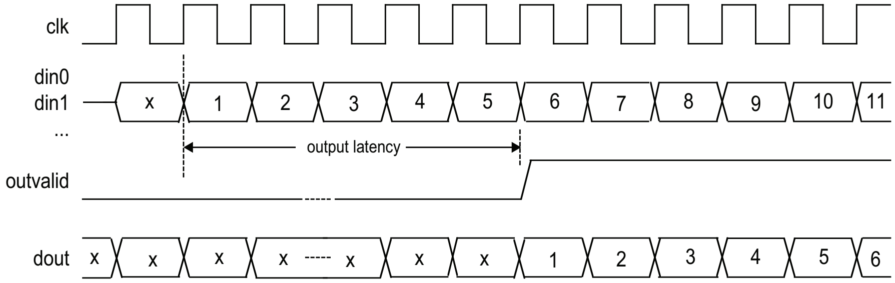


Figure 2-8. Timing Diagram for a Continuous, Parallel, Punctured (Rate=2/3) Decoder

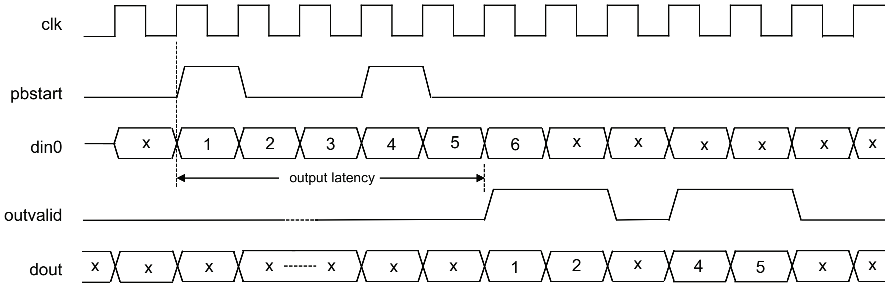


Figure 2-9. Timing Diagram for a Continuous, Hybrid (Two Cycles), Non-punctured Decoder

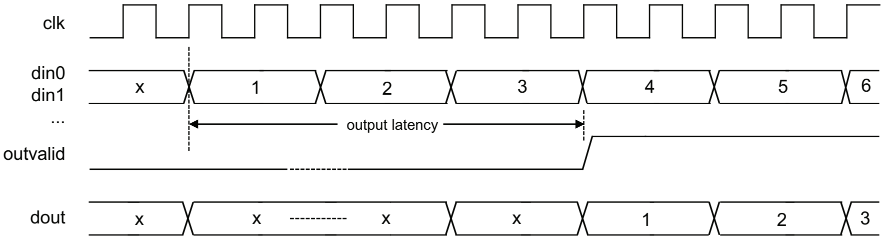


Figure 2-10. Timing Diagram for a Continuous, Hybrid (Two cycles), Punctured (Rate=2/3) Decoder

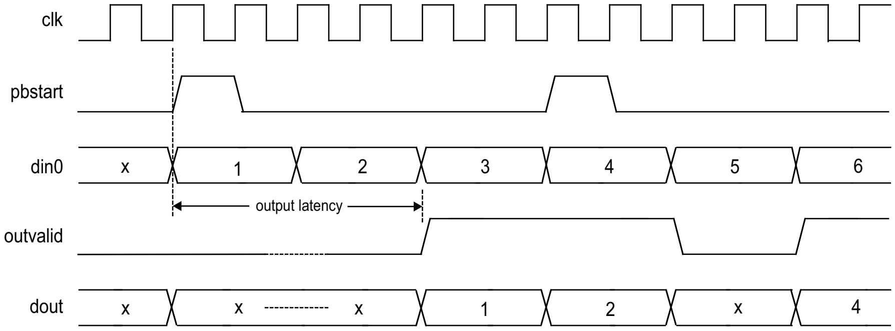


Figure 2-11. Timing Diagram for a Block, Parallel, Non-punctured Decoder with Two Tracebacks

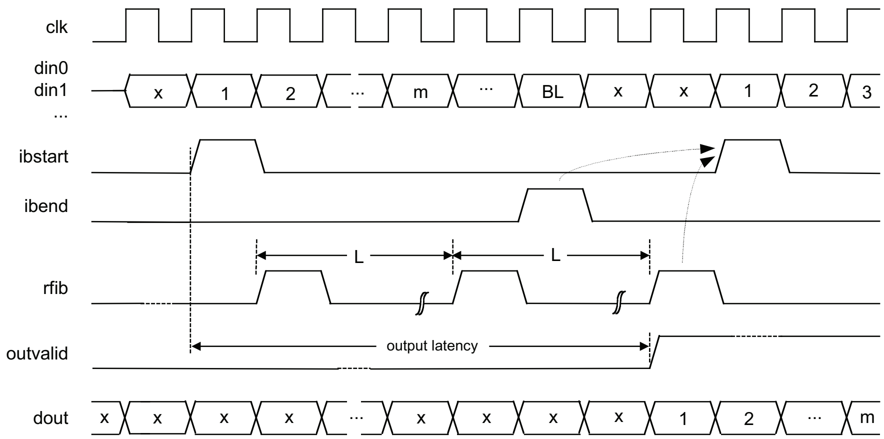
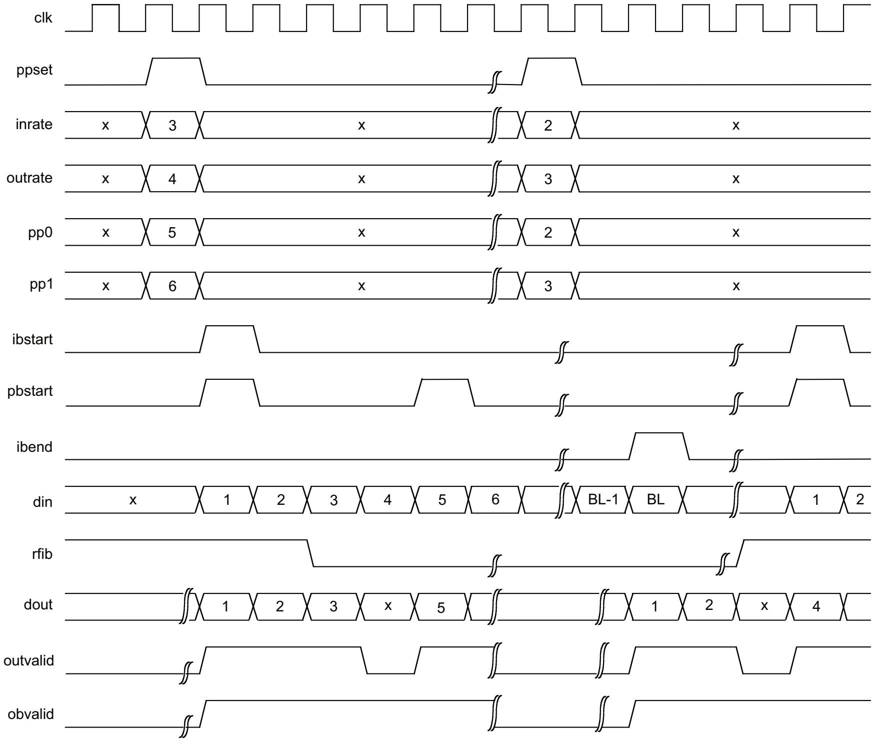


Figure 2-12. Timing Diagram for a Block, Parallel, Dynamic Punctured Decoder



Core Configurations

Table 2-4 lists the configurations and parameters for some standard configurations supported by the IP core. Results for these configurations in each Lattice device family are provided in Appendix A: “Resource Utilization” on page 31.

Table 2-4. Core Configurations

Configuration	1	2	3	4	5
Compatible Standard	IEEE 802.16 2004- SC PHY	3GPP	DVB-S, IEEE 802.11A	IEEE 802.16-2004-OFDM PHY (dynamic puncturing)	IEEE 802.16-2004-OFDM PHY (fixed puncturing)
Primary Options					
Constraint length (K)	3	9	7	7	7
Code Rate (k/n)	2/3	1/2	1/2	1/2	5/6
Operation Mode	Block	Block	Continuous	Block	Block
Traceback Length	30	63	42	42	90
Block Options					
Termination Mode	Tail Biting	Zero Flushing		Zero Flushing	Zero Flushing
Number of Tracebacks	2	2	-	2	2
Puncture Settings					
Puncturing	Fixed	None	None	Dynamic	Fixed
Puncture Pattern	10 11	—	—	Through Port	10101 11010
Max Input Rate	—	—	—	5	—
Max Output Rate	—	—	—	6	—
Generator Polynomials					
Radix	Octal	Octal	Octal	Octal	Octal
GP0, GP1 (GP2,... N/A)	7_8 5_8	561_8 753_8	171_8 133_8	171_8 133_8	171_8 133_8
Implementation					
Implementation Method	Parallel	Parallel	Parallel	Parallel	Parallel
Hybrid Index	—	—	—	—	—
Inputs					
Decoder Input	Soft Decision	Soft Decision	Soft Decision	Soft Decision	Soft Decision
Soft Width	3	3	3	3	4
Data Type	Signed	Signed	Signed	Signed	Unsigned
BER (Bit Error Rate)					
BER Monitor	No	No	No	No	No
BER Period	—	—	—	—	—

Chapter 3:

Parameter Settings

The IPexpress™ tool is used to create IP and architectural modules in the Diamond or ispLEVER software. Refer to "IP Core Generation" on page 22 for a description on how to generate the IP.

Table 3-1 provides the list of user configurable parameters for the Block Viterbi Decoder IP core. The parameter settings are specified using the Block Viterbi Decoder IP core Configuration GUI in IPexpress. The numerous PCI Express parameter options are partitioned across multiple GUI tabs as shown in this chapter.

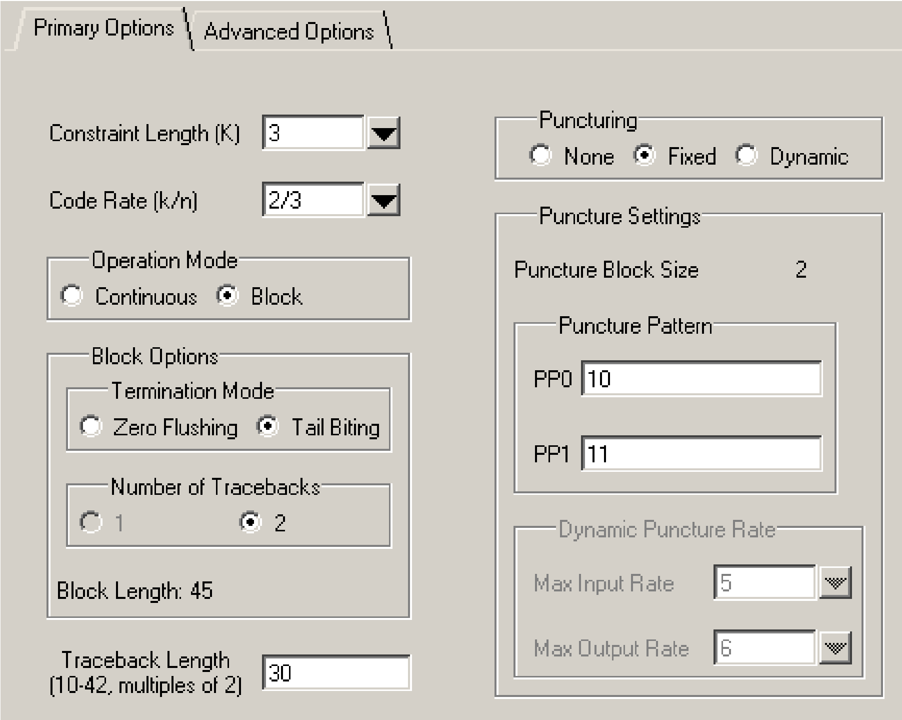
Table 3-1. Block Viterbi Decoder Parameter Descriptions

Parameter	Range	Default
Primary Options		
Constraint length (K)	3 to 9	3
Code Rate (k/n)	1/2, 1/3,...,1/7 for Non-Punctured Decoder 2/3, 3/4,..., 12/13 for Punctured Decoder	2/3
Operation Mode	Continuous/Block	Block
Traceback Length	3K to 14K	30
Block Options		
Termination Mode	Zero Flushing/Tail Biting	Tail Biting
Number of Tracebacks	1, 2	-
Puncture Settings		
Puncturing	None/Fixed/Dynamic	Fixed
Puncture Pattern	PP0 and PP1 are each k bits wide binary patterns	11 10
Max Input Rate	1 to 12 when Number of Tracebacks = 2 1 to 6 when Number of Tracebacks = 1	—
Max Output Rate	(Max Input Rate+1) to (2*Max Input Rate-1)	
Generator Polynomials		
Radix	Binary/Octal/Hexadecimal	Octal
GP0, GP1, GP2, GP3, GP4, GP5, GP6	K bits wide number for each polynomial	7 5
Implementation		
Implementation Method	Parallel/Hybrid	Parallel
Hybrid Index	1 to (K-1)	—
Inputs		
Decoder Input	Hard Decision/Soft Decision	Soft Decision
Soft Width	3 to 8 bits	3
Data Type	Signed/Unsigned	Signed
BER (Bit Error Rate)		
BER Monitor	Yes/No}	No
BER Period	4 to 32	—

Primary Options Tab

Figure 3-1 shows the contents of the Primary Options tab.

Figure 3-1. Primary Options Tab



Constraint Length (K) 3

Code Rate (k/n) 2/3

Operation Mode
○ Continuous ● Block

Block Options

Termination Mode
○ Zero Flushing ● Tail Biting

Number of Tracebacks
○ 1 ● 2

Block Length: 45

Traceback Length
(10-42, multiples of 2) 30

Puncturing
○ None ● Fixed ○ Dynamic

Puncture Settings

Puncture Block Size 2

Puncture Pattern
PP0 10
PP1 11

Dynamic Puncture Rate
Max Input Rate 5
Max Output Rate 6

Primary Options

Constraint length (K)

Constraint length is equal to the number of input data values (present and past) used to generate the convolutional code in the encoder.

Code Rate (k/n)

This is the symbol output rate of the encoder, defined as the number of output bits per input bit in the encoder. For non-punctured decoder, this can be set from 1/2 to 1/7. For punctured decoder, this can be set to m/m+1, where m can range from 2 to 12.

Operation Mode

The operation mode of the decoder is either continuous or block.

Block Options

Termination Mode

This is the termination mode used for the convolutional coding of the input block. This parameter is required for block operation modes.

Number of Tracebacks

Number of tracebacks performed for decoding. This option is available only when zero-flushing termination mode is used.

Traceback Length

Traceback length is the number of trellis states the decoder traces back for performing decoding. The traceback length must be between 3K to 14K, where K is the Constraint Length. The range is further restricted by the value of some block related parameters. See the Configuring the Block Viterbi Decoder section of this document for details.

Puncturing

This option specifies whether input data is punctured or not. If the input is punctured, the decoder can be set to use either fixed puncture settings or dynamically variable puncture settings.

Puncture Settings

Puncture Pattern
Puncture pattern for fixed puncturing decoders. For dynamic puncture decoders, this pattern is applied through the input port.

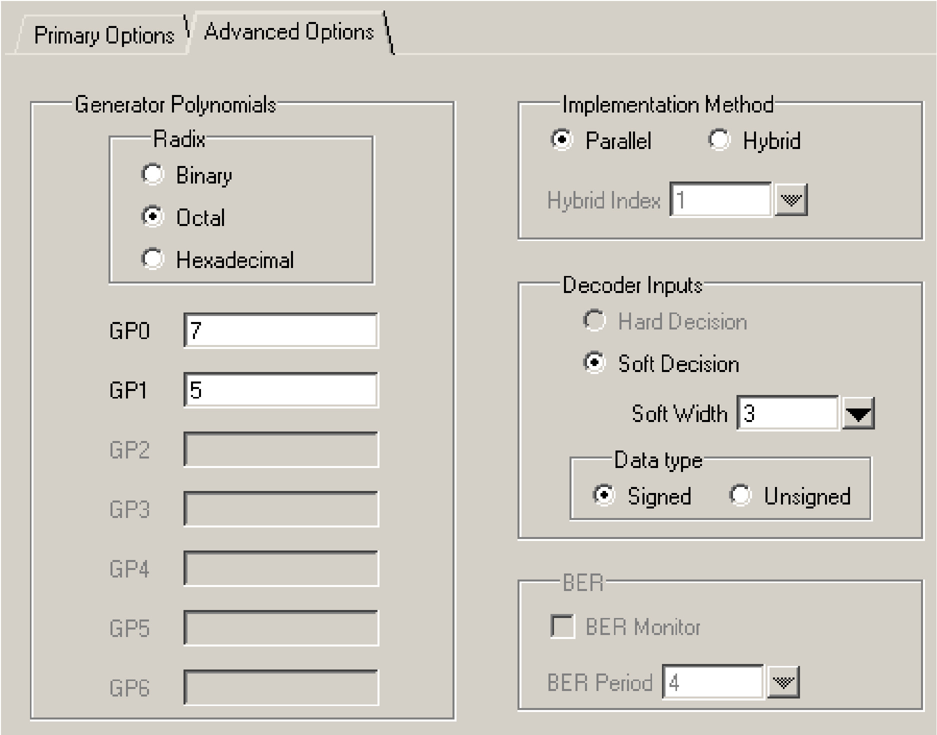
Max Input Rate
This is the maximum value for the numerator, k, of the code rate, when the puncture rate is dynamically set through an input port.

Max Output Rate
This is the maximum value for the denominator, n, of the code rate, when the puncture rate is dynamically set through an port.

Advanced Options Tab

Figure 3-2 shows the contents of the Advanced Options tab.

Figure 3-2. Advanced Options Tab



Generator Polynomials

Radix
○ Binary
● Octal
○ Hexadecimal

GP0 7
GP1 5
GP2
GP3
GP4
GP5
GP6

Implementation Method
● Parallel ○ Hybrid
Hybrid Index 1

Decoder Inputs
○ Hard Decision
● Soft Decision
Soft Width 3

Data type
● Signed ○ Unsigned

BER
□ BER Monitor
BER Period 4

Generator Polynomials

Radix
This parameter specifies the number system in which the generator polynomials are specified.

GP0, GP1, GP2, GP3, GP4, GP5, GP6

Generator polynomials used for generating the convolutional code. Two polynomials are always used for punctured decoders (either fixed or dynamic). For non-punctured decoders, the number of polynomials used is equal to n, where n is denominator of the Code Rate (k/n).The width of each polynomial is equal to constraint length, K.

Implementation Method

The implementation method can be either “parallel” or “hybrid”. In the parallel implementation, the decoder can produce one output data in one cycle. In hybrid implementations, it takes multiple clock cycles to generate each output data, but a smaller number of device resources are used.

Hybrid Index

This controls the resource-throughput trade-off in hybrid implementations. It takes $2^{\text{Hybrid Index}}$ cycles to produce one output data.

Inputs

Decoder Input

Specifies whether the decoder is fed with a hard decision or soft decision input. For punctured decoders, this option is not available and decoder has to be fed with soft decision inputs.

Soft Width

Input data width for soft decision inputs.

Data Type

Specifies whether the input data type is represented in sign-magnitude form (signed) or unsigned offset form (unsigned). See the section “configuring the Block Viterbi Decoder” for details.

BER (Bit Error Rate)

BER Monitor

Specifies whether the optional bit error rate (BER) monitor is added to the Viterbi decoder.

BER Period

This determines the duration for which the BER is accumulated. The BER value starts accumulating from zero for up to 2^(BER Period) clock cycles. After this period, the accumulated value is placed on the BER output port. The BER value is then reset and the monitor starts accumulating again.

IP Core Generation

This chapter provides information on how to generate the Block Viterbi Decoder IP core using the Diamond or ispLEVER software IPexpress tool, and how to include the core in a top-level design.

Licensing the IP Core

An IP core- and device-specific license is required to enable full, unrestricted use of the Block Viterbi Decoder IP corein a complete, top-level design. Instructions on how to obtain licenses for Lattice IP cores are given at:

http://www.latticesemi.com/products/intellectualproperty/aboutip/isplevercoreonlinepurchas.cfm

Users may download and generate the Block Viterbi Decoder IP core and fully evaluate the core through functional simulation and implementation (synthesis, map, place and route) without an IP license. The Block Viterbi Decoder IP corealso supports Lattice's IP hardware evaluation capability, which makes it possible to create versions of the IP core that operate in hardware for a limited time (approximately four hours) without requiring an IP license. See "Hardware Evaluation" on page 27 for further details. However, a license is required to enable timing simulation, to open the design in the Diamond or ispLEVER EPIC tool, and to generate bitstreams that do not include the hardware evaluation timeout limitation.

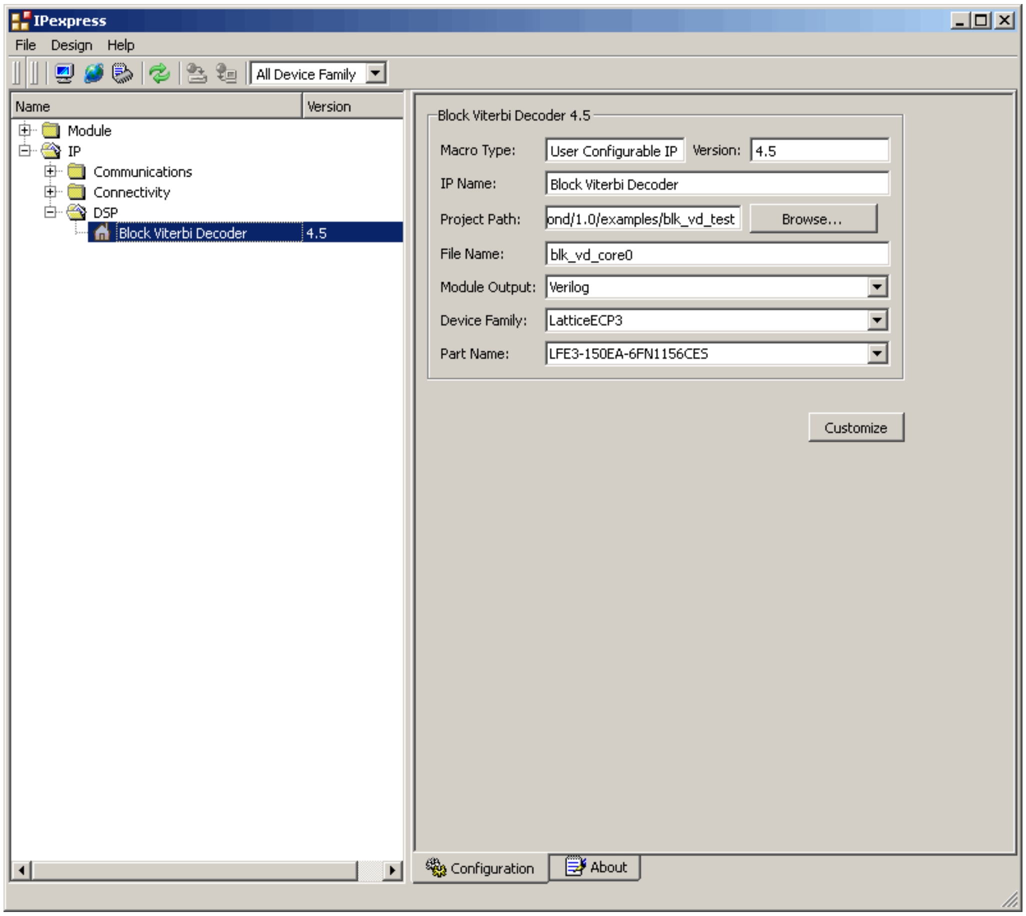
Getting Started

The Block Viterbi Decoder IP core is available for download from Lattice's IP server using the IPexpress tool. The IP files are automatically installed using ispUPDATE technology in any customer-specified directory. After the IP core has been installed, the IP core will be available in the IPexpress GUI dialog box shown in Figure 4-1.

The IPexpress tool GUI dialog box for the Block Viterbi Decoder IP core is shown in Figure 4-1. To generate a specific IP core configuration the user specifies:

- **Project Path** – Path to the directory where the generated IP files will be loaded.
- **File Name** – "username" designation given to the generated IP core and corresponding folders and files.
- **(Diamond) Module Output** – Verilog or VHDL.
- **(ispLEVER) Design Entry Type** – Verilog HDL or VHDL.
- **Device Family** – Device family to which IP is to be targeted (e.g. LatticeSCM, Lattice ECP2M, LatticeECP3, etc.). Only families that support the particular IP core are listed.
- **Part Name** – Specific targeted part within the selected device family.

Figure 4-1. The IPexpress Tool Dialog Box (Diamond Version)



Block Viterbi Decoder 4.5

Macro Type: User Configurable IP Version: 4.5

IP Name: Block Viterbi Decoder

Project Path: ond/1.0/examples/blk_vd_test

File Name: blk_vd_core0

Module Output: Verilog

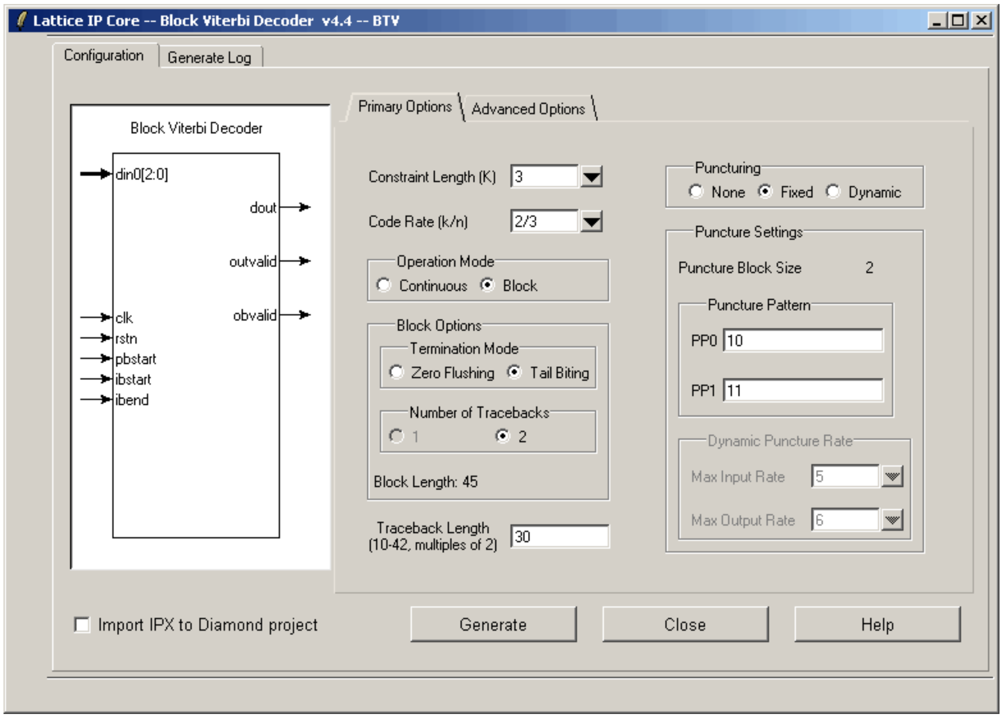
Device Family: LatticeECP3

Part Name: LFE3-150EA-6FN1156CES

Note that if the IPexpress tool is called from within an existing project, Project Path, Module Output (Design Entry in ispLEVER), Device Family and Part Name default to the specified project parameters. Refer to the IPexpress tool online help for further information.

To create a custom configuration, the user clicks the **Customize** button in the IPexpress tool dialog box to display the Block Viterbi Decoder IP coreConfiguration GUI, as shown in Figure 4-2. From this dialog box, the user can select the IP parameter options specific to their application. Refer to "Parameter Settings" on page 18for more information on the Block Viterbi Decoder IP coreparameter settings.

Figure 4-2. The IPexpress Tool Dialog Box - Configuration GUI (Diamond Version)



Constraint Length (K) 3

Code Rate (k/n) 2/3

Operation Mode: ○ Continuous ● Block

Block Options

Termination Mode: ○ Zero Flushing ● Tail Biting

Number of Tracebacks: ○ 1 ● 2

Block Length: 45

Traceback Length (10-42, multiples of 2) 30

Puncturing: ○ None ● Fixed ○ Dynamic

Puncture Settings

Puncture Block Size 2

Puncture Pattern

PP0 10

PP1 11

Dynamic Puncture Rate

Max Input Rate 5

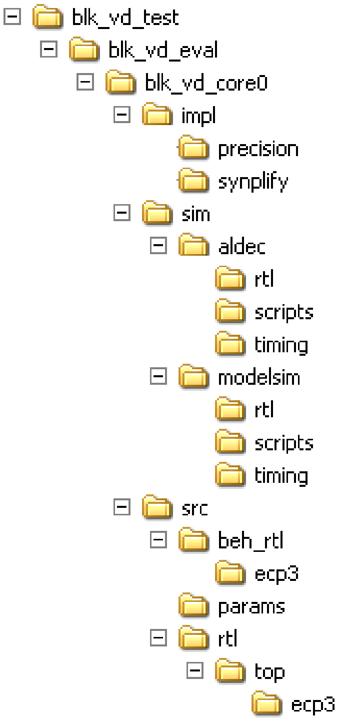
Max Output Rate 6

□ Import IPX to Diamond project

IPexpress-Created Files and Top Level Directory Structure

When the user clicks the **Generate** button in the IP Configuration dialog box, the IP core and supporting files are generated in the specified "Project Path" directory. The directory structure of the generated files is shown in Figure 4-3.

Figure 4-3. LatticeECP3 Block Viterbi Decoder IP core Directory Structure



```
blk_vd_test
  blk_vd_eval
    blk_vd_core0
      impl
        precision
        synplify
      sim
        aldec
          rtl
          scripts
          timing
        modelsim
          rtl
          scripts
          timing
      src
        beh_rtl
          ecp3
        params
        rtl
          top
            ecp3
```

Table 4-1 provides a list of key files and directories created by the IPexpress tool and how they are used. The IPexpress tool creates several files that are used throughout the design cycle. The names of most of the created files are customized to the user's module name specified in the IPexpress tool.

Table 4-1. File List

File	Description
<*username*>_inst.v	This file provides an instance template for the IP.
<*username*>.v	This file provides the VITERBI core for simulation.
<*username*>_beh.v	This file provides a behavioral simulation model for the VITERBI core.
<*username*>_bb.v	This file provides the synthesis black box for the user's synthesis.
<*username*>.ngo *.ngo	The ngo files provide the synthesized IP core.
<*username*>.lpc	This file contains the IPexpress tool options used to recreate or modify the core in the IPexpress tool.
<*username*>_generate.tcl	Created when GUI "Generate" button is pushed, invokes generation, may be run from command line.
<*username*>_generate.log	IPexpress scripts log file.
<*username*>_gen.log	IPexpress IP generation log file

Instantiating the Core

The generated Viterbi IP core package includes black-box (<*username*>_bb.v) and instance (<*user-name*>_inst.v) templates that can be used to instantiate the core in a top-level design. An example RTL top-level reference source file that can be used as an instantiation template for the IP core is provided in `\<project_dir>\blk_vd_eval\<username>\src\rtl\top`. Users may also use this top-level reference as the starting template for the top-level for their complete design.

Running Functional Simulation

Simulation support for the Viterbi IP core is provided for Aldec Active-HDL (Verilog and VHDL) simulator, Mentor Graphics ModelSim simulator. The functional simulation includes a configuration-specific behavioral model of the Viterbi IP core. The test bench sources stimulus to the core, and monitors output from the core. The generated IP core package includes the configuration-specific behavior model (<*username*>_beh.v) for functional simulation in the “Project Path” root directory. The simulation scripts supporting ModelSim evaluation simulation is provided in `\<project_dir>\blk_vd_eval\<username>\sim\modelsim\scripts`. The simulation script supporting Aldec evaluation simulation is provided in `\<project_dir>\blk_vd_eval\<username>\sim\aldec\scripts`. Both ModelSim and Aldec simulation is supported via test bench files provided in `\<project_dir>\blk_vd_eval\testbench`. Models required for simulation are provided in the corresponding `\models` folder. Users may run the Aldec evaluation simulation by doing the following:

1. Open Active-HDL.
2. Under the Tools tab, select **Execute Macro**.
3. Browse to folder `\<project_dir>\blk_vd_eval\<username>\sim\aldec\scripts` and execute one of the "do" scripts shown.

Users may run the ModelSim evaluation simulation by doing the following:

1. Open ModelSim.
2. Under the File tab, select **Change Directory** and choose the folder `<project_dir>\blk_vd_eval\<username>\sim\modelsim\scripts`.
3. Under the Tools tab, select **Execute Macro** and execute the ModelSim “do” script shown.

Note: When the simulation completes, a pop-up window will appear asking “Are you sure you want to finish?” Answer ***No*** *to analyze the results. Answering* ***Yes*** *closes ModelSim.*

Synthesizing and Implementing the Core in a Top-Level Design

The Block Viterbi Decoder IP itself is synthesized and provided in NGO format when the core is generated through IPexpress. You may combine the core in your own top-level design by instantiating the core in your top-level file as described in “Instantiating the Core” on page 26 and then synthesizing the entire design with either Synplify or Precision RTL Synthesis.

The following text describes the evaluation implementation flow for Windows platforms. The flow for Linux and UNIX platforms is described in the Readme file included with the IP core.

The top-level file <*userame*>_top.v is provided in `\<project_dir>\blk_vd_eval\<username>\src\rtl\top`. Push-button implementation of the reference design is supported via the project file <*username*>.ldf (Diamond) or .syn (ispLEVER) located in `\<project_dir>\blk_vd_eval\<username>\impl\(synplify or precision).`

To use this project file in Diamond:

1. Choose **File > Open > Project**.
2. Browse to `\<project_dir>\blk_vd_eval\<username>\impl\synplify (or precision)` in the Open Project dialog box.
3. Select and open <*username*>.ldf. At this point, all of the files needed to support top-level synthesis and implementation will be imported to the project.
4. Select the **Process** tab in the left-hand GUI window.
5. Implement the complete design via the standard Diamond GUI flow.

To use this project file in ispLEVER:

1. Choose **File > Open Project.**
2. Browse to `\<project_dir>\blk_vd_eval\<username>\impl\synplify (or precision)` in the Open Project dialog box.
3. Select and open <*username*>.syn. At this point, all of the files needed to support top-level synthesis and implementation will be imported to the project.
4. Select the device top-level entry in the left-hand GUI window.
5. Implement the complete design via the standard ispLEVER GUI flow.

Hardware Evaluation

The Block Viterbi Decoder IP supports Lattice's IP hardware evaluation capability, which makes it possible to create versions of the IP core that operate in hardware for a limited period of time (approximately four hours) without requiring the purchase of an IP license. It may also be used to evaluate the core in hardware in user-defined designs.

Enabling Hardware Evaluation in Diamond:

Choose **Project > Active Strategy > Translate Design Settings**. The hardware evaluation capability may be enabled/disabled in the Strategy dialog box. It is enabled by default.

Enabling Hardware Evaluation in ispLEVER:

In the Processes for Current Source pane, right-click the **Build Database** process and choose **Properties** from the dropdown menu. The hardware evaluation capability may be enabled/disabled in the Properties dialog box. It is enabled by default.

Updating/Regenerating the IP Core

By regenerating an IP core with the IPexpress tool, you can modify any of its settings including: device type, design entry method, and any of the options specific to the IP core. Regenerating can be done to modify an existing IP core or to create a new but similar one.

Regenerating an IP Core in Diamond

To regenerate an IP core in Diamond:

1. In IPexpress, click the **Regenerate** button.
2. In the Regenerate view of IPexpress, choose the IPX source file of the module or IP you wish to regenerate.

3. IPexpress shows the current settings for the module or IP in the Source box. Make your new settings in the T**arget** box.

4. If you want to generate a new set of files in a new location, set the new location in the **IPX Target File** box. The base of the file name will be the base of all the new file names. The IPX Target File must end with an .ipx extension.

5. Click **Regenerate.** The module's dialog box opens showing the current option settings.

6. In the dialog box, choose the desired options. To get information about the options, click **Help**. Also, check the About tab in IPexpress for links to technical notes and user guides. IP may come with additional information. As the options change, the schematic diagram of the module changes to show the I/O and the device resources the module will need.

7. To import the module into your project, if it's not already there, select **Import IPX to Diamond Project** (not available in stand-alone mode).

8. Click **Generate**.

9. Check the Generate Log tab to check for warnings and error messages.

10. Click **Close**.

The IPexpress package file (.ipx) supported by Diamond holds references to all of the elements of the generated IP core required to support simulation, synthesis and implementation. The IP core may be included in a user's design by importing the .ipx file to the associated Diamond project. To change the option settings of a module or IP that is already in a design project, double-click the module's .ipx file in the File List view. This opens IPexpress and the module's dialog box showing the current option settings. Then go to step 6 above.

Regenerating an IP Core in ispLEVER

To regenerate an IP core in ispLEVER:

1. In the IPexpress tool, choose **Tools > Regenerate IP/Module**.

2. In the Select a Parameter File dialog box, choose the Lattice Parameter Configuration (.lpc) file of the IP core you wish to regenerate, and click **Open**.

3. The Select Target Core Version, Design Entry, and Device dialog box shows the current settings for the IP core in the Source Value box. Make your new settings in the Target Value box.

4. If you want to generate a new set of files in a new location, set the location in the LPC Target File box. The base of the .lpc file name will be the base of all the new file names. The LPC Target File must end with an .lpc extension.

5. Click **Next**. The IP core's dialog box opens showing the current option settings.

6. In the dialog box, choose desired options. To get information about the options, click **Help**. Also, check the About tab in the IPexpress tool for links to technical notes and user guides. The IP core might come with additional information. As the options change, the schematic diagram of the IP core changes to show the I/O and the device resources the IP core will need.

7. Click **Generate**.

8. Click the **Generate Log** tab to check for warnings and error messages.

Chapter 5:

Support Resources

This chapter contains information about Lattice Technical Support, additional references, and document revision history.

Lattice Technical Support

There are a number of ways to receive technical support.

Online Forums

The first place to look is Lattice Forums (http://www.latticesemi.com/support/forums.cfm). Lattice Forums contain a wealth of knowledge and are actively monitored by Lattice Applications Engineers.

Telephone Support Hotline

Receive direct technical support for all Lattice products by calling Lattice Applications from 5:30 a.m. to 6 p.m. Pacific Time.

- For USA & Canada: 1-800-LATTICE (528-8423)
- For other locations: +1 503 268 8001

In Asia, call Lattice Applications from 8:30 a.m. to 5:30 p.m. Beijing Time (CST), +0800 UTC. Chinese and English language only.

- For Asia: +86 21 52989090

E-mail Support

- techsupport@latticesemi.com
- techsupport-asia@latticesemi.com

Local Support

Contact your nearest Lattice Sales Office.

Internet

www.latticesemi.com

References

[1] 3GPP TS 25.212 V4.2.0 (2001-09)

[2] 3GPP2 C.S0002-A Version 5.0 Date: July 13, 2001

[3] IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems, October 2004 (IEEE Standard 802.16-2004)

[4] IEEE Standard for Information Technology Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications

[5] Digital Video Broadcasting (DVB): Framing Structure, Channel Coding and Modulation for 11/12 GHz Satellite Services, ETSI- EN 300 421, 1997-98.

LatticeEC/ECP

- HB1000, *LatticeEC/ECP Family Handbook*

LatticeECP2M

- HB1003, *LatticeECP2M Family Handbook*

LatticeECP3

- HB1009, *LatticeECP3 Family Handbook*

LatticeSC/M

- DS1004, *LatticeSC/M Family Data Sheet*

LatticeXP

- HB1001, *LatticeXP Family Handbook*

LatticeXP2

- DS1009, *Lattice XP2 Datasheet*

Revision History

Date	Document Version	IP Versions	Change Summary
—	—	4.0	Previous Lattice releases.
December 2006	02.3	4.1	Updated appendices. Added support for LatticeECP2M device family.
May 2007	02.4	4.2	Updated appendices. Added support for LatticeXP2 device family.
April 2008	02.5	4.3	Updated appendices.
May 2009	02.6	4.4	Updated appendices and added support for the LatticeECP3 device family.
June 2010	02.7	4.5	Added support for Diamond software.
			Divided document into chapters. Added table of contents.
			Added Quick Facts table in Chapter 1, “Introduction.”
			Added new content in Chapter 4, “IP Core Generation.”

Resource Utilization

This appendix gives resource utilization information for Lattice FPGAs using the Block Viterbi Decoder IP core.

IPexpress is the Lattice IP configuration utility, and is included as a standard feature of the Diamond and ispLEVER design tools. Details regarding the usage of IPexpress can be found in the IPexpress and Diamond and ispLEVER help systems. For more information on the Diamond or ispLEVER design tools, visit the Lattice web site at: www.latticesemi.com/software.

LatticeECP and LatticeEC FPGAs

Table A-1. Performance and Resource Utilization[1]

Configuration	Parameters	SLICEs	LUTs	Registers	IOB	sysMEM™ EBRs	f_{MAX} (MHz)
IEEE 802.16a 2004-SC-PHY	See Table 2-4 on page 17.	280	457	232	11	2	126
3GPP	See Table 2-4 on page 17.	5041	9922	3160	13	16	101
DVB-S, IEEE 802.11a	See Table 2-4 on page 17.	1310	2562	864	10	4	106
IEEE 802.16 2004-OFDM PHY (dynamic puncturing)	See Table 2-4 on page 17.	1474	2742	1032	29	4	108
IEEE 802.16 2004-OFDM PHY (fixed puncturing)	See Table 2-4 on page 17.	1735	3254	1185	13	4	108

1. Performance and utilization data are generated targeting an LFEC20E-5F672C device using Lattice Diamond 1.0 and Synplify Pro D-2009.12L-1 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeECP/EC family.

Ordering Part Number

The Ordering Part Number (OPN) for the Block Viterbi Decoder IP on the LatticeEC devices is VTERB-BLK-E2-U4.

LatticeECP2 FPGAs

Table A-2. Performance and Resource Utilization[1]

Configuration	Parameters	SLICEs	LUTs	Registers	IOB	sysMEM EBRs	f_{MAX} (MHz)
IEEE 802.16a 2004-SC-PHY	See Table 2-4 on page 17.	291	469	232	11	2	207
3GPP	See Table 2-4 on page 17.	6345	11747	3160	13	16	138
DVB-S, IEEE 802.11a	See Table 2-4 on page 17.	1636	3017	864	10	4	178
IEEE 802.16 2004-OFDM PHY (dynamic puncturing)	See Table 2-4 on page 17.	1801	3201	1032	29	4	175
IEEE 802.16 2004-OFDM PHY (fixed puncturing)	See Table 2-4 on page 17.	1935	3467	1185	13	4	129

1. Performance and utilization data are generated targeting an LFE2-50E-7F484C device using Lattice Diamond 1.0 and Synplify Pro D-2009.12L-1 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeECP2 family.

Ordering Part Number

The Ordering Part Number (OPNs) for the Block Viterbi Decoder IP on the LatticeECP2 devices is VTERB-BLK-P2- U4.

LatticeECP2M FPGAs

Table A-3. Performance and Resource Utilization[1]

Configuration	Parameters	SLICEs	LUTs	Registers	IOB	sysMEM EBRs	f_{MAX} (MHz)
IEEE 802.16a 2004-SC-PHY	See Table 2-4 on page 17.	291	469	232	11	2	211
3GPP	See Table 2-4 on page 17.	6345	11747	3160	13	16	135
DVB-S, IEEE 802.11a	See Table 2-4 on page 17.	1636	3017	864	10	4	179
IEEE 802.16 2004-OFDM PHY (dynamic puncturing)	See Table 2-4 on page 17.	1801	3201	1032	29	4	176
IEEE 802.16 2004-OFDM PHY (fixed puncturing)	See Table 2-4 on page 17.	1935	3467	1185	13	4	176

1. Performance and utilization data are generated targeting an LFE2M-35E-7F672C device using Lattice Diamond 1.0 and Synplify Pro D-2009.12L-1 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeECP2M family.

Ordering Part Number

The Ordering Part Number (OPNs) for the Block Viterbi Decoder IP on the LatticeECP2M devices is VTERB-BLK-PM-U4.

LatticeECP3 FPGAs

Table A-4. Performance and Resource Utilization[1]

Configuration	Parameters	SLICEs	LUTs	Registers	IOB	sysMEM EBRs	f_{MAX} (MHz)
IEEE 802.16a 2004-SC-PHY	See Table 2-4 on page 17.	285	469	232	11	2	187
3GPP	See Table 2-4 on page 17.	6349	11736	3159	13	16	132
DVB-S, IEEE 802.11a	See Table 2-4 on page 17.	1626	3011	864	10	4	168
IEEE 802.16 2004-OFDM PHY (dynamic puncturing)	See Table 2-4 on page 17.	1768	3191	1032	29	4	171
IEEE 802.16 2004-OFDM PHY (fixed puncturing)	See Table 2-4 on page 17.	1935	3485	1185	13	4	146

1. Performance and utilization data are generated targeting an LFE3-95E-8FN672CES device using Lattice Diamond 1.0 and Synplify Pro D-2009.12L-1 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeECP3 family

Ordering Part Number

The Ordering Part Number (OPNs) for the Block Viterbi Decoder IP on the LatticeECP3 devices is VTERB-BLK-E3-U4.

LatticeSC and LatticeSCM FPGAs

Table A-5. Performance and Resource Utilization[1]

Configuration	Parameters	SLICEs	LUTs	Registers	IOB	sysMEM EBRs	f_{MAX} (MHz)
IEEE 802.16a 2004-SC-PHY	See Table 2-4 on page 17.	263	433	233	11	2	261
3GPP	See Table 2-4 on page 17.	4923	9426	3391	13	16	207
DVB-S, IEEE 802.11a	See Table 2-4 on page 17.	1239	2438	864	10	4	236
IEEE 802.16 2004-OFDM PHY (dynamic puncturing)	See Table 2-4 on page 17.	1389	2617	1032	29	4	230
IEEE 802.16 2004-OFDM PHY (fixed puncturing)	See Table 2-4 on page 17.	1743	3227	1186	13	4	224

1. Performance and utilization data are generated targeting an LFSCM3GA25E-7F900C device using Lattice Diamond 1.0 and Synplify Pro D-2009.12L-1 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeSC/SCM family.

Ordering Part Number

The Ordering Part Number (OPNs) for the Block Viterbi Decoder IP on the LatticeSC/M devices is VTERB-BLK-SC-U4.

LatticeXP FPGAs

Table A-6. Performance and Resource Utilization[1]

Configuration	Parameters	SLICEs	LUTs	Registers	IOB	sysMEM EBRs	f_{MAX} (MHz)
IEEE 802.16a 2004-SC-PHY	See Table 2-4 on page 17.	280	457	232	11	2	116
3GPP	See Table 2-4 on page 17.	5041	9922	3160	13	16	92
DVB-S, IEEE 802.11a	See Table 2-4 on page 17.	1310	2562	864	10	4	101
IEEE 802.16 2004-OFDM PHY (dynamic puncturing)	See Table 2-4 on page 17.	1474	2742	1032	29	4	104
IEEE 802.16 2004-OFDM PHY (fixed puncturing)	See Table 2-4 on page 17.	1735	3254	1185	13	4	100

1. Performance and utilization data are generated targeting an LFXP20E-5F256C device using Lattice Diamond 1.0 and Synplify Pro D-2009.12L-1 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeXP family.

Ordering Part Number

The Ordering Part Number (OPNs) for the Block Viterbi Decoder IP on the LatticeXP devices is VTERB-BLK-XM-U4.

LatticeXP2 FPGAs

Table A-7. Performance and Resource Utilization[1]

Configuration	Parameters	SLICEs	LUTs	Registers	IOB	sysMEM EBRs	f_{MAX} (MHz)
IEEE 802.16a 2004-SC-PHY	See Table 2-4 on page 17.	291	469	232	11	2	183
3GPP	See Table 2-4 on page 17.	6345	1147	3160	13	16	128
DVB-S, IEEE 802.11a	See Table 2-4 on page 17.	1636	3017	864	10	4	160
IEEE 802.16 2004-OFDM PHY (dynamic puncturing)	See Table 2-4 on page 17.	1801	3201	1032	29	4	153
IEEE 802.16 2004-OFDM PHY (fixed puncturing)	See Table 2-4 on page 17.	1935	3467	1185	13	4	136

1. Performance and utilization data are generated targeting an LFXP2-17E-7F484C device using Lattice Diamond 1.0 and Synplify Pro D-2009.12L-1 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeXP2 family.

Ordering Part Number

The Ordering Part Number (OPNs) for the Block Viterbi Decoder IP on the LatticeXP2 devices is VTERB-BLK-X2-U4.