Block Convolutional Encoder User's Guide

June 2010
IPUG31_03.5

Table of Contents

Lattice's Block Convolutional Encoder IP core is a parameterizable core for convolutional encoding of continuous or burst input data streams. The core allows different code rates and constraint lengths and supports puncturing. It can operate in continuous or block mode, whichever is required by the channel. In block mode, either Zero Flushing or Tail Biting codes can be generated. All the configurable parameters, including operation mode, termination mode, generator polynomials, code rate, and puncture pattern, can be defined by the user to suit the needs of the application. The code rate and the puncture pattern can also be varied through the input ports dynamically, providing further flexibility for the IP usage. Lattice's Block Convolutional Encoder IP core is compatible with many networking and wireless standards that use convolutional encoding.

Quick Facts

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

Table 1-1. Block Convolutional Encoder IP Core for LatticeEC/ECP/XP Devices Quick Facts

		Block Convolutional Encoder IP Configuration				
		Puncture Rate 2/3 Constraints 3 Block 802.16- 2004 SC PHY	**Non-punc-ture Rate 1/2 Constraints 9 Block 3GPP/ CDMA2000**	**Non-punc-ture Rate 1/2 Constraints 7 Continuous 802.11a, also DVB-S**	**Dynamic puncture Max rate 5/6 Constraints 7 Block 802.16-2004 OFDM PHY**	**Puncture Rate 3/4 Constraints 7 Continuous 802.11a, also DVB-S**
Core Requirements	FPGA Families Supported	Lattice EC/ECP/XP				
	Minimal Device Needed	LFEC1E/LFECP6E/LFXP3E				
Resource Utilization	Targeted Device	LFEC20E-5F672C/LFECP20E-5F672C/LFXP20E-5F484C				
	LUTs	50	50	50	150	50
	sysMEM EBRs	0				
	Registers	50	50	50	150	100
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 Convolutional Encoder IP Core for LatticeECp2/ECP2M/XP2 Devices Quick Facts

		Block Convolutional Encoder IP Configuration				
		Puncture Rate 2/3 Constraints 3 Block 802.16- 2004 SC PHY	**Non-punc-ture Rate 1/2 Constraints 9 Block 3GPP/ CDMA2000**	**Non-punc-ture Rate 1/2 Constraints 7 Continuous 802.11a, also DVB-S**	**Dynamic puncture Max rate 5/6 Constraints 7 Block 802.16-2004 OFDM PHY**	**Puncture Rate 3/4 Constraints 7 Continuous 802.11a, also DVB-S**
Core Requirements	FPGA Families Supported	Lattice ECP2/ECP2M/XP2				
	Minimal Device Needed	LFE2-6E/ LFE2M20E/ LFXP2-5E				
Resource Utilization	Targeted Device	LFE2-50E-7F672C/LFE2M35E-7F484C/LFXP2-17E-7F484C				
	LUTs	50	50	50	150	50
	sysMEM EBRs	0				
	Registers	50	50	50	150	100
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 Convolutional Encoder IP Core for LatticeSC/SCM Devices Quick Facts

		Block Convolutional Encoder IP Configuration				
		Puncture Rate 2/3 Constraints 3 Block 802.16- 2004 SC PHY	**Non-punc-ture Rate 1/2 Constraints 9 Block 3GPP/ CDMA2000**	**Non-punc-ture Rate 1/2 Constraints 7 Continuous 802.11a, also DVB-S**	**Dynamic puncture Max rate 5/6 Constraints 7 Block 802.16-2004 OFDM PHY**	**Puncture Rate 3/4 Constraints 7 Continuous 802.11a, also DVB-S**
Core Requirements	FPGA Families Supported	Lattice SC/SCM				
	Minimal Device Needed	LFSC3GA15E/ LFSCM3GA15EP1				
Resource Utilization	Targeted Device	LFSC3GA25E-7F900C/ LFSCM3GA25EP1-7F900C				
	LUTs	50	50	50	150	50
	sysMEM EBRs	0				
	Registers	50	50	50	150	100
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 Convolutional Encoder IP Core for LatticeECP3 Devices Quick Facts

		Block Convolutional Encoder IP Configuration				
		Puncture Rate 2/3 Constraints 3 Block 802.16- 2004 SC PHY	**Non-punc-ture Rate 1/2 Constraints 9 Block 3GPP/ CDMA2000**	**Non-punc-ture Rate 1/2 Constraints 7 Continuous 802.11a, also DVB-S**	**Dynamic puncture Max rate 5/6 Constraints 7 Block 802.16-2004 OFDM PHY**	**Puncture Rate 3/4 Constraints 7 Continuous 802.11a, also DVB-S**
Core Requirements	FPGA Families Supported	Lattice ECP3				
	Minimal Device Needed	LFE3-95E-8FN672CES				
Resource Utilization	Targeted Device	LFE3-35EA				
	LUTs	50	50	50	150	50
	sysMEM EBRs	0				
	Registers	50	50	50	150	100
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				

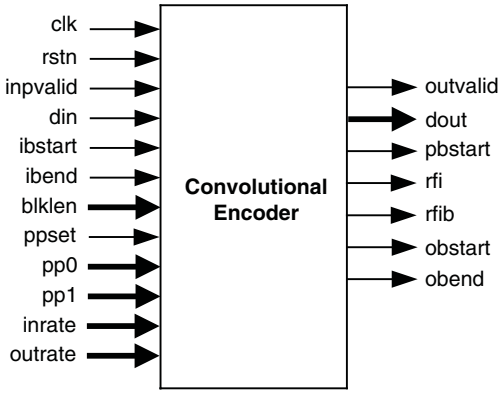
Features

- Compatible with IEEE 802.16-2004, IEEE 802.11a, 3GPP, 3GPP2 and DVB standards
- Supports both continuous and block encoding
- Variable constraint length from 3 to 9
  - Supports both Zero Flushing and Tail Biting termination modes
  - Supports both internal and external zero padding in Zero Flushing mode
- Supports both internal and external tail adding in Tail Biting mode
- Supports a wide range of programmable code rates (input_rate/output_rate)
- User defined generator polynomials
- Output puncturing with unrestricted, user programmable puncture patterns
- Supports dynamic puncturing mode, in which both the code rate and puncture patterns can be varied through ports
- Punctured code rate can be programmed to k/n, where k can be from 2 to 12 and n can be from k+1 to 2k-1; additionally, rate 1/2 is supported in dynamic puncture mode
- Handshake signals to support breaks in data stream or encoder busy conditions

Functional Description

This chapter provides a functional description of the Block Convolutional Encoder IP core. Figure 2-1 shows the interface diagram for Block Convolutional Encoder.

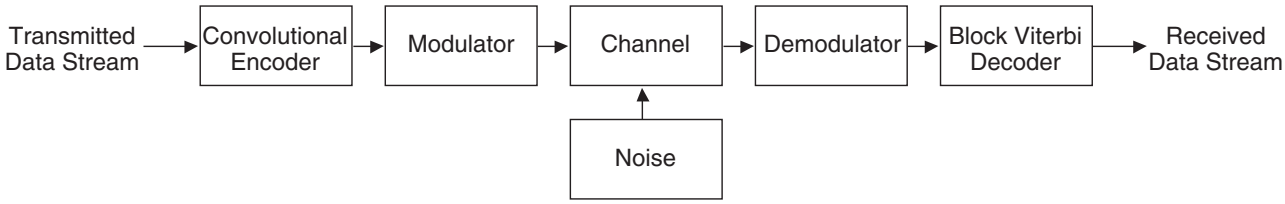
Figure 2-1. Block Convolutional Encoder Interface Diagram



General Description

Figure 2-2 shows a digital communication system using the Convolutional Encoder. The digital data stream (such as voice, image or any packetized data) is first convolutionally encoded, then modulated and finally transmitted through a channel. The noise block in Figure 2-2 represents channel noise added to the channel. The data received from the channel at the receiver side is first demodulated and then decoded using a Viterbi decoder. The decoded output is equivalent to the original transmitted data stream.

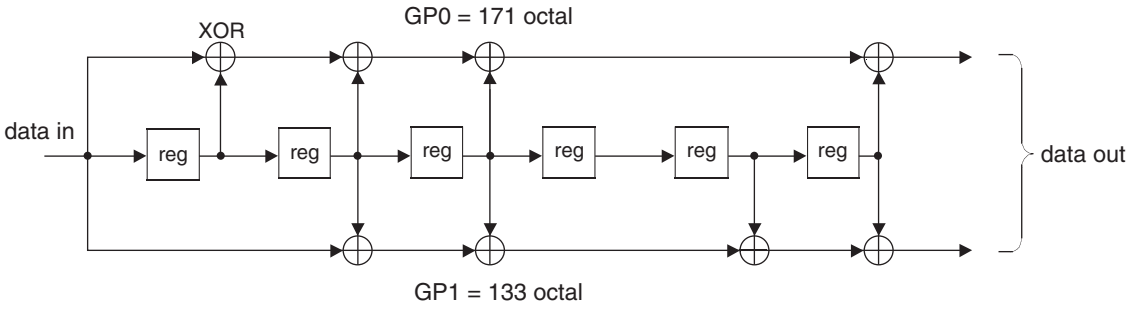
Figure 2-2. Digital Communication System



Convolutional Encoding

Convolutional encoding is a process of adding redundancy to a signal stream to provide error correction capability. Figure 2-3 shows an example of 1/2 rate convolutional encoding.

Figure 2-3. Convolutional Encoding

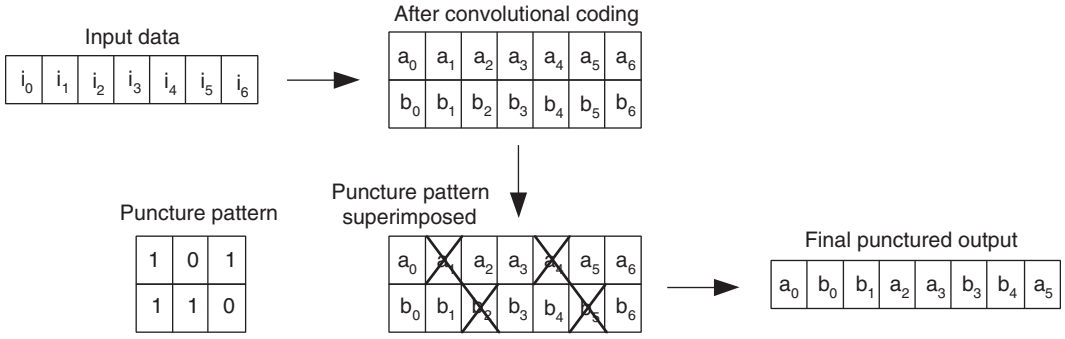


In this example, each input symbol has two corresponding output symbols; hence, the encoding is called 1/2 rate convolutional encoding. To generate the output, the encoder uses seven values of the input signal: one present and six past. The set of past values of input data is called a “state.” The number of input data values used to generate the code is called the constraint length. In this case, the constraint length is 7. Each set of outputs is generated by XORing 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 data bit and a value of '0' signifies an unused bit.

Punctured Codes

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. A pair of binary strings called a “puncture pattern” is used to make the selection of punctured symbols. A “1” in the pattern means the corresponding symbol is kept in the output stream, while a “0” means the symbol in that position is removed. Figure 2-4 shows an example of puncturing.

Figure 2-4. Puncturing Process



If puncturing is employed in the encoder, the decoder will have to “depuncture” the data before decoding. Depuncturing is usually done by inserting NULL symbols for the punctured symbols. NULL symbols are equidistant from both '0' and '1'.

Similar to non-punctured codes, the rate of a punctured code is defined as k/n, where k is the input symbol rate and n is the output symbol rate. The input rate and output rate can be easily recognized by looking at the number of columns and the number of “1”s in the puncture pattern, respectively. As an example, the above Figure 2-4 gives a punctured code with rate 3/4.

For punctured encoding, the code rate and the puncture pattern can be defined in two ways with the Block Convolution Encoder IP core. It can be either set statically using the GUI or specified dynamically through the input ports. The former mode is referred as “fixed puncturing” and the latter as “dynamic puncturing.”

Continuous and Block Encoding

The convolutional encoding process can be applied on either a continuous stream or blocks of input data. When the input data stream is continuous, the encoder is configured to continuous mode. On the other hand, if the input data stream is block based (or frame based), the core is set for block encoding. The major difference between the continuous and block encoding is the termination method that is used for block codes.

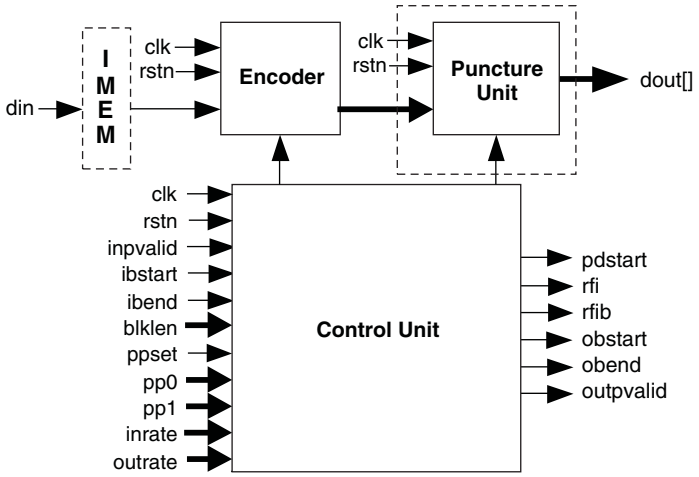
Zero Flushing and Tail Biting Termination Modes

In block encoding, the code must be terminated appropriately so that the decoding process can start from a suitable initial state. The IP core supports two block termination 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.

Functional Description

A simplified architecture diagram of the Block Convolutional Encoder IP core is shown in Figure 2-5. It consists of an encoder, a puncture unit, a control unit, and an optional input memory.

Figure 2-5. Block Convolutional Encoder Internal Architecture



Encoder

The encoder module takes input data and performs convolutional encoding. The encoder uses generator polynomials configured by the user. When punctured encoding is enabled, the encoder performs 1/2 rate encoding irrespective of the encoder rate. The puncture unit uses the 1/2 rate code to generate the appropriate user-programmed rate.

The initial state of the encoder is related to the configuration settings of the IP core, as shown in Table 2-1:

Table 2-1. Initial State of the Convolutional Encoder

Configuration Settings	Initial State
Continuous encoding	0
Block encoding, Zero Flushing code, zero added inside	0
Block encoding, Zero Flushing code, zero added outside	0
Block encoding, Tail Biting code, tail added inside	Last K-1 samples in the block, where K is the constraint length
Block encoding, Tail Biting code, tail added outside	Undefined

Both Zero Flushing and Tail Biting for block encoding can be performed either inside or outside the IP core. If the outside mode is selected, it is the user's responsibility to append the information with the initial states. If the inside mode is selected, the core will generate the zero padding bits after the block for Zero Flushing mode, or initialize the state with the tail information before the block for Tail Biting mode.

Puncture Unit

This unit performs data puncturing, as previously explained. The input is a two-channel data stream and the output is always a one-channel output. The unit is capable of performing puncturing of any code rate with a user programmable puncture pattern.

Input Memory

For “tail adding inside” mode, the core has to store the whole message block and use the last K-1 bits to initialize the registers, where K is the constraint length. This is achieved by having an input memory module inside the core.

Control Unit

The control unit generates the handshake signals `obstart`, `obend`, `outvalid`, `rfi`, `rfib` and `pbstart` using the input signals `inpvalid`, `ibstart`, `ibend` and the status of the encoder. In dynamic puncture mode, it contains registers that latch the `inrate`, `outrate`, `pp0` and `pp1` when `ppset` signal is asserted. The control unit also generates various control signals required by the encoder and the puncture unit for different continuous and block encoding schemes.

Interfacing to the Block Convolutional Encoder Core

The puncturing-enabled convolutional encoder is a multi-rate system, with the output rate greater than the input rate. The data rate mismatch between input and output can be managed by using the output signal `rfi` (ready for input). The driving system should not apply an input to the encoder if the `rfi` output is low (if this is done, the data will be ignored until `rfi` is high). When valid data is applied at the input `din`, input `inpvalid` must be asserted high. Even if the `rfi` output is high, the driving system can blackout the input by pulling inpvalid low. The core will optimize throughput by using up a portion of any user asserted blackout cycles as wait cycles (for data-rate matching).

When the core works under block mode, the start and end of a block is defined either by the `ibstart` and `ibend` signals, or by `ibstart` and `blklen` input signals. In the latter case, the core has an internal counter to generate the `ibend` signal counting from the input `ibstart`. However, if the tail-biting termination mode is selected and the tail-adding is implemented inside, the block length is always read from the `blklen` input port. This is because the whole block of data must be stored in the core before encoding and memory has to be provided based on the block length.

The output signal `pbstart` is asserted high to coincide with the start of a punctured block. This signal can be used to synchronize the Viterbi decoder that is decoding the encoded stream.

The output control signal `outvalid` is high whenever the output is valid. This can be used as an enable signal to latch the output to a memory.

For block encoding, the output signal rfib is asserted high to inform the input source that the core is ready to accept the next block of data. The output control signal obstart and obend are used to inform the output destination the start and end of each block.

For the dynamic puncturing encoder, the ppset signal is used to load the valid inrate, outrate, pp0 and pp1 values that are applied on those ports. These data have to be loaded-in at least five clock cycles before the start of next data block, for them to be effective.

Signal Descriptions

A description of I/O interface signals is given in Table 2-2. Refer to Figure 2-1 for a top-level diagram of the IP core.

Table 2-2. Signal Descriptions

Port	Bits	I/O	Description
clk	1	I	System clock
rstn	1	I	System wide asynchronous active-low reset signal.
inpvalid	1	I	Input valid signal to denote valid data being presented at the encoder input. For Punctured encoding, it must be asserted only if the encoder output rfi is high.
din	1	I	Input data to the encoder: For Punctured encoding, it must be presented only if the encoder output rfi is high.
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. Also, this signal can only be asserted if the encoder output rfib is high. This port is available for block encoding only.
ibend	1	I	Input block end signal: This signal must be pulled high to indicate that the last data of a block is being applied on the input port. This port is available for block encoding only. This signal is not available if the block length is read from port blklen.
blklen	4 to 16 or 4 to 9	I	The length of input data block is applied at this port: The width of this port can be selected from 4 to 16 bits for zero-flushing code, and from 4 to 9 bits for tail-biting code. The value on this port is read only when ibstart is high. This port is available in the “Block Length Read from Port” mode only.
inrate	1-4	I	Input rate of the convolutional code for next block: This port is available for dynamic puncture only
outrate	2-5	I	Output rate of the convolutional code for next block: This port is available for dynamic puncture only
pp0	1-12	I	Puncture pattern 0 for next block: This port is available for dynamic puncturing encoder only
pp1	1-12	I	Puncture pattern 1 for next block: This port is available for dynamic puncturing encoder only
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 for dynamic puncturing encoder only
outvalid	1	O	Output valid signal: This indicates that the output on dout is a valid encoded data.
pbstart	1	O	This output signal goes high whenever the encoder outputs the first data of a punctured block and signifies the start of a punctured block.
obstart	1	O	Output block start signal: It goes high when the first data of a block is on the dout output port. This port is available for block encoding only.
obend	1	O	Output block end signal: It goes high when the last data of a block is on the dout output port. This port is available for block encoding only.
rfib	1	O	Ready for next input block: This signal goes high to indicate that the core is ready for reading the next input block. This port is available for block encoding only.

Table 2-2. Signal Descriptions (Continued)

Port	Bits	I/O	Description
rfi	1	O	This port is available only for punctured encoder. This signal signifies that the encoder is ready for input. If rfi goes low, the encoder will not accept input data in the next clock edge.
dout	2 to 8 (non-puntured) 1 (punctured)	O	Output data of the encoder: The data is valid only if the output outvalid is high.

Timing Diagrams

The top-level timing diagrams for different cases are given in Figure 2-6 through Figure 2-13.

Figure 2-6. Timing Diagram for a Rate 1/2 Continuous Non-punctured Encoder

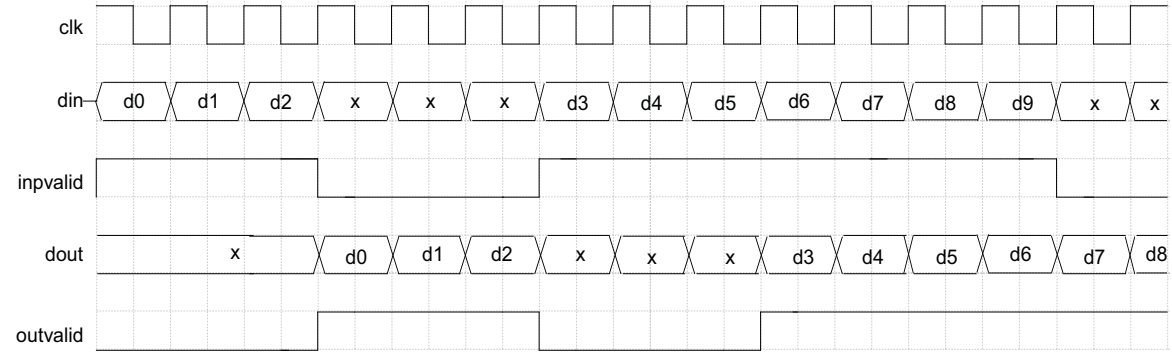


Figure 2-7. Timing Diagram for a Rate 3/5 Continuous Punctured Encoder

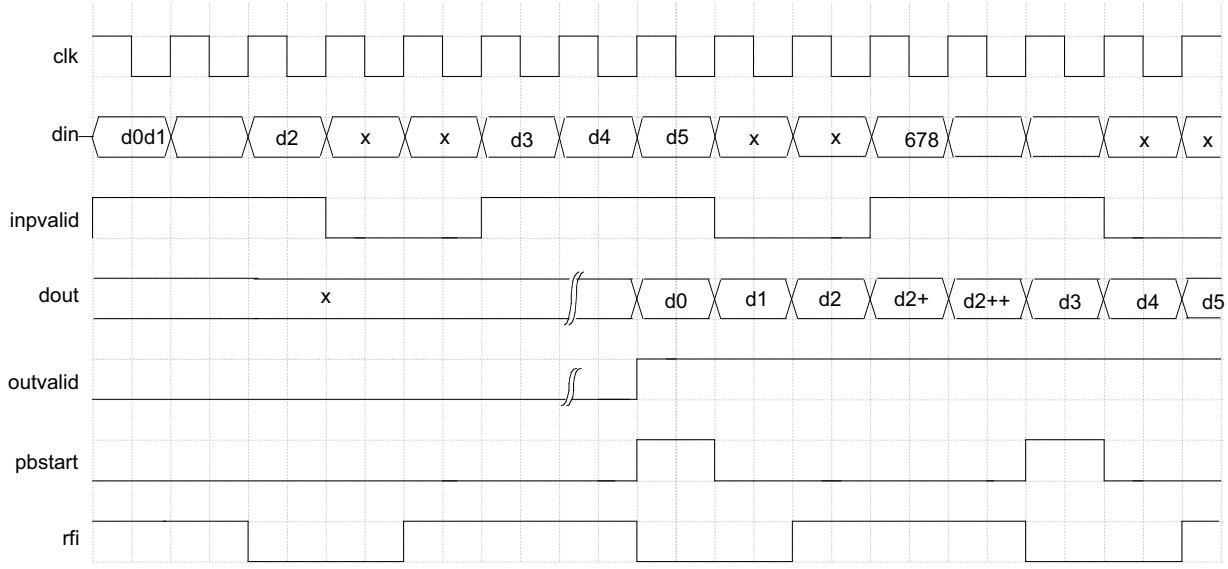


Figure 2-8. Timing diagram for a Rate 1/2 Block Non-punctured Encoder with Zero Flushing

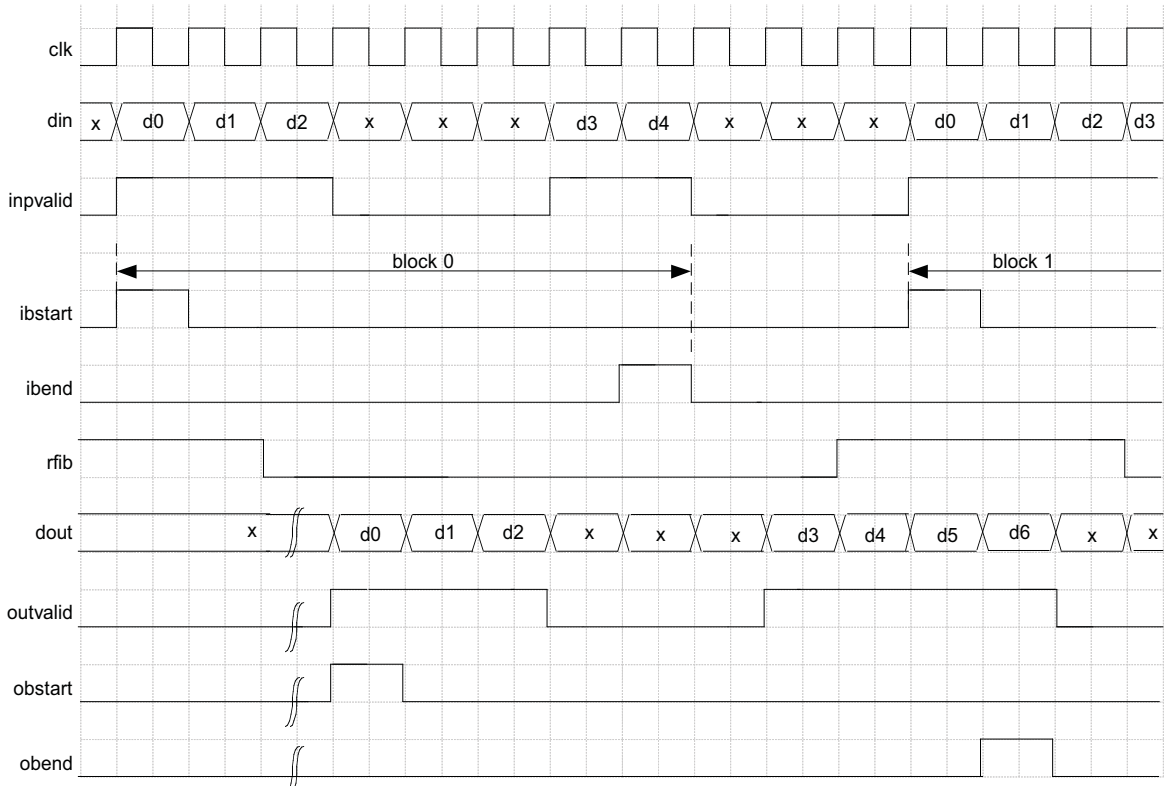


Figure 2-9. Timing Diagram for a Rate 3/5 Block Punctured Encoder with Tail Adding Outside

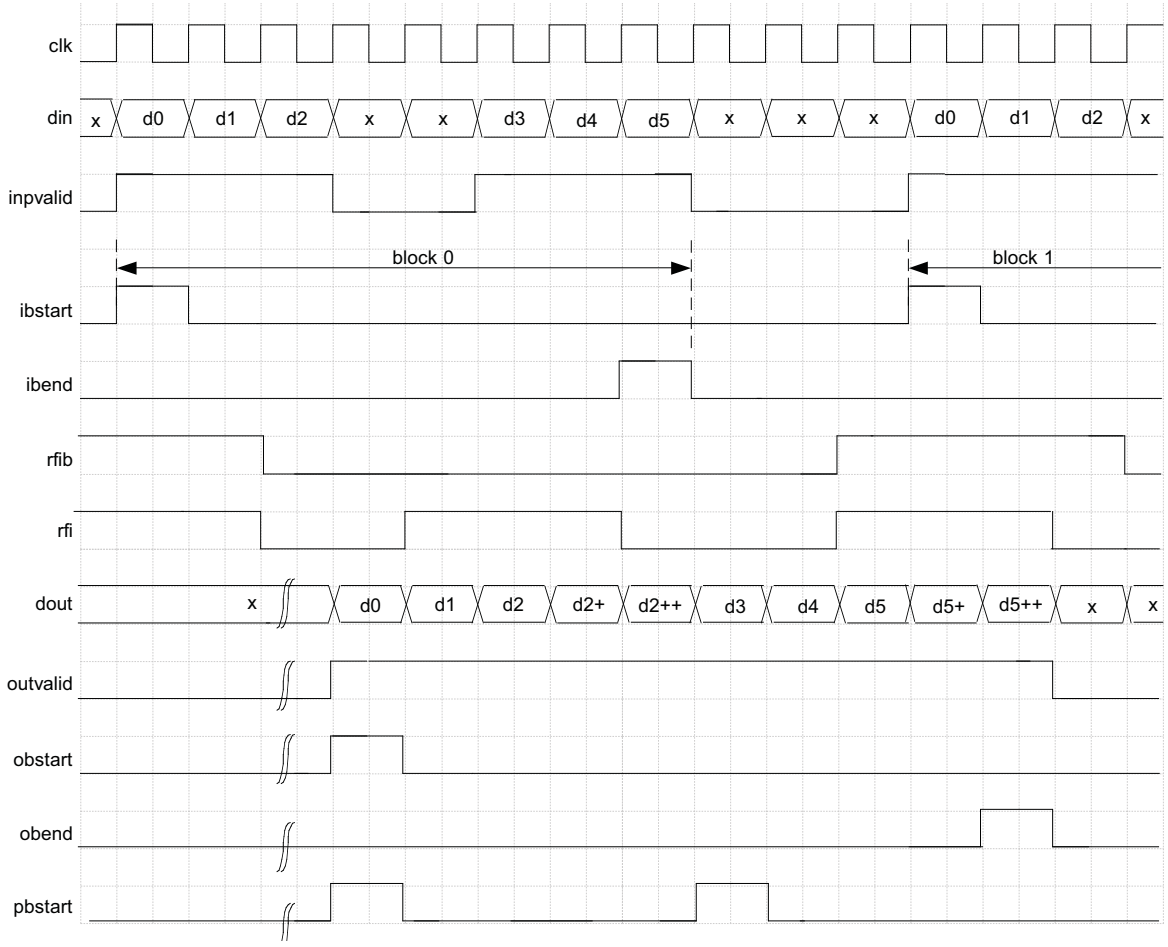


Figure 2-10. Timing Diagram for a Rate 1/2 Block Non-punctured Encoder with Tail Adding Inside

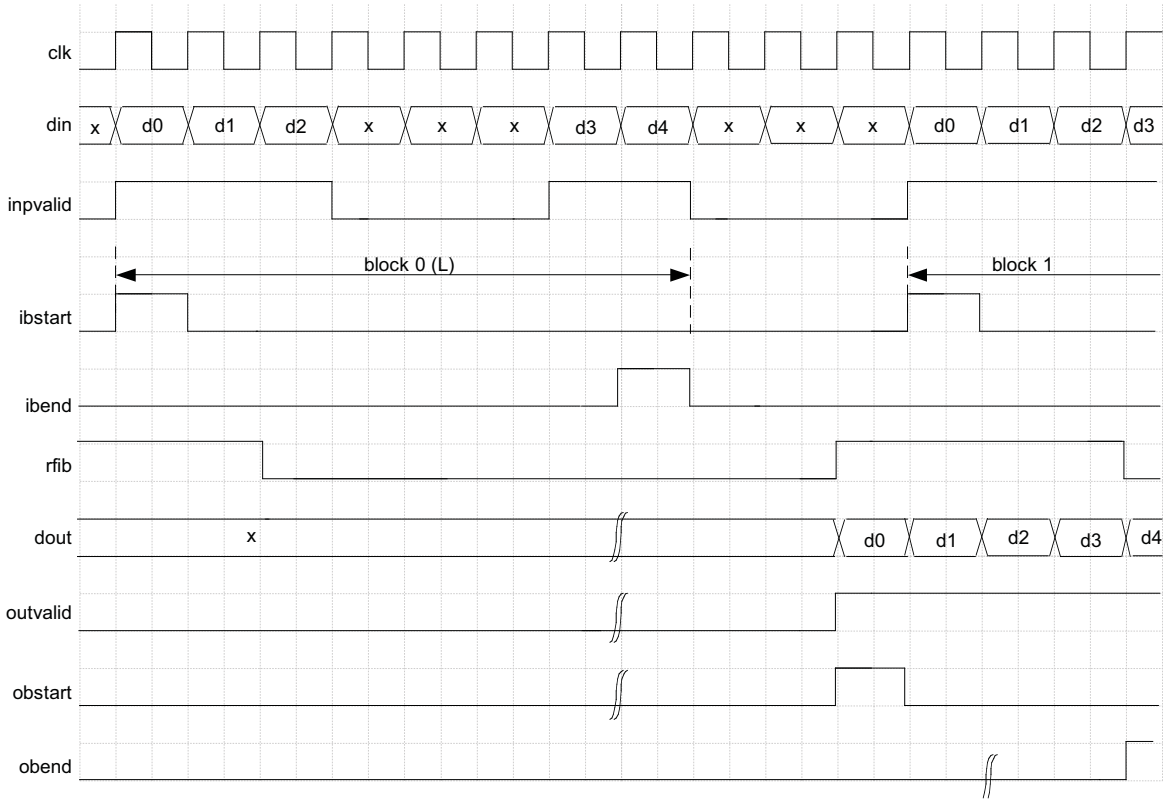


Figure 2-11. Timing Diagram for a Dynamic Punctured Encoder with Zero Termination Outside (First Input Block After System Reset)

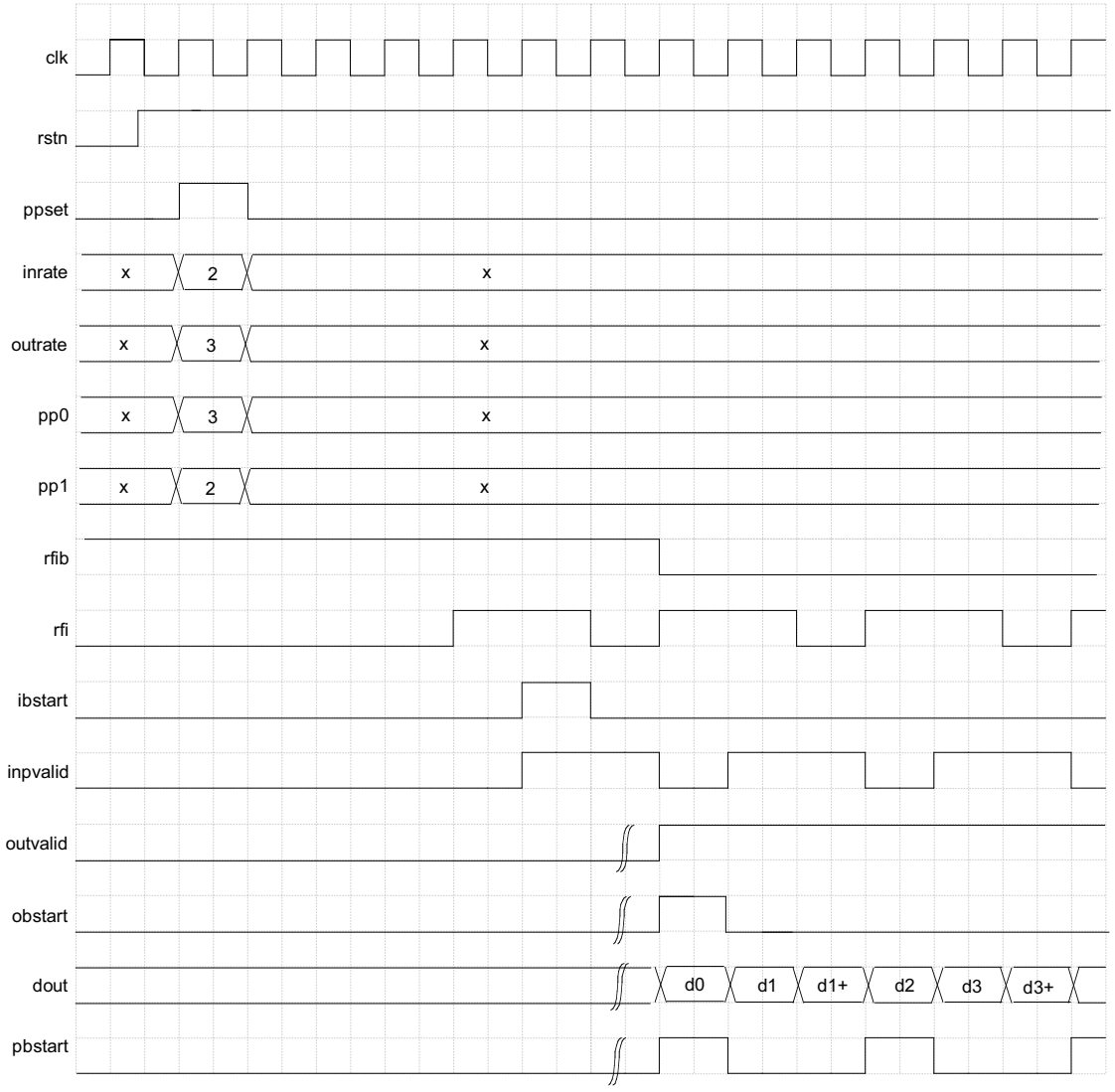


Figure 2-12. Timing Diagram for a Dynamic Punctured Encoder with Zero Termination Outside (The ppset Asserted After the Previous Block Coding is Finished)

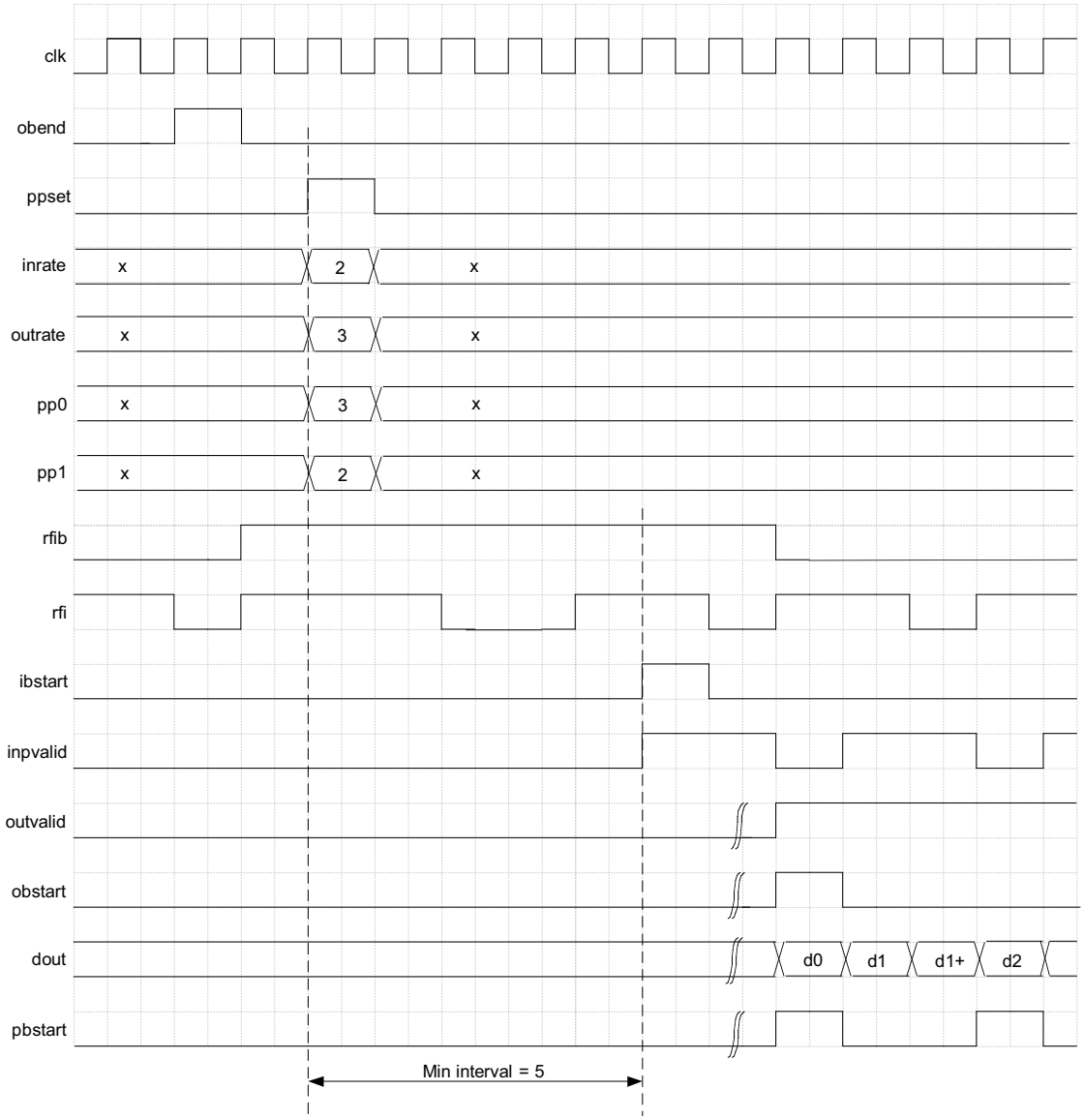
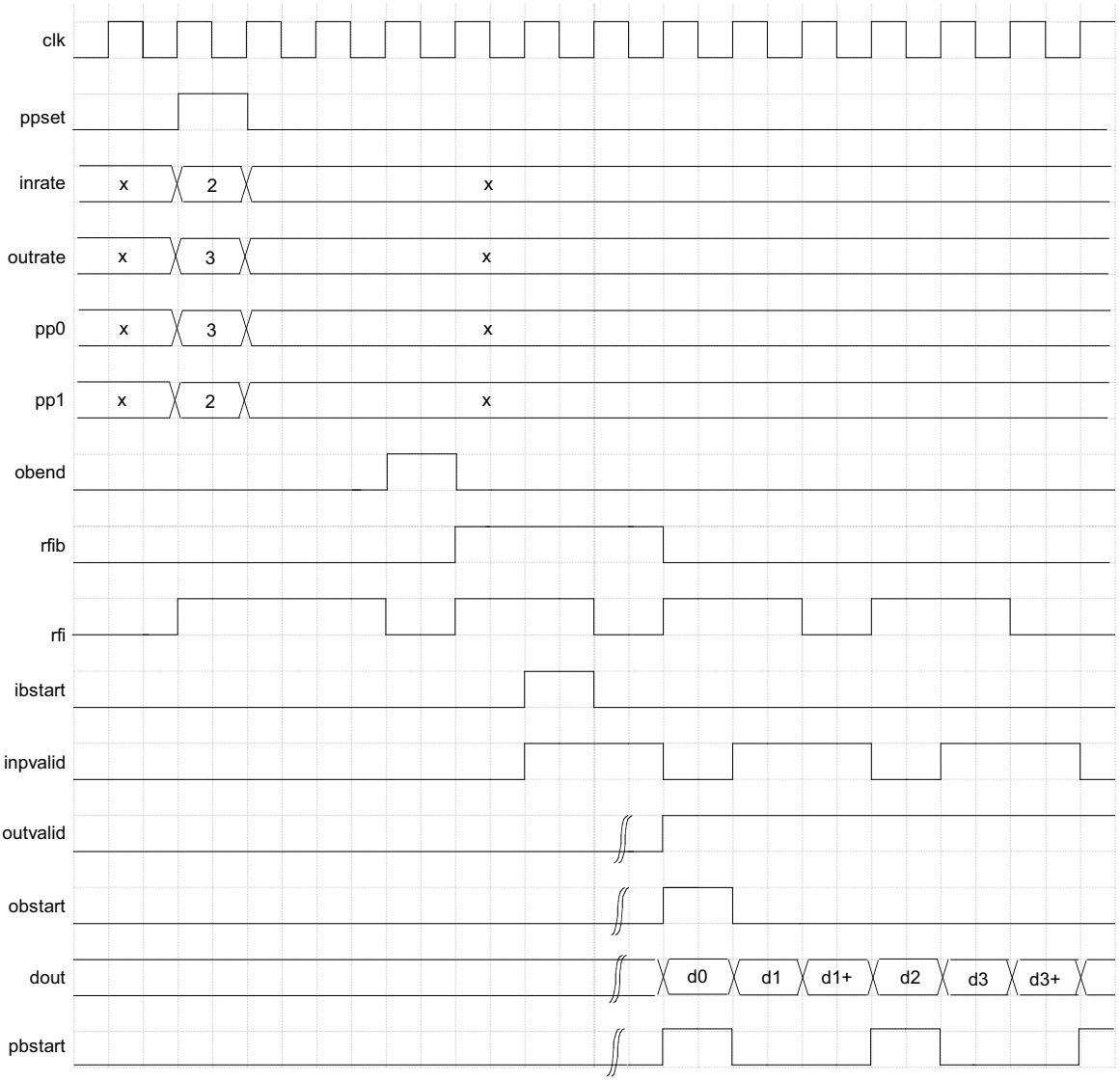


Figure 2-13. Timing Diagram for a Dynamic Punctured Encoder with Zero Termination Outside (The ppset Asserted Before the Previous Block Coding is Finished)



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 Convolutional Encoder IP core. The parameter settings are specified using the Block Convolutional Encoder IP core Configuration GUI in IPexpress. The numerous PCI Express parameter options are partitioned across multiple GUI tabs as shown in this chapter.

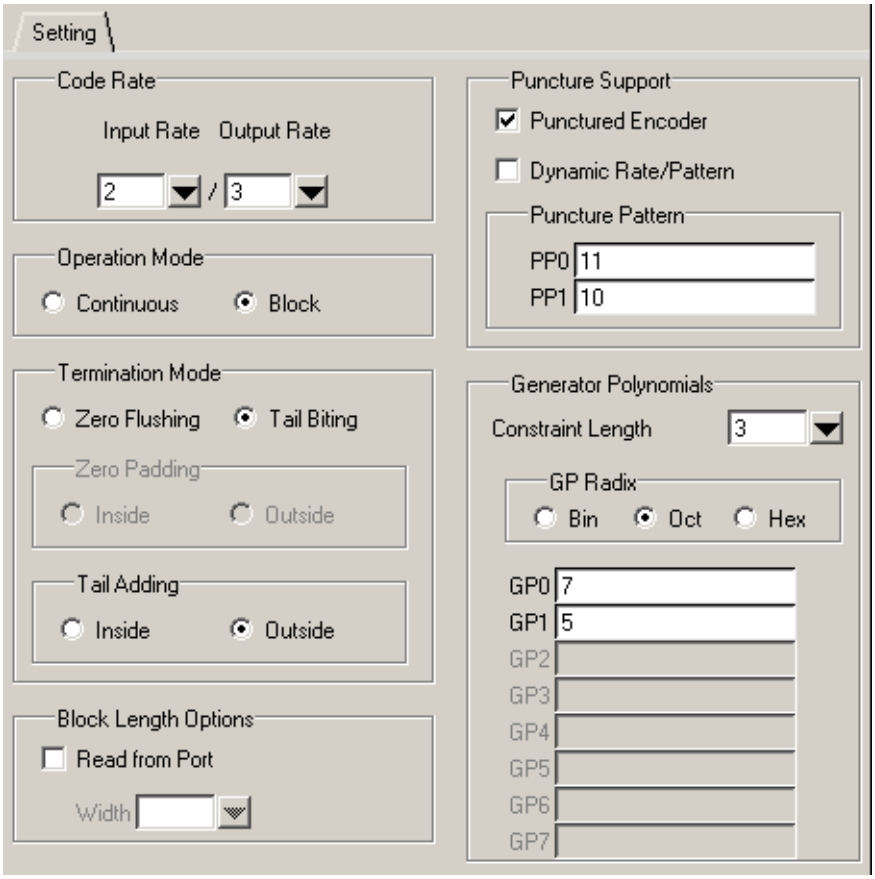
Table 3-1. Block Convolutional Encoder Parameter Descriptions

Parameter	Range	Default
Code Rate		
Input Rate (k) (Maximum Input Rate(mir))	1 (non-puncture) or 2 to 12 (fixed puncture) for "Input Rate(k)" 1 to 12 (dynamic puncture) for "Maximum Input Rate(mir)"	2 for **k** 5 for mir
Output Rate (n) (Maximum Output Rate (mor))	2 to 8 (non-puncture) or k+1 to 2k-1(fixed puncture) for "Output Rate(n)" mir+1 to 2*mir-1 (dynamic puncture; mir>1) or 2 (dynamic puncture; mir=1) for "Maximum Output Rate(mor)"	3 for **n** 6 for mor
Puncture Support		
Punctured Encoder	Enabled, Disabled	Enabled
Dynamic Rate/Pattern	Enabled, Disabled	Disabled
Puncture Pattern (PP0, PP1)	k bits for each	11 (PP0) 10 (PP1)
Operation Mode		
Operation Mode	Continuous, Block	Block
Termination Mode		
Termination Mode	Zero Flushing, Tail Biting	Tail Biting
Zero Padding Mode	Inside, Outside	Outside
Tail Adding Mode	Inside, Outside	Outside
Block Length Options		
Block Length Read From Port	Enabled. Disabled	Disabled
Block Length Width	4-16 for zero-flushing termination block or 4-9 tail-biting termination block	9
Generator Polynomials		
Constraint Length (K)	3 to 9	3
GP Radix	Bin, Oct, Hex	Oct
GP0	k bits	7
GP1	k bits	5
GP2	k bits	NA
GP3	k bits	NA
GP4	k bits	NA
GP5	k bits	NA
GP6	k bits	NA
GP7	k bits	NA

Block Convolutional Encoder Parameters

Figure 3-1 shows the contents of the Block Convolutional Encoder configuration parameters.

Figure 3-1. Block Convolutional Encoder Configuration Parameters



Code Rate

The code rate of the convolutional encoder is expressed using two values: input rate (numerator of the code rate) and output rate (denominator of the code rate).

Input Rate (Max Input Rate)
When the the Puncture Support "Dynamic Rate/Pattern" parameter is disabled, this parameter defines the input rate of the encoder for non-puncture and fixed puncture modes. The rate is equal to 1 for non-punctured codes and should be between 2 and 12 for fixed punctured codes.

When "Dynamic Rate/Pattern" is enabled, this parameter specifies the Max Input Rate. It should be between 1 and 12.

Output Rate (Max Output Rate)
When the Puncture Support "Dynamic Rate/Pattern" parameter is disabled, this parameter defines the output rate of the encoder for non-puncture and fixed puncture modes. It can be from 2 to 8 for non-punctured codes, and from k+1 to 2k-1 for punctured codes.

When "Dynamic Rate/Pattern" is enabled, this parameter specifies the Max Output Rate. Its value should be from mir+1 to 2*mir-1 when mir is greater than 1; and equal to 2 if mir is 1.

Operation Mode

This parameter determines the operation mode of the core. The core works in either continuous mode or block mode.

Puncture Support

Punctured Encoder
This parameter determines whether the IP core supports punctured (Enabled) or non-punctured (Disabled) output.

Dynamic Rate/Pattern
This parameter defines whether the core supports dynamic puncturing encoding (Enabled) or fixed puncturing encoding (Disabled).

Puncture Pattern
This parameter allows the user to set the puncture pattern for punctured encoders. This parameter is only valid for fixed puncturing encoding. The puncture pattern composed of PP0 and PP1 is defined by the user. The total number of 1's in both puncture patterns must equal the output rate, and the number of bits for each puncture pattern must equal the input rate

Termination Mode

This parameter determines the termination mode of block convolutional code. It can be Tail-biting mode or Zero-Flushing mode. For block decoding only.

Zero Padding Mode
This parameter determines the zero padding mode. It can be supported inside or outside of the IP core. This parameter is valid for for Block Encoding and Zero-Flushing modes only.

Tail Adding Mode
This parameter determines the tail adding mode. It can be supported inside or outside of the IP core. This parameter is valid for Block Encoding and Tail-Biting modes only.

Block Length Options

Block Length Read From Port
When this parameter is enabled, the block length is read from an external port. This parameter is always enabled when Tail Adding Mode is set to Inside. This parameter is valid for block encoding only.

Block Length Width
This parameter determines the width of the block length port. For zero-flushing mode, it can be from 4 to 16 bits. For tail-biting mode, it can be from 4 to 9 bits. This parameter is only valid when the Block Length Read from Port is enabled.

Generator Polynomials

GP0, GP1, GP2, GP3, GP4, GP5, GP6 and GP7 are generator polynomials. For non-puncturing encoders, the number of generator polynomials is always equal to the output rate. For puncturing encoders, the number of generator polynomials is 2. The polynomial values can be provided in any of the three radixes: binary ("Bin"), octal ("Oct"), or hexadecimal ("Hex") which defined by parameter "GP Radix".

Constraint Length
This parameter defines the constraint register length. The value can be any integer from 3 to 9.

GP Radix
This parameter sets the input radix of the generator polynomials.

IP Core Generation

This chapter provides information on how to generate the Block Convolutional Encoder 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 Convolutional Encoder IP core in 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 Convolutional Encoder IP core and fully evaluate the core through functional simulation and implementation (synthesis, map, place and route) without an IP license. The Block Convolutional Encoder IP core also 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 28 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.

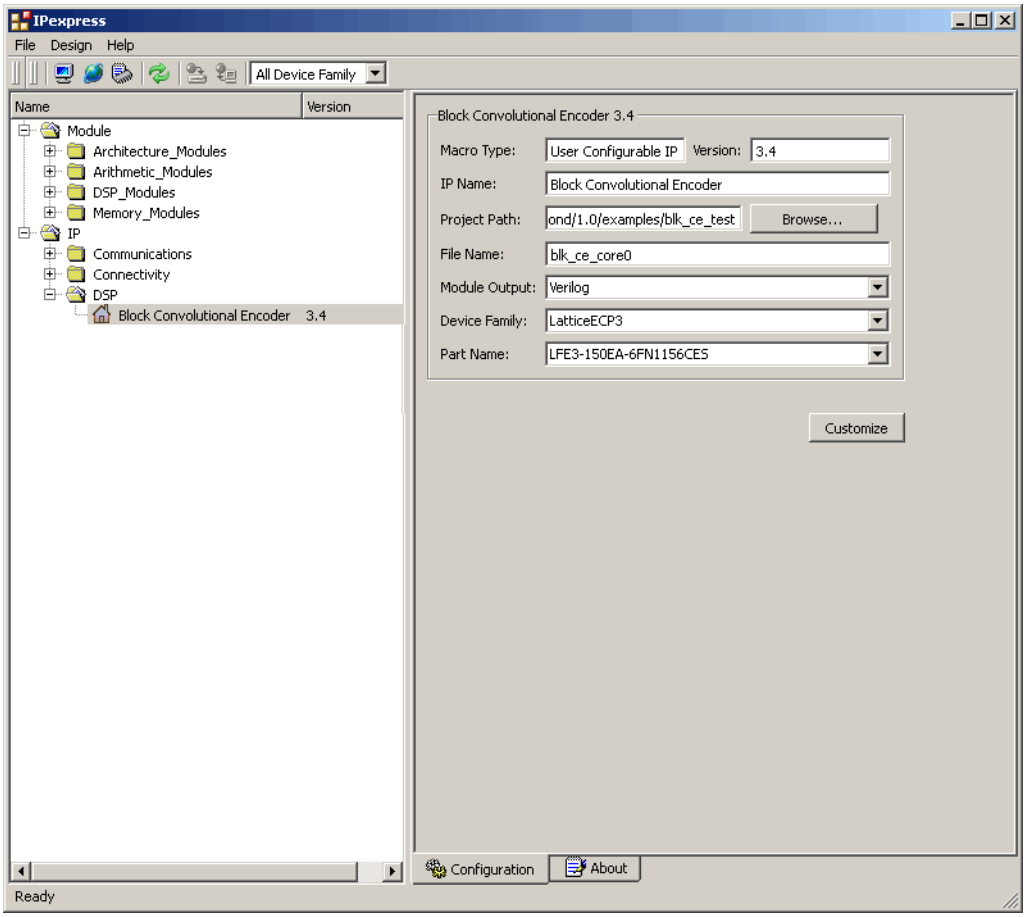
Getting Started

The Block Convolutional Encoder IP core is available for download from the 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 Diamond or ispLEVER IPexpress tool GUI dialog box for the Block Convolutional Encoder 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.

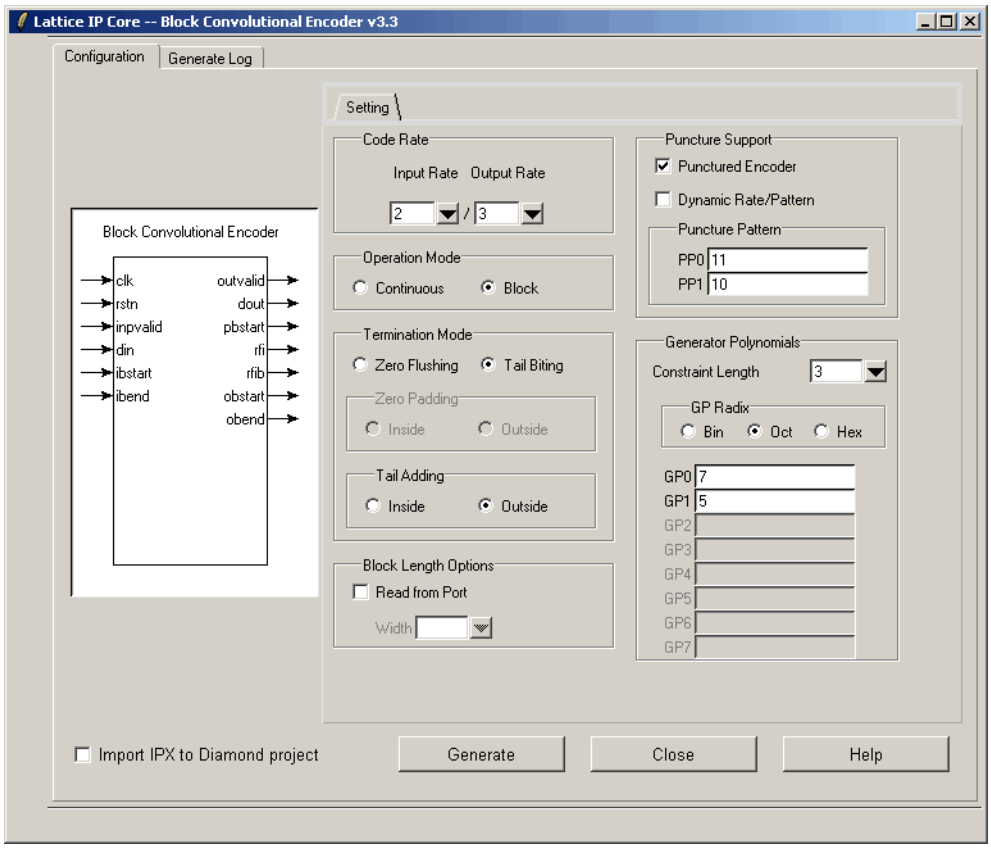
Figure 4-1. The IPexpress Tool Dialog Box (Diamond Version)



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 Convolutional Encoder IP core Configuration 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 19for more information on the Block Convolutional Encoder IP core parameter settings.

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



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 Convolutional Encoder IP core Directory Structure

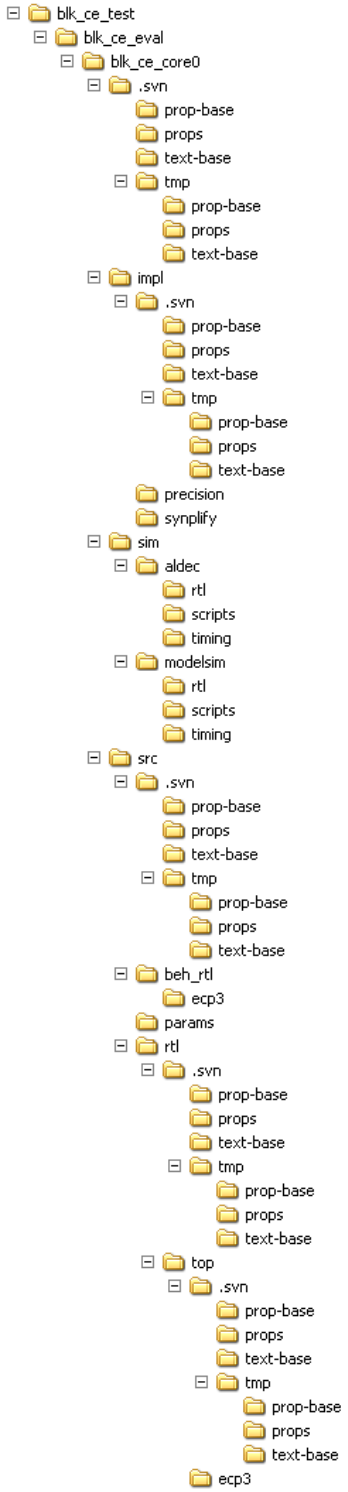


Table 4-1 provides a list of key files and directories created by the IPexpress tool and how they are used. The IPexpress tool creates several files that are used throughout the design cycle. The names of most of the created files are customized to the user's module name specified in the IPexpress tool.

Table 4-1. File List

File	Description
<*username*>_inst.v	This file provides an instance template for the IP.
<*username*>.v	This file provides a wrapper for the Convolutional Encoder core for simulation.
<*username*>_beh.v	This file provides a behavioral simulation model for the Convolutional Encoder core.
<*username*>_bb.v	This file provides the synthesis black box for the user's synthesis.
<*username*>.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*>.ipx	The IPX file holds references to all of the elements of an IP or Module after it is generated from the IPexpress tool (Diamond version only). The file is used to bring in the appropriate files during the design implementation and analysis. It is also used to re-load parameter settings into the IP/Module generation GUI when an IP/Module is being re-generated.
<*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 Convolutional Encoder 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_ce_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 Convolutional Encoder 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 Convolutional Encoder 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_ce_eval\<*username*>\sim\modelsim\scripts. The simulation script supporting Aldec evaluation simulation is provided in \<*project_dir*>\blk_ce_eval\<*username*>\sim\aldec\scripts. Both ModelSim and Aldec simulation is supported via test bench files provided in \<*project_dir*>\blk_ce_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_ce_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_ce_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 Convolutional Encoder IP core 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_ce_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_ce_eval\<*username*>\impl\(synplify or precision).

To use this project file in Diamond:

1. Choose **File > Open > Project**.

2. Browse to \<*project_dir*>\blk_ce_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_ce_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 Convolutional Encoder IP core 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.

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

- IEEE Std 802.16-2004 IEEE Standard for Local and metropolitan area networks Part 16: Air Interface for Fixed Broadband Wireless Access Systems
- 3GPP TS 25.212 V4.2.0 (2001-09)
- 3GPP2 C.S0002-A Version 5.0 Date: July 13, 2001
- IEEE Standard for Information Technology Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications

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 Version	Change Summary
—	—	2.0	Previous Lattice releases.
June 2006	03.1	3.0	Added support for LatticeECP2, LatticeSC and LatticeXP FPGA families.
December 2006	03.2	3.1	Updated appendices. Added support for LatticeECP2M FPGA family.
June 2007	03.3	3.2	Updated appendices. Added support for LatticeXP2 FPGA family.
May 2009	03.4	3.3	Added LatticeECP2S, LatticeECP2MS and LatticeECP3 FPGA support.
			Added VHDL flow support.
			Added Precision RTL Synthesis and Aldec Active-HDL tools support
			Added Linux/Solaris Platform support.
			Updated GSR usage.
June 2010	03.5	3.4	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 3, “Parameter Settings.”
			Added new content in Chapter 4, “IP Core Generation.”

Appendix A:

Resource Utilization

This appendix gives resource utilization information for Lattice FPGAs using the Block Convolutional Encoder 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 or ispLEVER help system. 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 shows the resource utilization for the Block Convolutional Encoder IP core implemented in a LatticeEC/P FPGA. Table A-2 lists the parameter settings for the IP core configurations shown in Table A-1.

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

IPexpress User-Configurable Mode	Slices	LUTs	Registers	sysMEM™ EBRs	I/Os	f_{MAX} (MHz)
Config1	44	42	48	0	13	404
Config2	24	25	34	0	12	372
Config3	9	6	16	0	7	563
Config4	119	143	131	0	30	278
Config5	43	46	53	0	8	397

1. Performance and utilization data are generated targeting an LFEC/P20E-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 Convolutional Encoder core targeting LatticeECP/EC devices is CONV-BLK-E2-U3.

Table A-2. Parameter Settings of the Evaluation

Configuration	Config1 (default)	Config2	Config3	Config4	Config5
OPN	CONV-ENCO-E2-N1	CONV-ENCO-E2-N1	CONV-ENCO-E2-N1	CONV-ENCO-E2-N1	N/A
Description	K=3, Rate 2/3, Punc, (802.16-2004 SC PHY)	K=9, SDD, Rate 1/2, Non-punc, (3GPP/CDMA2000)	K=7, Rate 1/2, Non-punc, (802.11a, also DVB-S) Xilinx data sheet I	K=7 Dynamic Puncture (802.16-2004 OFDM PHY)	K=7, Rate 3/4, Punc, (802.11a, also DVB-S) Xilinx data sheet II
Input Rate	2	1	1	—	3
Output Rate	3	2	2	—	4
Max Input Rate	—	—	—	5	—
Max Output Rate	—	—	—	6	—
Operation Mode	Block	Block	Continuous	Block	Continuous
Termination Mode	Tail Biting	Zero Flushing	—	Zero Flushing	—
Zero Padding mode	-	Inside	—	Outside	—
Tail Adding mode	Outside	—	—	—	—
Block Length Options	—	—	—	—	—
Punctured Encoder	Yes	No	No	Yes	Yes

Table A-2. Parameter Settings of the Evaluation

Configuration	Config1 (default)	Config2	Config3	Config4	Config5
Dynamic Rate/Pattern	No	—	—	Yes	No
Puncture Pattern	PP0=11 PP1=10	—	—	—	PP0=101 PP1=110
Constraint Length	3	9	7	7	7
Generator Polynomials	GP0=7_8 GP1=5_8	GP0=561_8 GP1=753_8	GP0=171_8 GP1=133_8	GP0=171_8 GP1=133_8	GP0=171_8 GP1=133_8

LatticeECP2 and LatticeECP2S FPGAs

Table A-3 shows the resource utilization for the Block Convolutional Encoder IP core implemented in a LatticeECP2/S FPGA. Table A-2 lists the parameter settings for the IP core configurations shown in Table A-3.

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

IPexpress User-Configurable Mode	Slices	LUTs	Registers	sysMEM EBRs	I/Os	f_{MAX} (MHz)
Config1	44	42	48	0	13	404
Config2	24	25	34	0	12	372
Config3	9	6	16	0	7	563
Config4	119	143	131	0	30	278
Config5	43	46	53	0	8	397

1. Performance and utilization data are generated targeting an LFE2-50E-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 LatticeECP2/ECP2S family.

Ordering Part Number

The Ordering Part Number (OPN) for the Convolutional Encoder core targeting LatticeECP2/S devices is CONV-BLK-P2-U3.

LatticeECP2M FPGAs

Table A-4 shows the resource utilization for the Block Convolutional Encoder IP core implemented in a LatticeEC2M FPGA. Table A-2 lists the parameter settings for the IP core configurations shown in Table A-4.

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

IPexpress User-Configurable Mode	Slices	LUTs	Registers	sysMEM EBRs	I/Os	f_{MAX} (MHz)
Config1	45	44	48	0	13	466
Config2	25	25	34	0	12	509
Config3	9	6	16	0	7	883
Config4	116	141	131	0	30	352
Config5	43	47	53	0	8	504

1. Performance and utilization data are generated targeting an LFE2M/S35E-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 LatticeECP2M family.

Ordering Part Number

The Ordering Part Number (OPN) for the Convolutional Encoder core targeting LatticeECP2M/S devices is CONV-BLK-PM-U3.

LatticeECP3 FPGAs

Table A-5 shows the resource utilization for the Block Convolutional Encoder IP core implemented in a LatticeECP3 FPGA. Table A-2 lists the parameter settings for the IP core configurations shown in Table A-5.

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

IPexpress User-Configurable Mode	Slices	LUTs	Registers	sysMEM EBRs	I/Os	f_{MAX} (MHz)
Config1	41	44	48	0	13	482
Config2	24	25	34	0	12	500
Config3	9	6	16	0	7	500
Config4	108	136	131	0	30	346
Config5	40	45	53	0	8	435

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 (OPN) for the Convolutional Encoder core targeting LatticeECP3 devices is CONV-BLK-E3-U3.

LatticeXP FPGAs

Table A-6 shows the resource utilization for the Block Convolutional Encoder IP core implemented in a LatticeXP FPGA. Table A-2 lists the parameter settings for the IP core configurations shown in Table A-6.

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

IPexpress User-Configurable Mode	Slices	LUTs	Registers	sysMEM EBRs	I/Os	f_{MAX} (MHz)
Config1	44	42	48	0	13	327
Config2	24	25	34	0	12	361
Config3	9	6	16	0	7	589
Config4	115	127	131	0	30	250
Config5	40	46	53	0	8	363

1. Performance and utilization data are generated targeting an LFXP20E-5F484C 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 (OPN) for the Convolutional Encoder core targeting LatticeXP devices is CONV-BLK-XM-U3.

LatticeXP2 FPGAs

Table A-7 shows the resource utilization for the Block Convolutional Encoder IP core implemented in a LatticeXP2 FPGA. Table A-2 lists the parameter settings for the IP core configurations shown in Table A-7.

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

IPexpress User-Configurable Mode	Slices	LUTs	Registers	sysMEM EBRs	I/Os	f_{MAX} (MHz)
Config1	45	44	48	0	13	481
Config2	25	25	34	0	12	491
Config3	9	6	16	0	7	647
Config4	116	141	131	0	30	298
Config5	43	47	53	0	8	495

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 (OPN) for the Convolutional Encoder core targeting LatticeXP2 devices is CONV-BLK-X2-U3.

LatticeSC and LatticeSCM FPGAs

Table A-8 shows the resource utilization for the Block Convolutional Encoder IP core implemented in a LatticeSC/M FPGA. Table A-2 lists the parameter settings for the IP core configurations shown in Table A-8.

Table A-8. Performance and Resource Utilization[1]

IPexpress User-Configurable Mode	Slices	LUTs	Registers	sysMEM EBRs	I/Os	f_{MAX} (MHz)
Config1	41	42	48	0	13	400
Config2	24	24	34	0	12	400
Config3	9	6	16	0	7	400
Config4	115	146	131	0	30	392
Config5	40	44	53	0	8	400

1. Performance and utilization data are generated targeting an LFSC/M3GA25E-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 (OPN) for the Convolutional Encoder core targeting LatticeSC/M devices is CONV-BLK-SC-U3.