Convolutional Encoder

User's Guide

October 2005
ipug03_03.0a October 10, 2005 9:48 a.m.

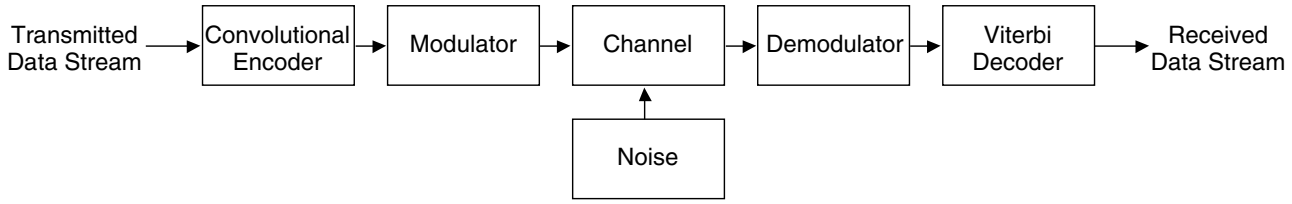
Introduction

Lattice's Convolutional Encoder core is a parameterizable core for convolutional encoding of a continuous input data stream. The core allows variable code rates, constraint lengths and generator polynomials. The core also supports puncturing. Puncturing enables a large range of transmission rates and reduces the bandwidth requirement on the channel. The architectural details of the core are given in the Convolutional Encoder Core Description section.

Convolutional Encoder Basics

Figure 1 shows a digital transmit-receive 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 1 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.

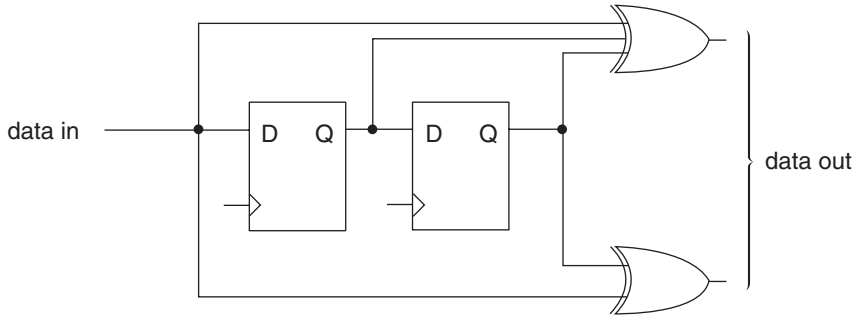
Figure 1. Digital Transmit-Receive System



Convolutional Coding

Convolutional encoding is a process of adding redundancy to a signal stream. Figure 2 shows an example of 1/2 rate convolutional encoding.

Figure 2. Convolutional Encoding



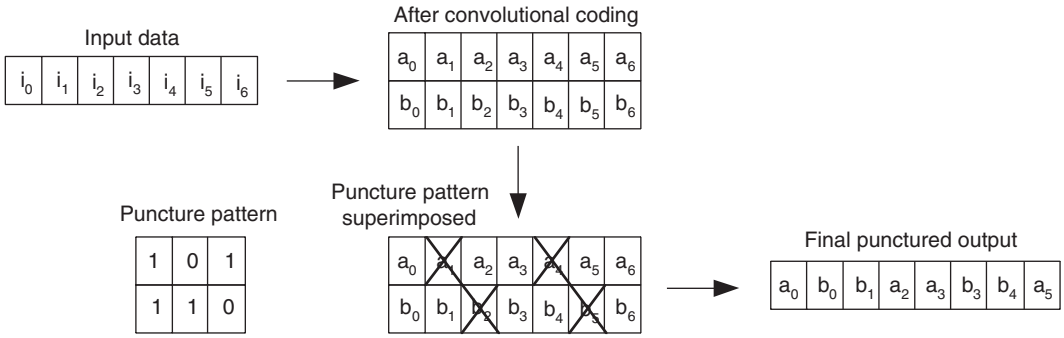
1/2 convolutional coding with constraint length = 3
and generator polynomials 111 and 101

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 three values of the input signal, one present and two 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 three. 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 111 and 101. The MSB of the GP corresponds to the input; the LSBs of the generator polynomial correspond to the state as shown in Figure 2. A bit value of '1' in the generator polynomial represents a used XOR bit and a value of '0' signifies an unused bit.

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 3 shows an example of puncturing.

Figure 3. Puncturing Process



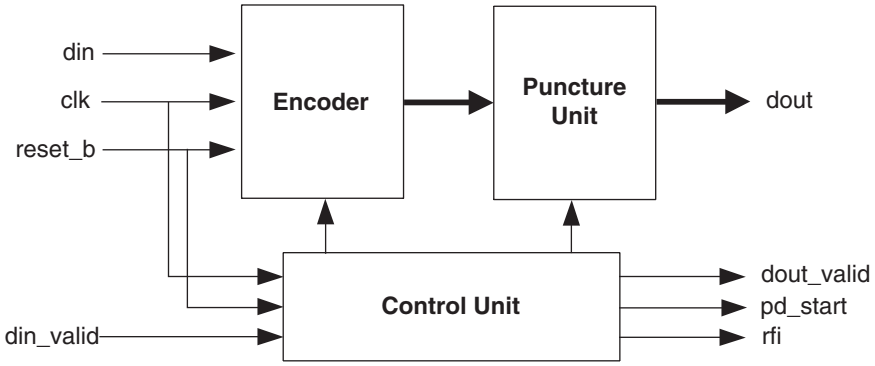
If puncturing is employed in the encoder, the decoder will have to "depuncture" the data before decoding. Depuncturing is done by inserting NULL symbols for the punctured symbols. NULL symbols are equidistant from either '0' or '1'.

Convolutional Encoder Core Description

Internal Architecture

Figure 4 shows the modules of the Convolutional Encoder and their interconnectivity. A brief description of the modules follows.

Figure 4. Convolutional Encoder Internal Architecture



Encoder

This 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 will use the 1/2 rate code to generate the appropriate user-programmed rate.

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 block size and any rate.

Control Unit

The control unit generates the handshake signals `dout_valid, rfi` and `pd_start` using `din_valid` and the status of the decoder. It also generates various control signals required by the encoder and puncture unit.

Signal Descriptions

The top-level representation of the Convolutional Encoder is shown in Figure 5. Table 1 contains the signal descriptions. Timing diagrams for the signals are shown in the Timing Diagrams section.

Figure 5. Convolutional Encoder Top Level Block Diagram

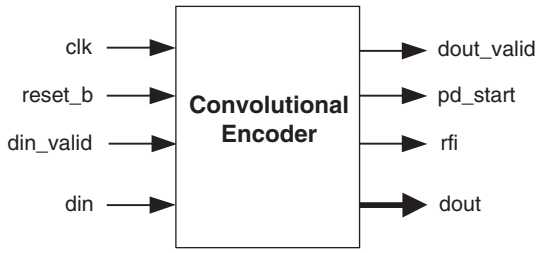


Table 1. Core Signals

Port	Bits	I/O	Description
clk	1	Input	System clock. The clock speed is equal to the input symbol rate.
reset_b	1	Input	System-wide asynchronous reset signal, active low.
din_valid	1	Input	Denotes valid data is being presented at the encoder input. Must be asserted only if the output rfi is high.
din	1	Input	Input data to the encoder. Must be presented only if the encoder output rfi is high.
dout_valid	1	Output	Denotes valid data is present at the encoder output.
pd_start	1	Output	Signifies the start of a punctured block.
rfi	1	Output	Indicates the encoder is ready for input.
dout	2 to 8 (non-punctured)	1 (punctured)	Output data of the encoder. The data is valid only if the signal dout_valid is high.

Interfacing the Convolutional Encoder Core

The puncturing-enabled convolutional encoder is a multi-rate system, with the output rate greater than the input rate. The actual output data rate is dependent on the puncturing rate. The data rate mismatch between input and output is addressed by 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 din_valid must be asserted high. Even if the rfi output is high, the driving system can black-out the input by pulling din_valid low. The core will optimize throughput by utilizing any user-asserted black-out cycles as wait cycles used for data-rate matching.

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

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

Convolutional Encoder Configuration Options

Configurable Parameters

The following core parameters give the user the capability to tailor the core to realize different Convolutional Encoder configurations. These parameters can be configured through the GUI dialog box in IPexpress™.

Constraint Length: This defines the constraint register length. The value can be any integer from 3 to 12.

Input Rate: This defines the input symbol rate for the encoder. The input rate for non-punctured codes is always 1. For punctured codes, the input rate can be any value from 2 to 12.

Output Rate: This defines the output symbol rate for the encoder. The output rate for non-punctured codes can be any value between from 2 to 8. For punctured codes, the output rate can be any value from 3 to 23 (k +1 to 2k -1, where k is the input rate).

Generator Polynomials: GP0, GP1, GP2, GP3, GP4, GP5, GP6 and GP7 are generator polynomials. For non-punctured encoders, the number of generator polynomials is always equal to the output rate. For punctured encoders, the number of generator polynomials is 2.

Punctured Data Support: The encoder supports punctured or non-punctured data. For punctured data, the block size (punctured block size) is equal to the input rate. The two puncture patterns PP0 and PP1 can be defined by the user. The total number of 1's in both puncture patterns must equal the output rate.

Generic Core Configurations
Table 2 shows the description of core configurations available in the standard evaluation package.

Table 2. Core Configurations

	Configuration #1
Constraint Length	7
Input Rate	1
Output Rate	2
Generator Polynomials	
GP0 (Octal)	171
GP1 (Octal)	133
GP2 (Octal)	—
GP3 (Octal)	—
GP4 (Octal)	—
GP5 (Octal)	—
GP6 (Octal)	—
GP7 (Octal)	—
Punctured Data Support	
Punctured Decoder	No
Punctured Block Size	—
Puncture Pattern - PP0	—
Puncture Pattern - PP1	—

Timing Diagrams

The top-level timing diagrams for various configurations are shown in Figure 6 and Figure 7.

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

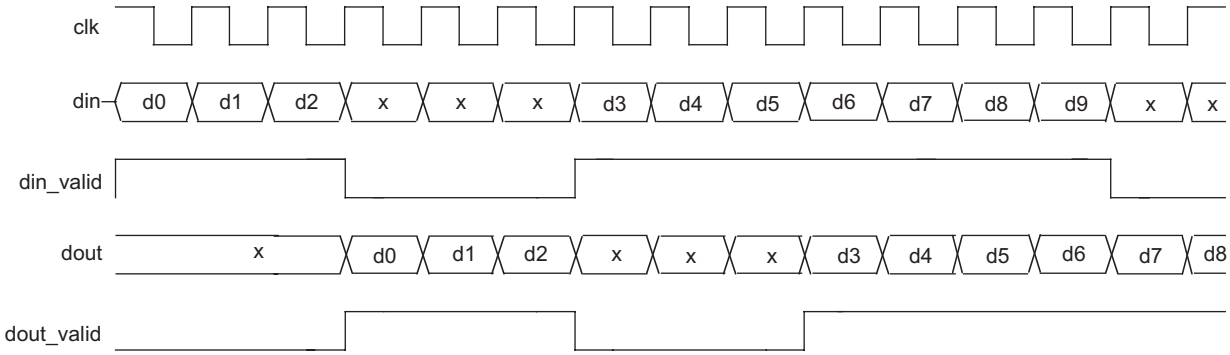
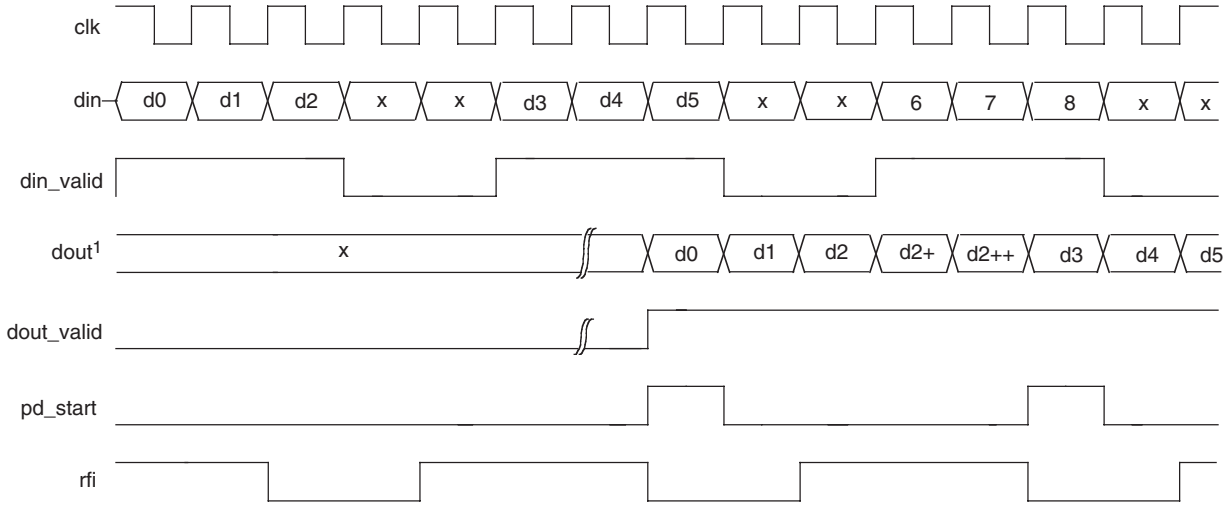


Figure 7. 3/5 Rate Punctured Encoder Timing Diagram

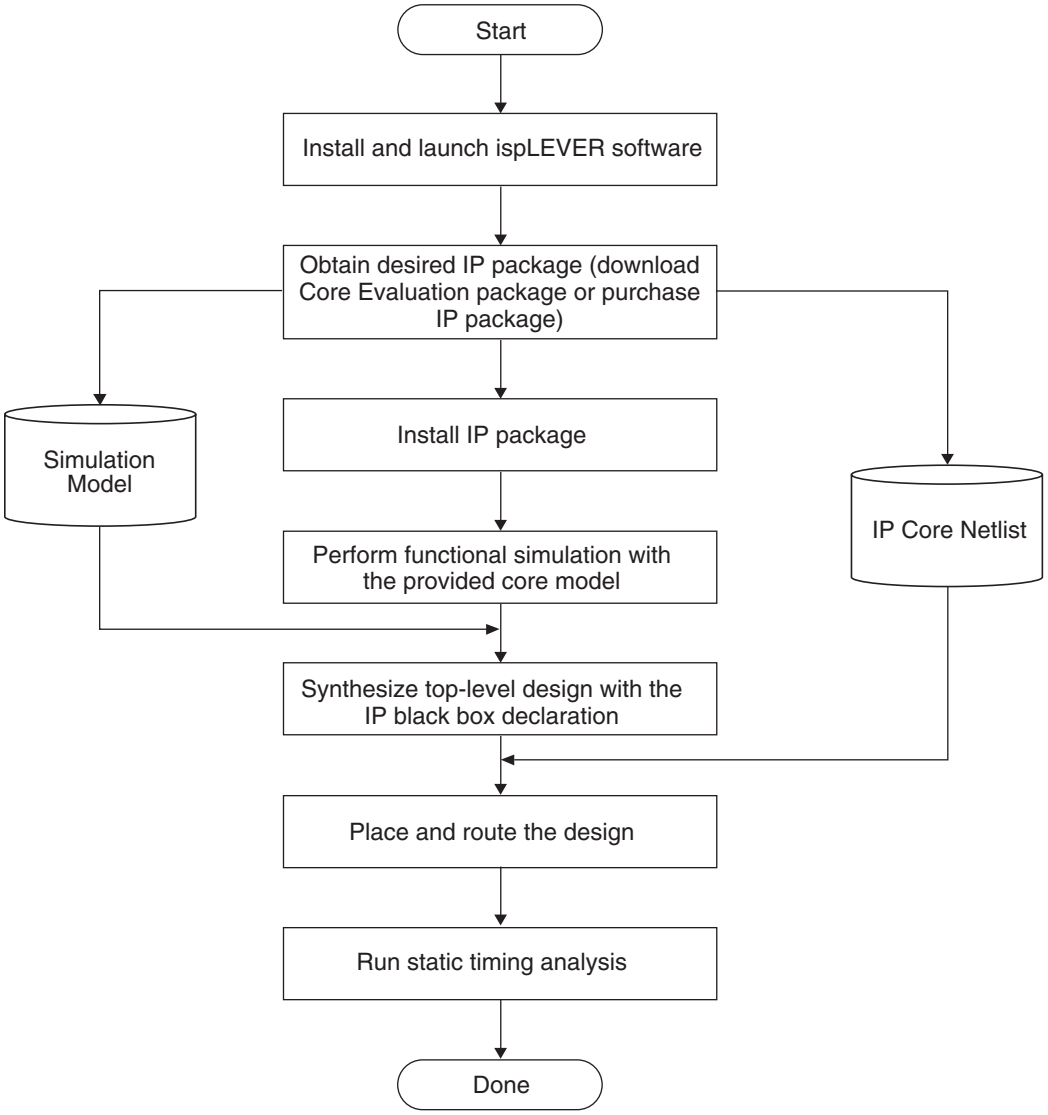


1. For information regarding `din` to `dout` latency, see Appendix A.

Convolutional Encoder Core Design Flow

Figure 8 illustrates the software flow model when designing with a Convolutional Encoder core.

Figure 8. Lattice IP Evaluation Flow



Convolutional Encoder File Hierarchy

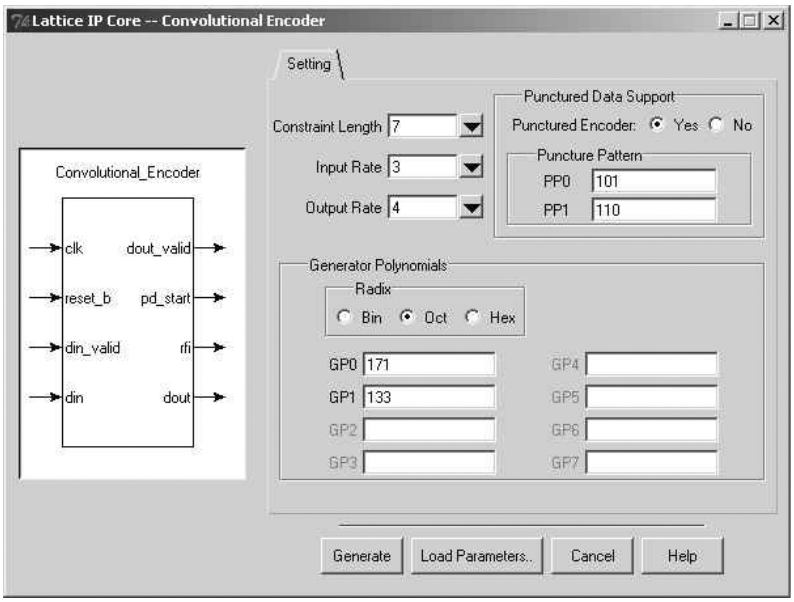
Table 3. File Hierarchy

File Name	Description
Parameter Files	
conv_enco_o4_1_00x_params.v	All configurable parameters
Core Files	
conv_enco_o4_1_00x.ngo	Core database file for ORCA® Series 4 devices.
conv_enco_o4_1_00x.prf	Core preference file for ORCA Series 4 devices.
conv_enco_xp_1_00x.ld2	Core database file for ispXPGA™ devices
Verilog Instantiation Templates	
ce_wrap.v	Verilog user design template for synthesis
conv_enco_o4_1_00x.v	Verilog synthesis model for the core
Testbench Files	
tb_ce_wrap_fsim.v	Testbench for RTL simulation

IPexpress

The Lattice IP configuration tool, IPexpress, is incorporated in the ispLEVER® software. IPexpress includes a GUI for entering the required parameters to configure the core. For more information on using IPexpress and the ispLEVER design software, refer to the software help and tutorials included with ispLEVER. For more information on ispLEVER, see the Lattice web site at: www.latticesemi.com/software.

Figure 9. Convolutional Encoder Parameter Configuration Dialog Box



Implementing a Convolutional Encoder Core Design

Black Box Consideration

Since the core is delivered as a gate-level netlist, the synthesis software will not re-synthesize the internal nets of the core. In the synthesis process, the instantiated core must be declared as a black box. The ispLEVER software automatically detects the provided netlist of the instantiated IP core in the design. For more detailed information regarding Synplify's black box declaration, please refer to the Instantiating Black Boxes in Verilog section of the Synplify reference manual.

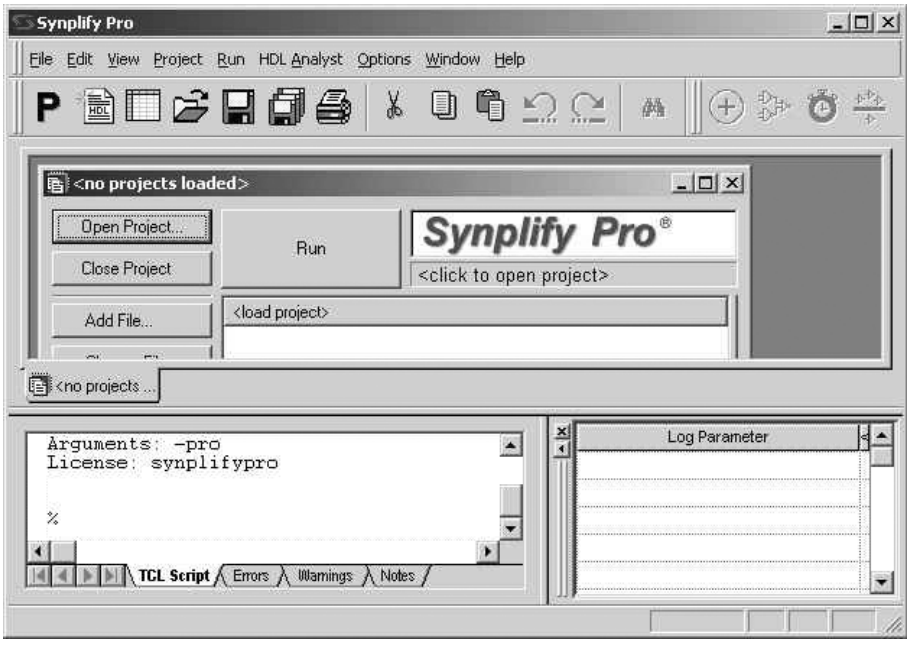
Synthesis

For design synthesis, either the OEM synthesis tools included with the ispLEVER software or a third-party synthesis tool can be used. When using an OEM synthesis tool, a design project must be created using the ispLEVER Project Navigator. For more information on how to create a project using ispLEVER Project Navigator, refer to the ispLEVER online documentation.

Synthesizing Design with Synplify Pro 7.1

1. Launch Synplify Pro 7.1, as shown in Figure 10.

Figure 10. Synplify Pro Window



2. Click **Open Project**. The **Open Project Window** is shown in Figure 11. Click **Project Wizard**. This will launch the **New Project Window**, as shown in Figure 12. Enter a project name and select the project directory. Click **Next**. This will launch the **File Order Window** as shown in Figure 13.

Figure 11. Open Project Window

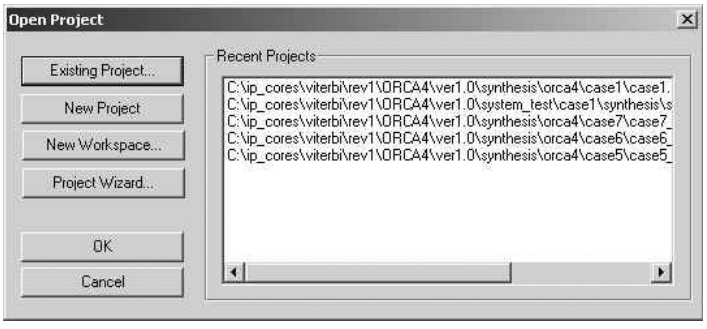


Figure 12. New Project Window

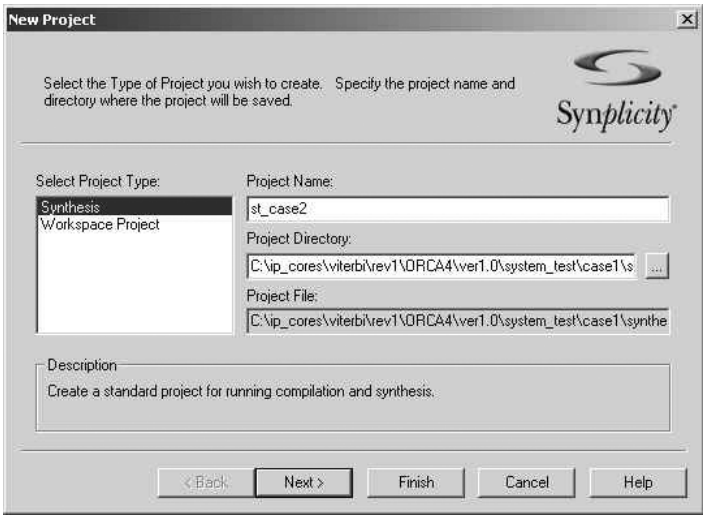
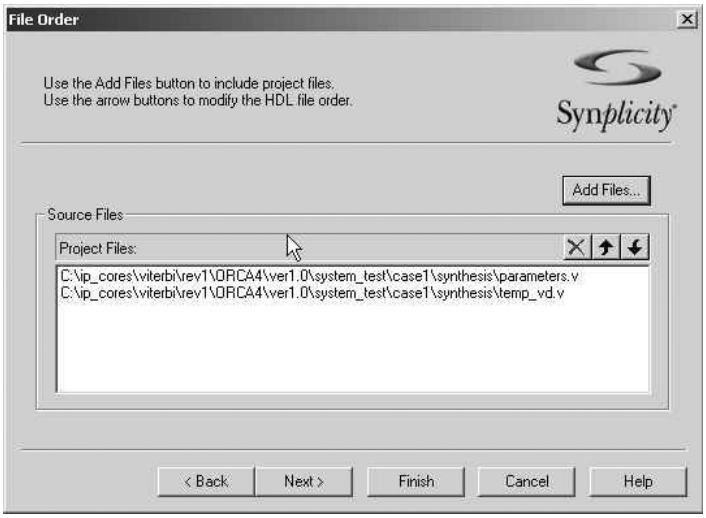
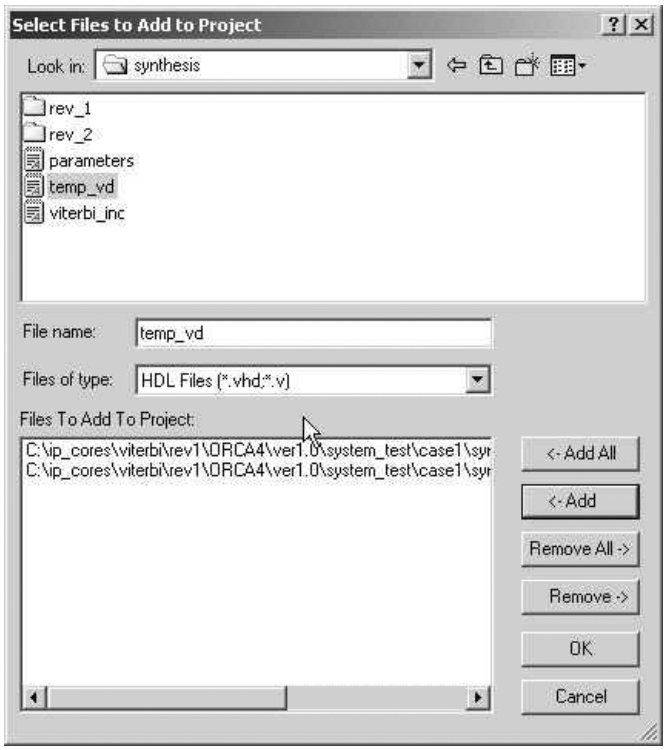


Figure 13. File Order Window



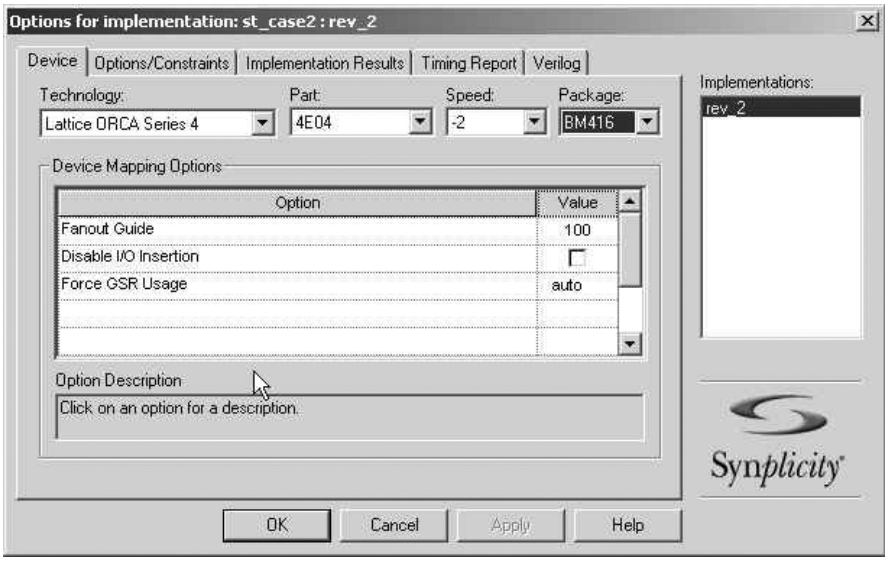
3. In the **File Order Window**, click **Add Files** to open the **Select Files to Add to Project Window** (Figure 14). Add parameters.v, the top-level design file, the verilog synthesis model for the core and other design files. The top-level design file must be the bottom file in the list. Click **OK**. Click **Finish** in the **File Order Window**.

Figure 14. Select Files to Add to Project Window



4. Click **Impl Options** in the Synplify Pro 7.1 window. Options for implementation window are shown in Figure 15. Select the **Technology**, **Part**, **Speed** and **Package**. Click **OK**.

Figure 15. Options for implementation Window



5. In the Synplify Pro 7.1 Window, Click **Run**. This will compile and map the design to the target technology. If there are no errors in the synthesis process, the window will say **Done**.

Functional Simulation with Modelsim

Both a sample script file (do_ce_fsim.do) and testbench template file (tb_ce_wrap_fsim.v) are provided with the Convolutional Encoder core release. For additional details refer to the readme.html file in the Convolutional Encoder core release directory.

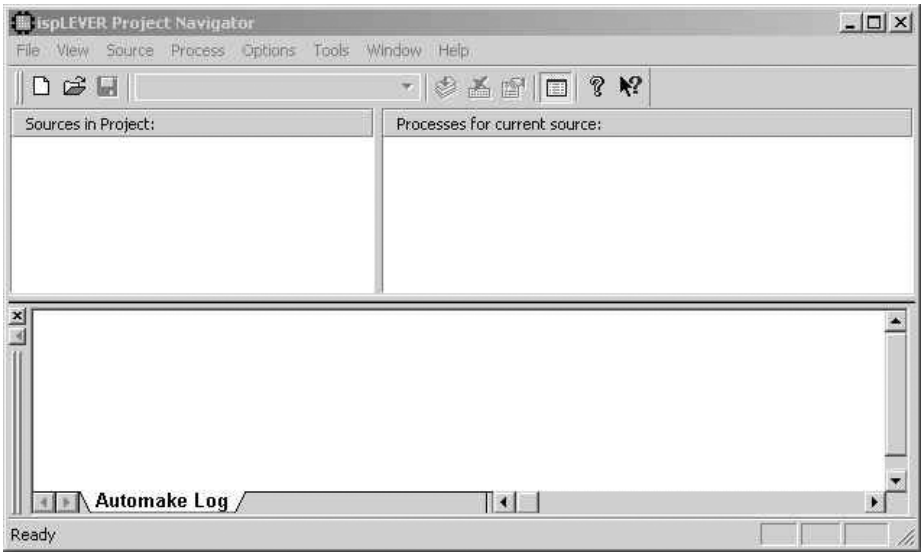
Place and Route

Place and route engines are included in the ispLEVER software. The place and route options are available inside the Constraint Editor. If an OEM synthesis tool is used for design synthesis, the same project can be used to complete the design place and route. If a non-OEM synthesis tool is used for design synthesis, a project must be created using the ispLEVER Project Navigator.

ispLEVER Software Flow for ORCA Devices

1. Create a project directory.
2. Copy the synthesized design EDIF file and the Convolutional Encoder core netlist to the project directory.
3. Launch the ispLEVER software. Figure 16 shows the ispLEVER **Project Navigator Window**.

Figure 16. ispLEVER Project Navigator Window



4. Select **File > New Project** and to create a new project or browse the project directory. All the design files will be generated in this directory. Type project name and select **EDIF** as a project type.
5. Double click the device name in the **Process Window** of the Project Navigator and select the device.
6. Select **Source > Import**. The **Import File Dialog Box** is shown in Figure 17. Enter the EDIF file name and click **Open**. The **Import EDIF Dialog Box** is shown in Figure 18. Select the EDIF vendor type and click **OK**. The EDIF file will be added to the project.

Figure 17. Import File Dialog Box

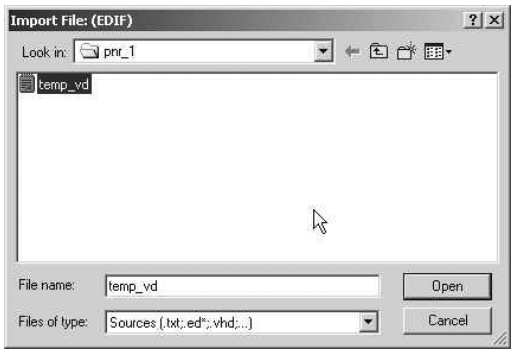
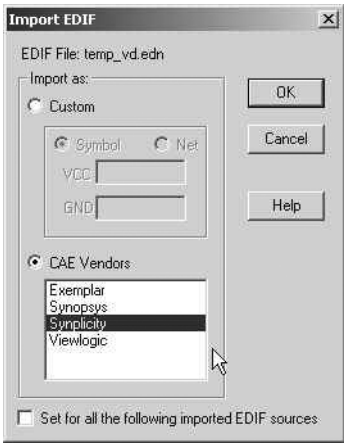
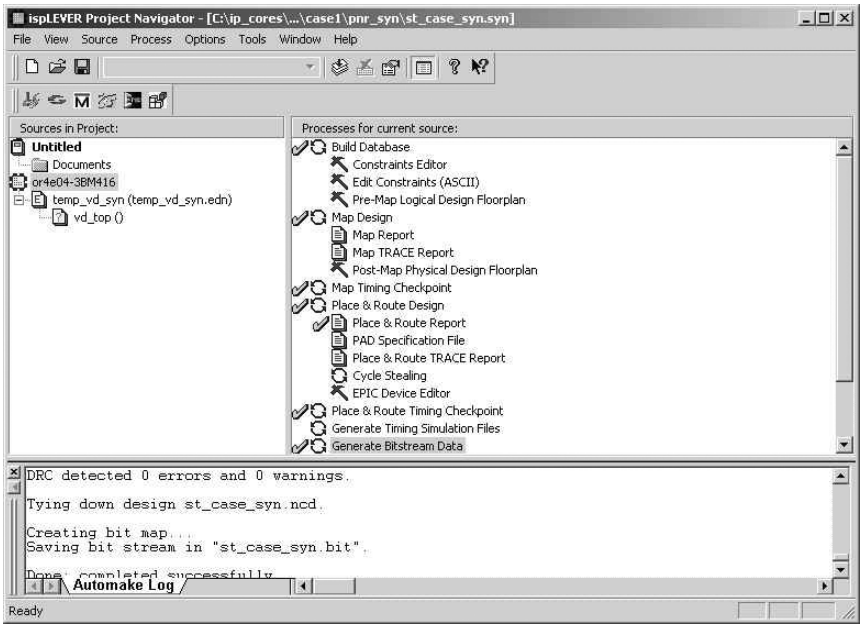


Figure 18. Import EDIF Dialog



7. Double click **Build Database**. This will read the EDIF netlist and generate the design database. If there are no errors in this process, you will see a green mark next to the **Build Database** process (Figure 19).

Figure 19. Project Navigator Window with Successfully Completed Processes for ORCA Design Flow



8. Double click on the **Constraint Editor**. You can enter the required preferences, including pin locking, using the constraint editor. You can also create an ASCII preference file using a text editor. For additional details on the ispLEVER preference language, refer to the ispLEVER online documentation.

9. Double click on **Map Design**. This will map the design into the physical elements of the target device. If there are no errors in this process, you will see a green mark next to the **Map Design** process (Figure 19).

10. Double click on **Place & Route Design**. This will place and route the mapped design. If there are no errors in this process, a green mark appears next to the **Place & Route Design** process (Figure 19).

ispLEVER Software Flow for ispXPGA Devices

1. Create a project directory.

2. Copy the synthesized design EDIF file and the Viterbi Decoder core netlist to the project directory.

3. Launch the ispLEVER software. Figure 16 shows the ispLEVER Project Navigator Window.

4. Select **File > New Project** and create or browse to the project directory. All design files will be generated in this directory. Figure 5 shows the **New Project Creation Dialog Box**. Type project name and select **EDIF** as a project type.

5. Double click on the device name in the **Process Window** of the Project Navigator and select the device. Figure 6 shows the **Device Selection Dialog Box**.

6. Select **Source > Import**. The **Import File Dialog Box** is shown in Figure 24. Enter the EDIF filename and click **Open**. The **Import EDIF Dialog Box** is shown in Figure 18. Select the EDIF vendor type and click **OK**. The EDIF file will be added to the project.

7. Double click on **Build Database**. This will read the EDIF netlist and generates the design database. If there are no errors in the process, you will see a green mark next to the **Build Database** process (Figure 19).

8. Double click on the **Constraint Editor**. You can enter the required preferences including pin locking using the Constraint Editor. You can also create an ASCII constraint file using a text editor. For additional details on the ispXPGA constraints, refer to the ispLEVER online documentation.

9. Double click on the **Pack & Place Design**. This will pack and place the design into the physical elements. If there are no errors in the process, you will see a green mark next to the **Pack & Place Design** process.

10. Double click on the **Route Design**. This will route the pack and placed design. If there are no errors in the process, you will see a green mark next to the **Route Design** process.

11. Double click on the **Timing Analysis**. This will execute static timing analyzer on the routed design. If there are no errors in the process, you will see a green mark next to the **Timing Analysis** process.

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Appendix for ORCA Series 4 FPGAs

Table 4. Core Configurations for ORCA Series 4 FPGAs

	Configuration #1
Constraint length	7
Input rate	1
Output rate	2
Generator Polynomials	
GP0 (octal)	171
GP1 (octal)	133
GP2 (octal)	—
GP3 (octal)	—
GP4 (octal)	—
GP5 (octal)	—
GP6 (octal)	—
GP7 (octal)	—
Punctured Data Support	
Punctured encoder	No
Puncture pattern – PP0	—
Puncture pattern – PP1	—

Supplied Netlist Configurations

The Ordering Part Number (OPN) for all configurations of this core in ORCA Series 4 devices is CONV-ENCO-04-N1. Table 4 lists the netlist configurations that are available in the Evaluation Package for this core, which can be downloaded from the Lattice web site at www.latticesemi.com.

You can use the IPexpress software tool to help generate new configurations of this IP core. IPexpress is the Lattice IP configuration utility, and is included as a standard feature of the ispLEVER design tools. Details regarding the usage of IPexpress can be found in the IPexpress and ispLEVER help system. For more information on the ispLEVER design tools, visit the Lattice web site at: www.latticesemi.com/software.

Table 5. Performance and Resource Utilization[1]

Configuration	ORCA 4 PFUs[2]	LUTs	Registers	External I/Os	sysMEM™ EBRs	f_{MAX} (MHz)	Latency[3]
conv_enco_o4_1_001.lpc	4	6	16	7	N/A	333	3

1. Performance and utilization characteristics using ispLEVER v.3.0 software and targeting the OR4E02, package BA352, speed 2. Synthesized using Synplicity's Synplify Pro v.7.1.1. When using this IP core in a different density, package, speed, or grade within the OR4E family, performance and utilization may vary.
2. Programmable Function Unit (PFU) is a standard logic block of Lattice FPGA devices. For more information, check the data sheet of the device.
3. The latency values are for `din` to `dout` with `din_valid` is high whenever `rfi` is high. The `din` to `dout` latency relationship can be explained as follows. For non-punctured encoders, the latency value is 3 when constraint length is greater than 4, otherwise the value is 2. For punctured encoders, the latency value is (output rate + 6) when constraint length is greater than 4, otherwise the value is (output rate + 4).

Appendix for ispXPGA® FPGAs

Supplied Core Configurations

Table 6 shows the description of core configurations available in the standard evaluation package.

Table 6. Core Configurations for ispXPGA FPGAs

	Configuration #1
Constraint Length	7
Input Rate	1
Output Rate	2
Generator Polynomials	
GP0 (Octal)	171
GP1 (Octal)	133
GP2 (Octal)	—
GP3 (Octal)	—
GP4 (Octal)	—
GP5 (Octal)	—
GP6 (Octal)	—
GP7 (Octal)	—
Punctured Data Support	
Punctured Encoder	No
Puncture Pattern -PP0	—
Puncture Pattern -PP1	—

Supplied Netlist Configurations

The Ordering Part Number (OPN) for all configurations of this core in ispXPGA devices is CONV-ENCO-XP-N1. Table 6 lists the netlist configurations that are available in the Evaluation Package for this core, which can be downloaded from the Lattice web site at www.latticesemi.com.

You can use the IPexpress software tool to help generate new configurations of this IP core. IPexpress is the Lattice IP configuration utility, and is included as a standard feature of the ispLEVER design tools. Details regarding the usage of IPexpress can be found in the IPexpress and ispLEVER help system. For more information on the ispLEVER design tools, visit the Lattice web site at: www.latticesemi.com/software.

Table 7. Performance and Resource Utilization[1]

Configuration	XPGA PFUs[2]	LUT-4s	Registers	External I/Os	sysMEM EBRs	f_{MAX} (MHz)	Latency[3]
conv_enco_xp_1_001.lpc	6	8	22	7	N/A	510	3

1. Performance and utilization characteristics are generated using LFX1200B, package FE680, speed 4 in Lattice ispLEVER v.3.x software. The evaluation version of this IP core only works on this specific device density, package, and speed grade.
2. Programmable Function Unit (PFU) is a standard logic block of Lattice FPGA devices. For more information, check the data sheet of the device.
3. The latency values are for `din` to `dout` with `din_valid` is high whenever `rfi` is high. The `din` to `dout` latency relationship can be explained as follows: For Non-punctured encoders, the latency value is 3 when Constraint Length is greater than 4 or else the value is 2. For punctured encoders, the latency value is (Output Rate + 6) when Constraint Length is greater than 4 or else the value is (Output Rate + 4).