Viterbi Decoder

User's Guide

October 2005
ipug04_02.0

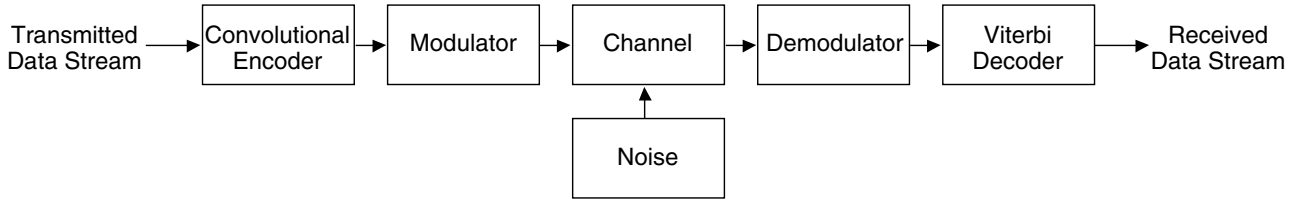
Introduction

Lattice's Viterbi Decoder core is a parameterizable core for decoding different combinations of convolutionally encoded sequences. The decoder core supports various code rates, constraint lengths and generator polynomials. The core also supports soft-decision decoding and is capable of decoding punctured codes. The architectural details of the core are given in the next section.

Viterbi Decoder Basics

Viterbi decoding is an efficient algorithm for decoding convolutionally encoded sequences. In the decoder, the convolutional coded sequences, corrupted by channel noise, are decoded back to the original sequence. A digital transmit-receive system is shown in Figure 1, which uses a Viterbi decoder for decoding the convolutionally coded data. The digital data stream (e.g., voice, image or any packetized data) is first convolutionally encoded, modulated and transmitted through a wired or wireless channel. The channel noise is symbolically denoted by a "noise" block added to the channel. 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.

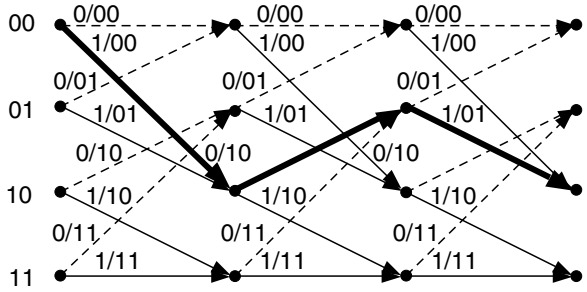
Figure 1. Digital Transmit-Receive System



Convolutional Encoding

Convolutional encoding can be considered as a series of state transitions for every input symbol. The input and the resulting state transition can be shown in a special state transition diagram called a "trellis tree" (Figure 2).

Figure 2. Trellis Tree



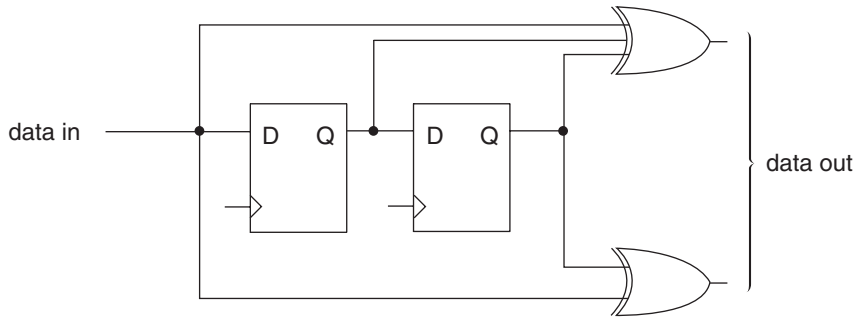
Trellis for 3 stages and constraint length = 3
Branches corresponding to input seq. 101 is highlighted

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, the trellis is formed in memory, where the metrics corresponding to every path are recorded. After constructing the trellis for a sufficient length (called the traceback length), the traceback is continued from the node having the minimum path metric. This will lead to a stating node on the trellis. From this point one can trace forward and decode the original sequence.

Figure 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 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 3. 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 GP correspond to the state as shown in Figure 3. A bit value of '1' in the generator polynomial represents a used XOR bit and a value of '0' signifies an unused bit.

Figure 3. Convolutional Encoding

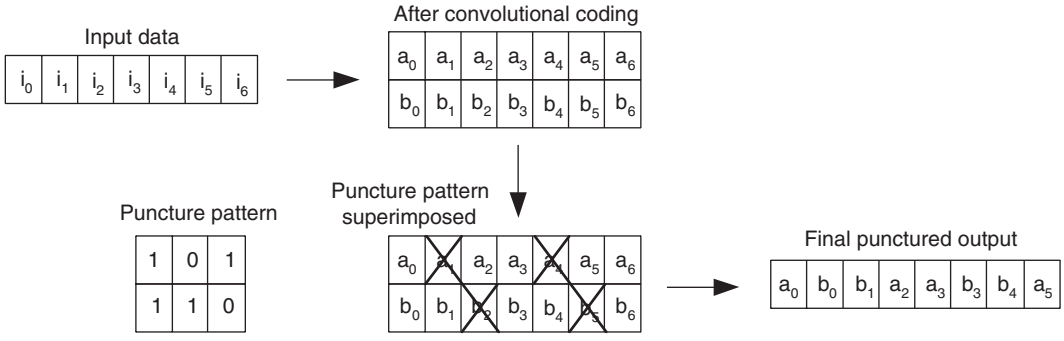


1/2 convolutional coding with constraint length = 3
and generator polynomials 111 and 101

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 4 shows an example of puncturing.

Figure 4. Puncturing Process



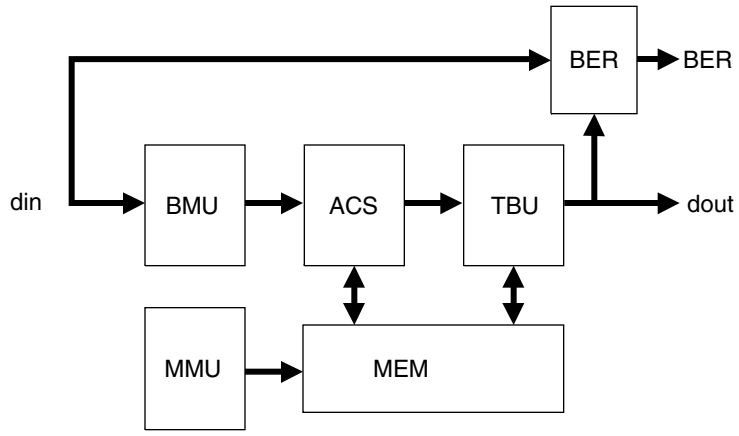
If puncturing is employed in the encoder, the decoder will have to "depuncture" the data before decoding. Depuncturing is done by inserting NULL symbols for the punctured symbols. NULL symbols are chosen to be equidistant from either '0' or '1'.

Viterbi Decoder Core Description

Internal Architecture

Figure 5 shows the modules of the Viterbi decoder and their interconnectivity. Below is a brief description of the modules.

Figure 5. Internal Architecture of the Viterbi Decoder



Branch Metric Unit (BMU)
The BMU receives 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, or l1 norm (sum of absolute values) for soft-decision encoded data.

Add, Compare and Select (ACS) Unit
The ACS unit adds the current metric to the accumulated metric for each path and determines the least metric for each state of the trellis. The ACS retrieves the accumulated metric from the register files, then adds the current metric. The result is stored back in the register files. The ACS unit also writes the survivor trellis path (the previous state information) in memory.

Trace Back Unit (TBU)
The traceback unit traces back the trellis after a block of data (determined by the trace back length) has been processed by the ACS. First, the TBU establishes an optimal path by starting from the node of minimum metric and traces back the path in the trellis all the way to the beginning of the trellis tree. Then, the original data corresponding to the encoded data is determined.

Memory (MEM)
The memory stores the accumulated metric and the previous state information (trace-back information).

Memory Management Unit (MMU)
The MMU generates addresses and read/write signals for the memory during different phases of operation.

Bit Error Rate (BER) Monitor
This optional module is used to estimate the bit error rate of the channel. The bit error rate is estimated by encoding the decoded output symbols using the same generator polynomials, then comparing them with the delayed input to the Viterbi decoder. Assuming that the error in decoding is zero or negligible, the error determined by BER is equal to the channel error.

Signal Descriptions

The top-level representation of the Viterbi Decoder is shown in Figure 6. Table 1 contains the signal description. Timing diagrams for the signals are shown in the Timing Diagram section.

Figure 6. Top Level Block Diagram

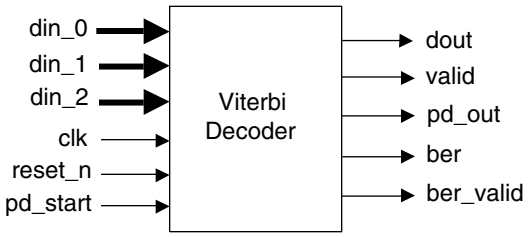


Table 1. Core Signals

Port	Bits	I/O	Description
clk	1	Input	System clock. The clock speed is equal to the input symbol rate.
reset_n	1	Input	System-wide asynchronous active-low reset signal.
pd_start	1	Input	Punctured data block start. This indicates the start of a new block of punctured data. Not available while decoding non-punctured codes
din_0, din_1, din_2	1 or 3 to 8 (each)	Input	Data input buses. • The buses become one-bit inputs for hard decision decoding. • The buses are equal to the soft width for soft decision decoding. • For punctured codes, the number of buses is one. • For non-punctured codes, the number of buses corresponds to the code rate: two for 1/2 rate codes, three for 1/3 rate codes.
dout	1	Output	Decoded data output.
valid	1	Output	This indicates the data currently presented on dout is valid.
pd_out	1	Output	Punctured decoder output. This indicates the data currently presented on dout is valid. Not available while decoding non-punctured codes.
ber	16	Output	Bit-error rate output
ber_valid	1	Output	This indicates valid data is being presented at the ber output port. This signal goes high at the end of every block of BER Period clocks.

Viterbi Decoder Configuration Options

Configurable Parameters

The following core parameters give the user the capability to tailor the core to realize different configurations of the Viterbi decoder.

Constraint Length: The value can be any integer from 3 to 8.

Code Rate: 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 codes, this can be equal to 1/2 or 1/3. For punctured codes, this parameter defines the code rate for the mother code and is fixed to 1/2.

Traceback Length: This is the length of the survivor sequence in the traceback through the Viterbi trellis. It can be any value from 8 to 128.

Generator Polynomials: GP0, GP1 and GP2 are generator polynomials. GP2 is only used in non-punctured decoders with a code rate of 3.

Implementation Methods: In the parallel implementation, the processing of data symbols is completed in one cycle. In the hybrid implementation, it is completed in $2^{\text{hybrid index}}$ cycles.

Decoder Inputs: The decoder supports hard and soft decision decoding. For soft decision decoding, the width (soft width) can be any integer from 3 to 8. The soft decision decoder also supports either signed or unsigned data types. For non-punctured codes, the decoder can be configured either as a hard-decision or soft-decision decoder, whereas for punctured codes, the decoder can only be a soft-decision decoder.

Punctured Data Support: The decoder supports punctured or non-punctured data. For punctured data, the block size (punctured block size) can be any value from 2 to 12. The core allows the puncture patterns PP0 and PP1 to be user defined.

BER Support: The optional BER monitor can be turned on or off. The BER Period parameter determines the period for which the BER is counted and is equal to log2 of the period. After this period, the BER value is reset and the system starts counting again.

Generic Core Configurations
Table 2 shows the description of the available core configurations.

Table 2. Core Configurations

	Configuration #1
Constraint Length	7
Code Rate	2
Traceback Length	42
Generator Polynomials	
GP0 (Octal)	171
GP1 (Octal)	133
GP2 (Octal)	—
Implementation Method	
Parallel/Hybrid	Parallel
Hybrid Index	—
Decoder Inputs	
Hard/Soft Decision	Soft
Soft Width	3
Data Type	Signed
Punctured Data Support	
Punctured Decoder	No
Punctured Block Size	—
Puncture Pattern - PP0	—
Puncture Pattern - PP1	—
BER	
BER Monitor	No
BER Period	—

Timing Diagrams

The top-level timing diagrams for various configurations are given in the following figures.

Figure 7. Timing Diagram for a Parallel, Non-punctured Decoder

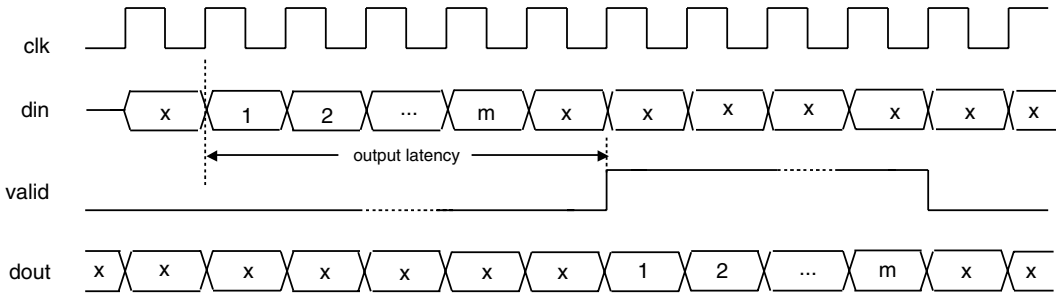


Figure 8. Timing Diagram for a Parallel, Punctured (Rate = 2/3) Decoder

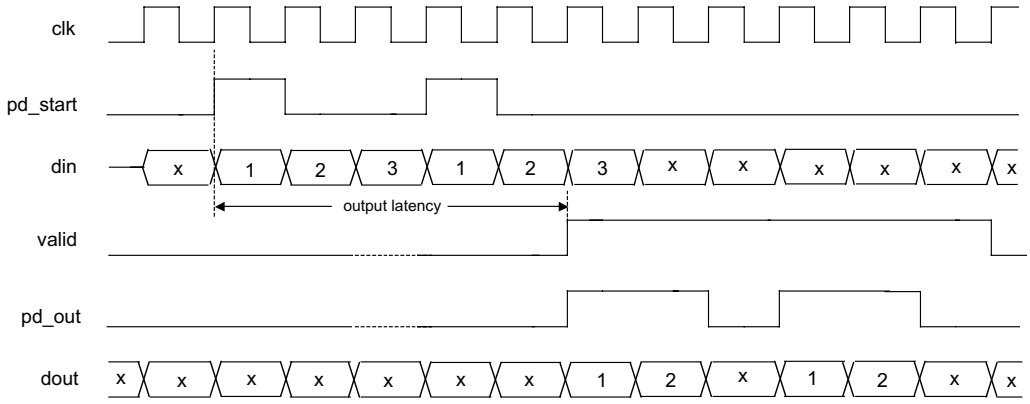


Figure 9. Timing Diagram for a Hybrid (2 Cycles), Non-punctured Decoder

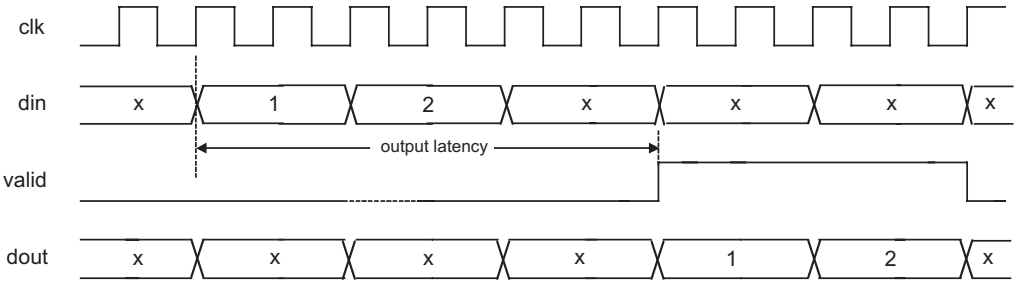
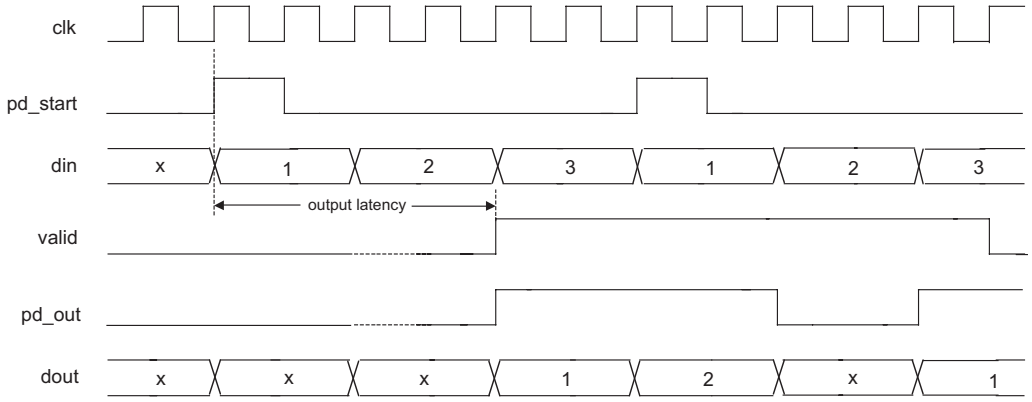


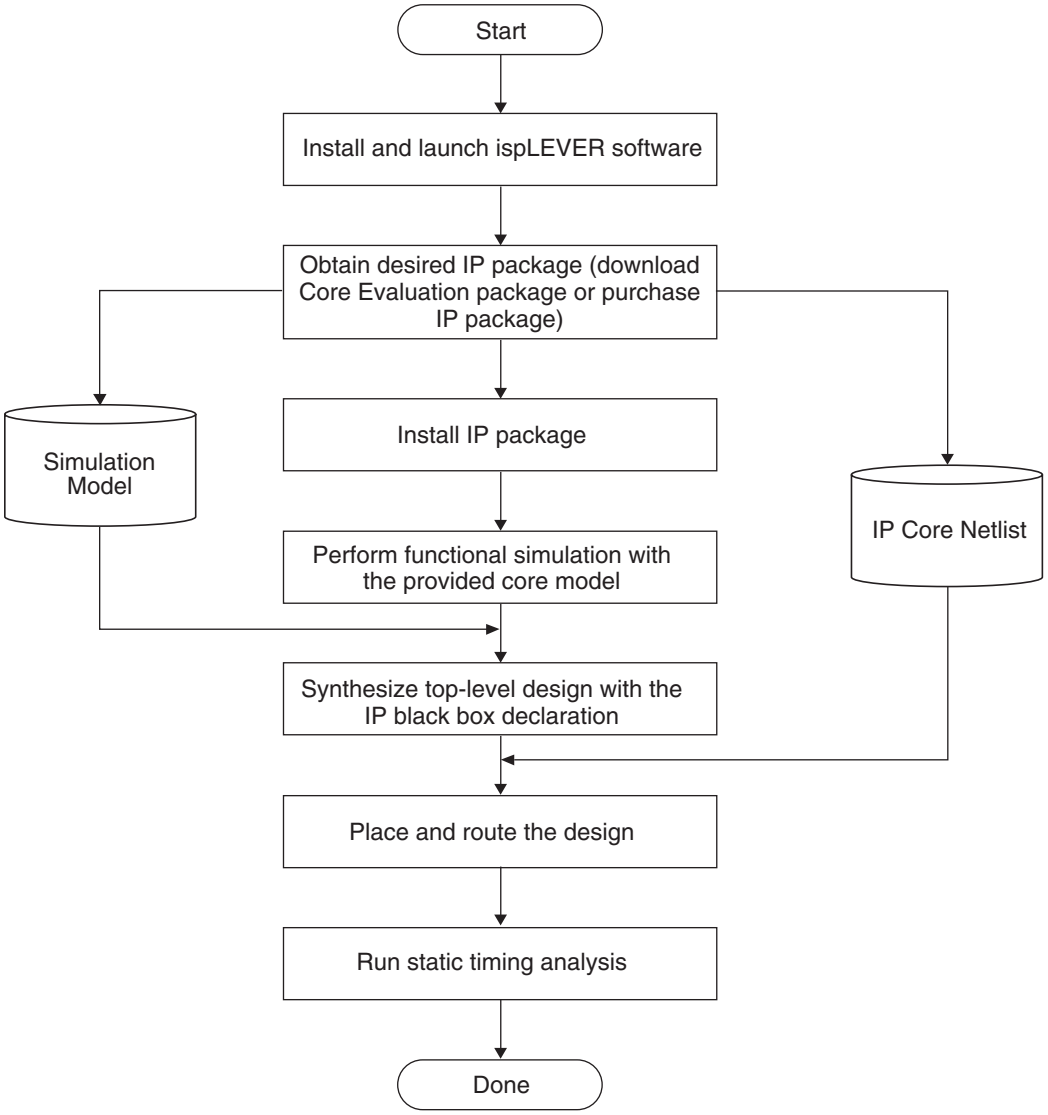
Figure 10. Timing Diagram for a Hybrid (2 Cycles), Punctured (Rate = 2/3) Decoder



Viterbi Decoder Core Design Flow

Figure 11 illustrates the software flow model used when designing with a Viterbi Decoder core.

Figure 11. IP Evaluation Flow



Viterbi Decoder File Hierarchy

Table 3. File Hierarchy

File Name	Description
Parameter Files	
vterb_deco_o4_1_00x_params.v	All configurable parameters.
Core Files	
vterb_deco_o4_1_00x.ngo	Core database file for ORCA® Series 4 devices.
vterb_deco_o4_1_00x.prf	Core preference file for ORCA Series 4 devices.
conv_enco_sp_1_00x.ld2	Core database file for ispXPGA® devices.
conv_enco_sp_1_00x.lct	Core constraint file for ispXPGA devices.
Verilog Instantiation Files	
vd_wrap.v	Verilog user design template for synthesis.
vterb_deco_o4_1_00x.v	Verilog synthesis model for the core.
Testbench Files	
tb_vd_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.

Implementing a Viterbi Decoder Core Design

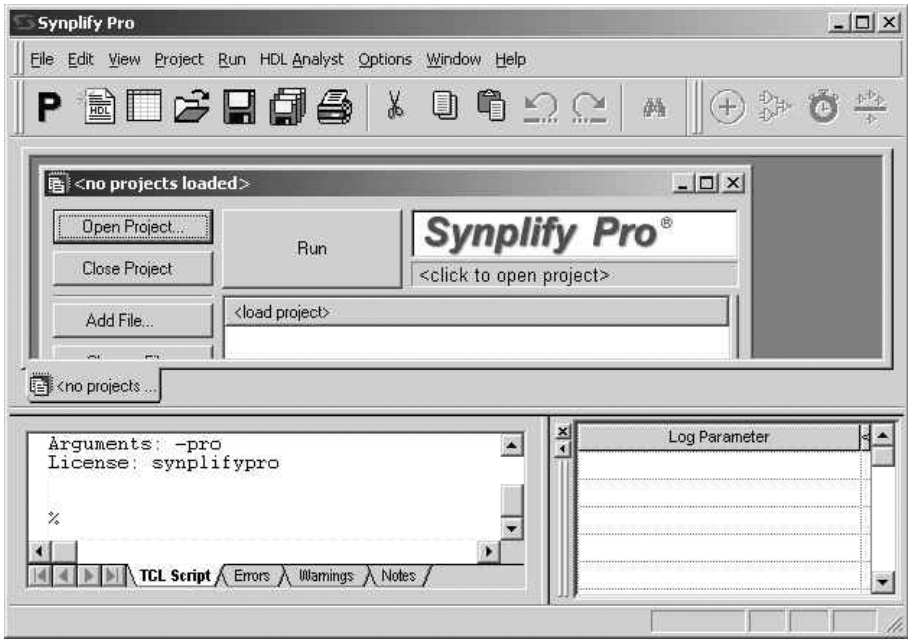
Synthesis

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 a Design with Synplify®

1. Launch the Synplify Pro 7.1 Window (Figure 12).

Figure 12. Synplify Pro Window



2. Click **Open Project**. The Open Project Window is shown in Figure 13. Click **Project Wizard**. The **New Project Window** is shown in Figure 14. Enter a project name and select the project directory. Click **Next**. The **File Order Window** is shown in Figure 15.

Figure 13. Open Project Window

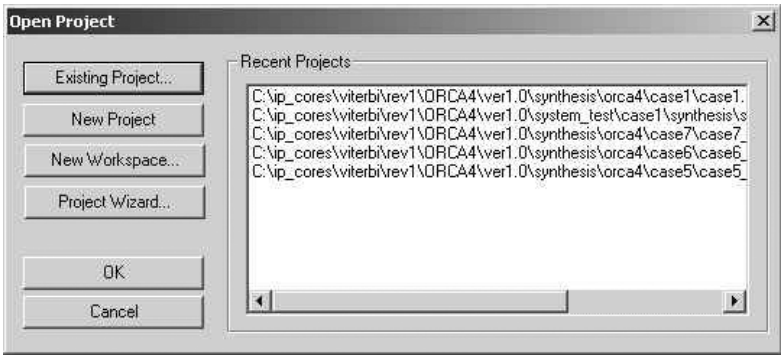


Figure 14. New Project Window

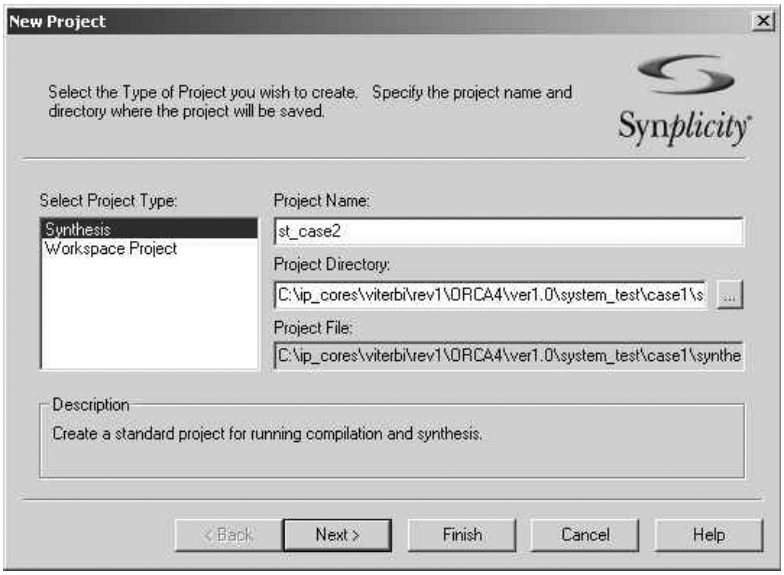
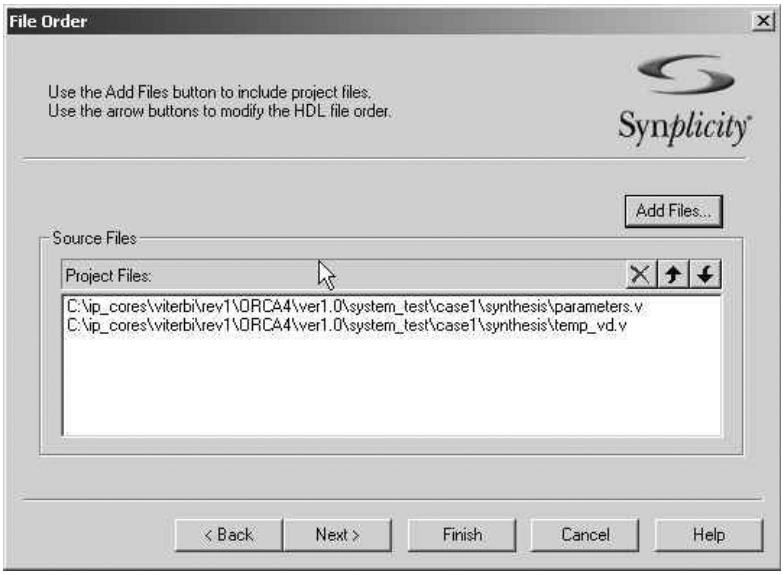


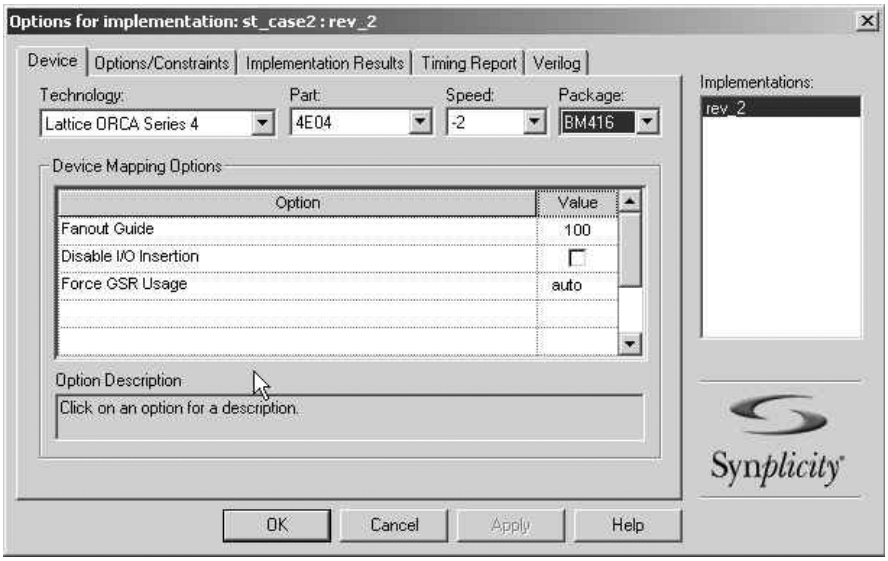
Figure 15. File Order Window



3. In the **File Order Window**, click **Add Files** to open the **Select Files to Add to Project Window** (Figure 19). Add parameters.v, top-level design file, verilog synthesis model for the core, and other design files. The top-level design file must be the last file on the list. Click **OK**. Click **Finish in the File Order Window**.

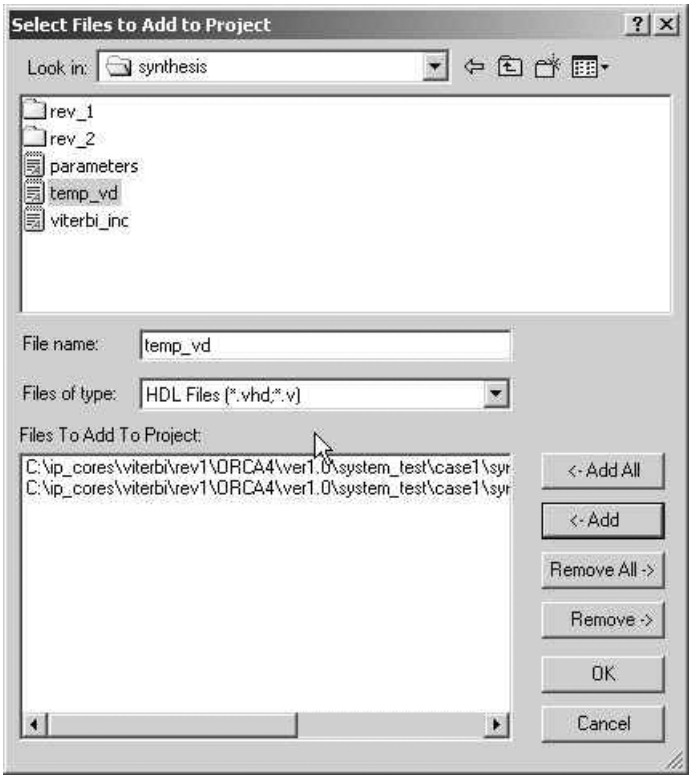
4. Click **Impl Options** in the Synplify Pro 7.1 Window. The **Options for Implementation Window** is shown in Figure 16. Select the **Technology**, **Part**, **Speed** and **Package**. Click **OK**.

Figure 16. Options for Implementation Window



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

Figure 17. Select Files to Add to Project Window



Functional Simulation with Modelsim

A sample script file (do_vd_fsim.do) and testbench template (tb_vd_wrap_fsim.v) are provided with the Viterbi Decoder core. For additional information, refer to the readme.html file in the Viterbi Decoder core directory.

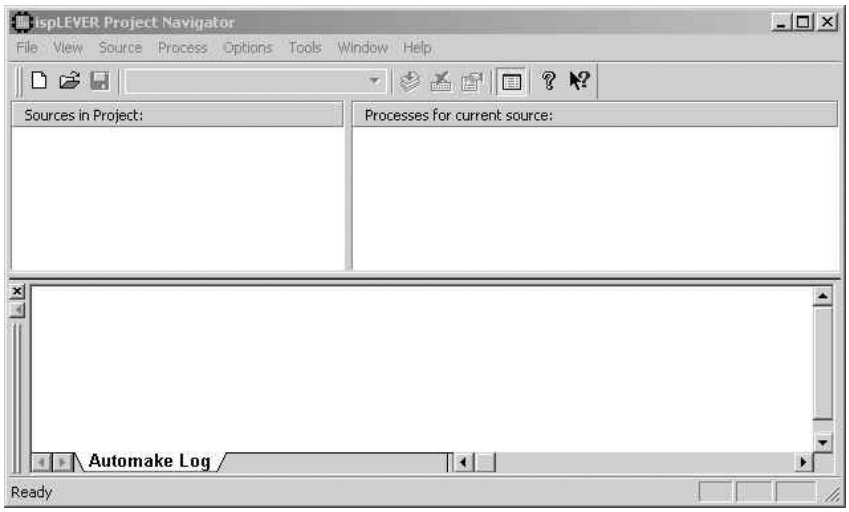
Place and Route

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 Series 4 Devices

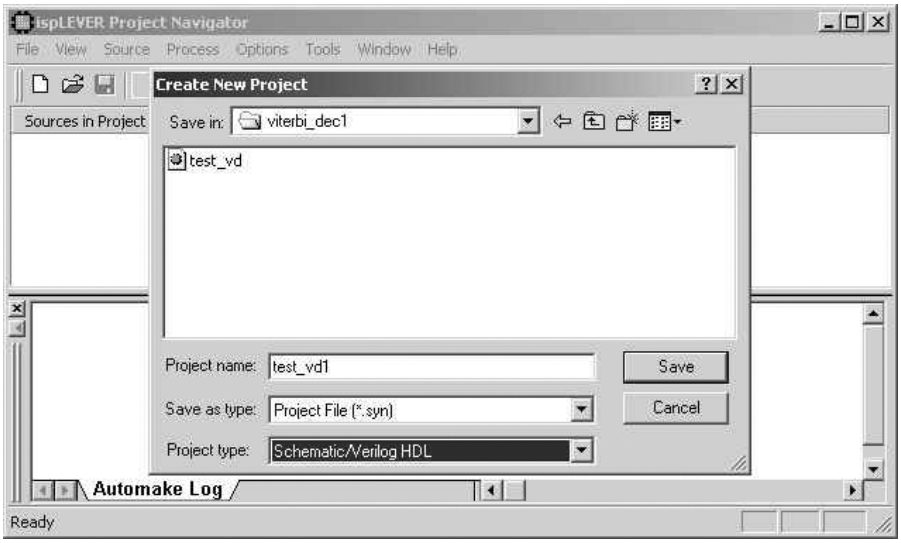
1. Create a project directory.
2. Copy the synthesized design EDIF file and the Convolutional Encoder core netlist to the project directory.
3. Launch the ispLEVER software. Figure 18 shows the ispLEVER **Project Navigator Window**.

Figure 18. ispLEVER Project Navigator Window



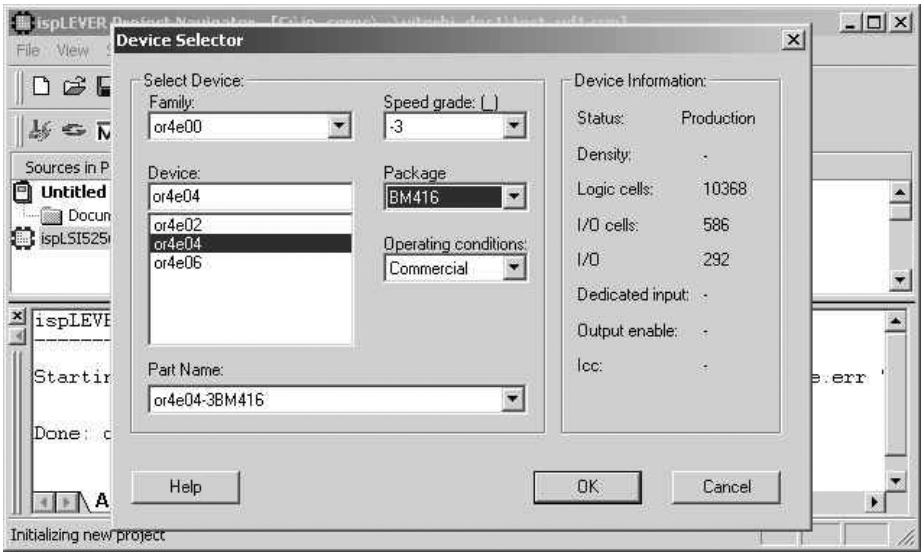
4. Select **File > New Project** and to create a new project or browse the project directory. All the design files will be generated in this directory. Figure 19 shows the **New Project Creation Dialog Box**. Type project name and select **EDIF** as a project type.

Figure 19. New Project Creation Dialog Box



5. Double click the device name in the **Process Window** of the Project Navigator and select the device. Figure 20 shows the **Device Selection Dialog Box**.

Figure 20. Device Selection Dialog Box



6. Select **Source > Import**. The **Import File Dialog Box** is shown in Figure 21. Enter the EDIF file name and click **Open**. The **Import EDIF Dialog Box** is shown in Figure 22. Select the EDIF vendor type and click **OK**. The EDIF file will be added to the project.

Figure 21. Import File Dialog Box

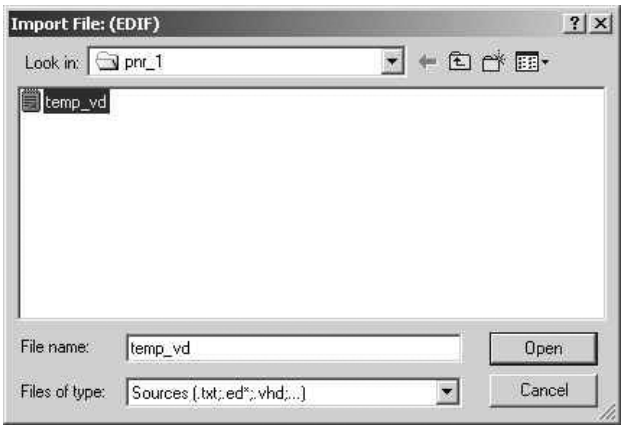
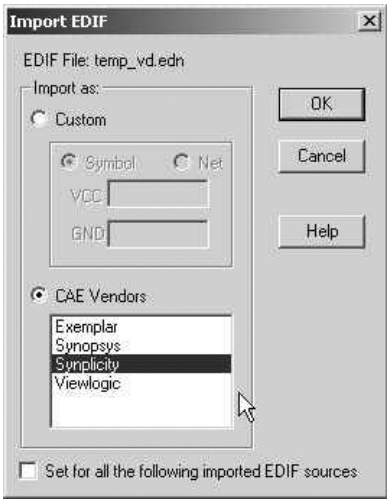
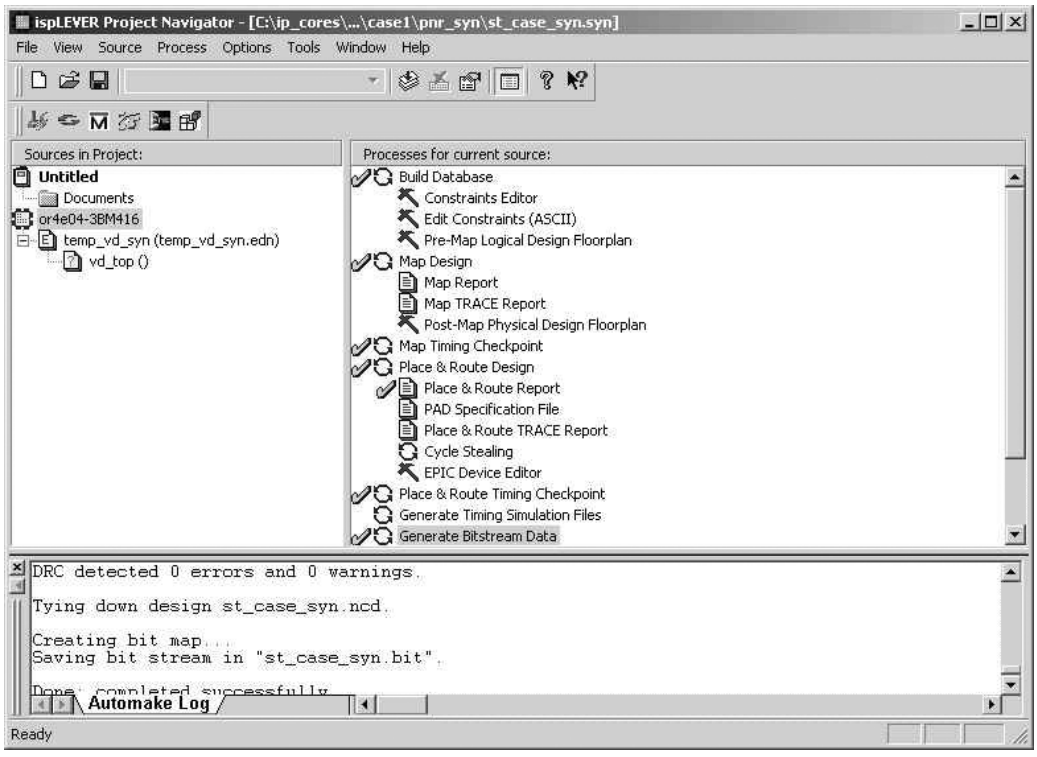


Figure 22. Import EDIF Dialog



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 23).

Figure 23. Project Navigator Window with Successfully Completed Processes for ORCA 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 23).

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 23).

11. Double click on **Generate Timing Simulation Files**. This generates the files for the optional back-end timing simulation (project_name.v and project_name.sdf). If there are no errors in this process, a green mark appears next to the **Generate Timing Simulation** process (Figure 23).

12. Double click on **Generate Bitstream Data**. This generates the bitstream file used to program the target device. If there are no errors in this process, a green mark appears next to the **Generate Bitstream Data** process (Figure 23).

ispLEVER Software Flow for ispXPGA Devices

The following steps illustrate ispLEVER software flow for XPGA 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 18 shows the ispLEVER **Project Navigator Window**.

4. Select **File > New Project** and create or browse to the project directory. All the design files will be generated in this directory. Figure 19 shows the **New Project Creation Dialog Box**. Type the 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 20 shows the **Device Selection Dialog Box**.

6. Select **Source > Import**. The **Import File Dialog Box** as shown in Figure 21. Enter the EDIF filename and click **Open**. The **Import EDIF Dialog Box** as shown in Figure 22. Select the EDIF vendor type and click **OK**. The EDIF file will then be added to the project.

7. Double click on the **Build Database**. This will read the EDIF netlist and generate the design database. If there are no errors in the process, you will see a green mark next to the **Build Database** process (Figure 24).

8. Double click on the **Constraint Editor**. You can enter the required preferences including the pin locking using the Constraint Editor. You can also create ASCII constraint file using a text editor. For additional details on the XPGA constraints, refer to 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 (Figure 24).

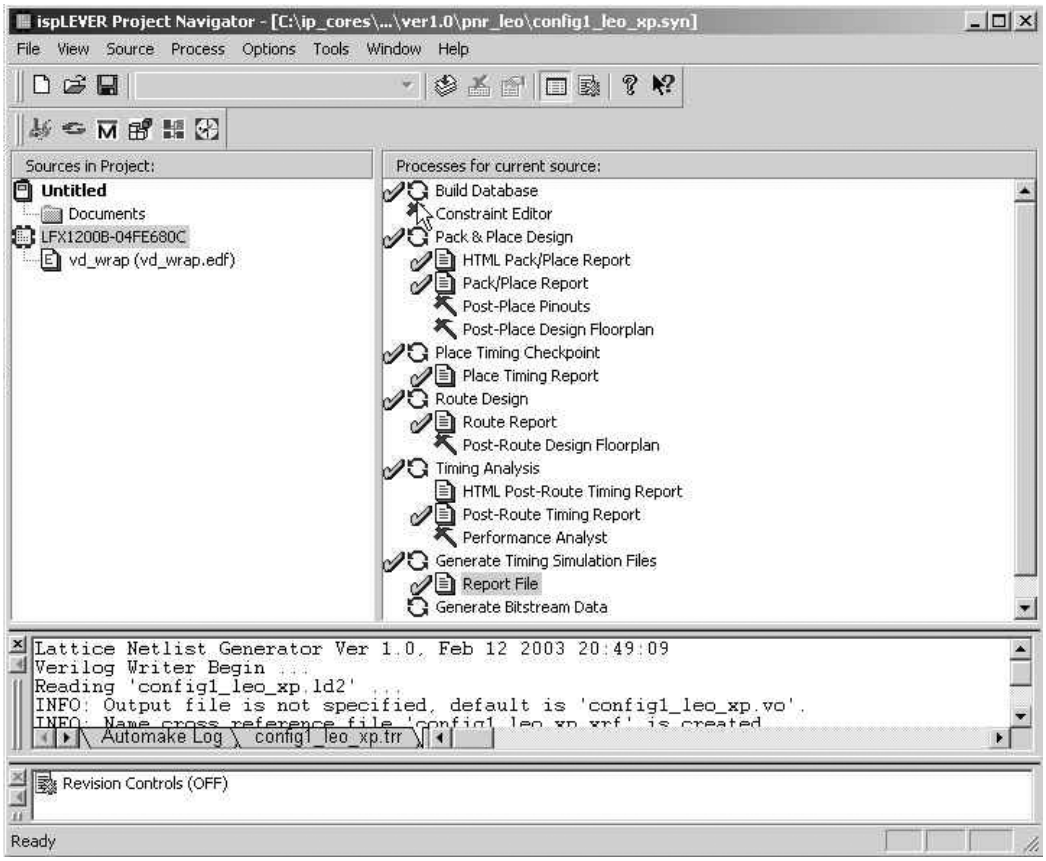
10. Double click on the **Route Design**. This will route the pack and placed design. If there are no errors in the process, you will see a green mark next to the **Route Design** process (Figure 24).

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 (Figure 24).

12. Double click on the **Generate Timing Simulation Files**. This will generate the files (*project_name*.vo and *project_name*.sdf) for the back-end timing simulation. If there are no errors in the process, you will see a green mark next to the **Generate Timing Simulation** process (Figure 24).

13. Double click on the **Generate Bitstream Data**. This will generate the bitstream file for programming. If there are no errors in the process, you will see a green mark next to the **Generate Bitstream Data** process (Figure 24).

Figure 24. Project Navigator Window with Successfully Completed Processes for ispXPGA Flow



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
Code Rate	2
Traceback Length	42
Generator Polynomials	
GP0 (Octal)	171
GP1 (Octal)	133
GP2 (Octal)	—
Implementation Method	
Parallel/Hybrid	Parallel
Hybrid Index	—
Decoder Inputs	
Hard/Soft Decision	Soft
Soft Width	3
Data Type	Signed
Punctured Data Support	
Punctured Decoder	No
Punctured Block Size	—
Puncture Pattern - PP0	—
Puncture Pattern - PP1	—
BER	
BER Monitor	No
BER Period	—

Supplied Netlist Configurations

The Ordering Part Number (OPN) for all configurations of this core in ORCA Series 4 devices is VTERB-DECO-04-N1. Table 5 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
vterb_deco_04_1_001.lpc	534	2104	997	10	8	71	174

1. Performance and utilization characteristics using ispLEVER® software and targeting the or4e04, package BM416, speed 2.
2. Programmable Function Unit (PFU) is a standard logic block of Lattice devices. For more information, check the data sheet of the device.

Appendix for ispXPGA FPGAs

Table 6. Core Configurations for ispXPGA FPGAs

	Configuration #1
Constraint Length	7
Code Rate	2
Traceback Length	42
Generator Polynomials	
GP0 (Octal)	171
GP1 (Octal)	133
GP2 (Octal)	—
Implementation Method	
Parallel / Hybrid	Parallel
Hybrid Index	—
Decoder Inputs	
Hard / Soft Decision	Soft
Soft Width	3
Data Type	Signed
Punctured Data Support	
Punctured Decoder	No
Punctured Block Size	—
Puncture Pattern -PP0	—
Puncture Pattern -PP1	—
BER	
BER Monitor	No
BER period	—

Supplied Netlist Configurations

The Ordering Part Number (OPN) for all configurations of this core in ispXPGA devices is VTERB-DECO-XP-N1. Table 7 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]

Config #	XPGA PFUs[2]	LUT-4s	Registers	External I/Os	SysMem EBRs	f_{MAX} (MHz)	Latency
vterb_deco_xp_1_001.lpc	1020	2879	1422	10	16	72	174

1. Performance and utilization characteristics are generated using LFX1200B, package FE680, speed 4 in Lattice's 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 devices. For more information, check the data sheet of the device.