latest version of the relevant information to establish, before ordering, that the information being relied upon is current.

**Type Conventions Used in This Document**

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning or Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Items in the user interface that you select or click. Text that you type into the user interface.</td>
</tr>
<tr>
<td><code>&lt;Italic&gt;</code></td>
<td>Variables in commands, code syntax, and path names.</td>
</tr>
<tr>
<td><strong>Ctrl+L</strong></td>
<td>Press the two keys at the same time.</td>
</tr>
<tr>
<td><strong>Courier</strong></td>
<td>Code examples. Messages, reports, and prompts from the software.</td>
</tr>
<tr>
<td>...</td>
<td>Omitted material in a line of code.</td>
</tr>
<tr>
<td>.</td>
<td>Omitted lines in code and report examples.</td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional items in syntax descriptions. In bus specifications, the brackets are required.</td>
</tr>
<tr>
<td>( )</td>
<td>Grouped items in syntax descriptions.</td>
</tr>
<tr>
<td>{ }</td>
<td>Repeatable items in syntax descriptions.</td>
</tr>
<tr>
<td></td>
<td>A choice between items in syntax descriptions.</td>
</tr>
</tbody>
</table>
Contents

Programming and Logic Analysis Tutorial  1
Introduction  1
  Learning Objectives  1
  Time to Complete This Tutorial  2
  System Requirements  2
  Accessing Online Help  3
  About the Tutorial Design  3
  About the Tutorial Data Flow  4
Task 1: Creating an EDIF-Type ispLEVER Project  6
  Creating a New Project with the Project Navigator  7
  Building the Design Files  11
  Examining Design Resources  13
Task 2: Generating and Adding the ispTRACY Core  14
  Linking the Core to the Design  20
  Re-Examining Design Resources  24
Task 3: Programming the Evaluation Board  26
  Placing and Routing the Design  27
  Generating the Bitstream  27
  Connecting to the Evaluation Board  29
  Downloading the Program  30
Task 4: Performing Logic Analysis  32
  Using One Shot Trace Mode  34
  Using Sample After Trigger Mode  43
  Restoring the Original Version of the Project  44
Summary  45
Glossary  45
Recommended Reference Materials  46
Programming and Logic Analysis Tutorial

Introduction

This tutorial describes IPexpress, the ispTRACY Core Linker, and the ispTRACY Logic Analyzer in ispLEVER, which all work together to add internal logic analysis and trace buffer logic to your design as part of your functional verification strategy. IPexpress and the ispTRACY Logic Analyzer enable you to evaluate the state of the internal signals within your device with minimal impact to scarce logic resources and I/Os. “Software-based” logic analyzers like ispTRACY provide a more economical and low-impact way to examine actual device behavior than traditional hardware logic analyzers running in a laboratory.

This tutorial also covers the basics of programming a LatticeEC LFEC20 device using the ispVM programming environment. For a more complete presentation on this topic, see the online help system for ispVM.

Learning Objectives

When you have completed this tutorial, you should be able to do the following:

- Understand the basic design flow, processes, and data files involved in incorporating an ispTRACY core into your design.
- Understand two distinct design flows possible to integrate an ispTRACY core: RTL instantiation or EDIF netlist.
- Define the basic trigger logic and trace buffer dimensions for an internal logic analysis.
- Link the internal ispTRACY logic analysis core interface to trigger and trace signals.
- Understand what additional resources are required to add internal logic analysis cores to your design.
- Program the standard LatticeEC evaluation board.
- Examine the internal signals using the ispTRACY Logic Analyzer waveform display.

**Time to Complete This Tutorial**
The time to complete this tutorial is about 60 minutes.

**System Requirements**
The following software is required to complete this tutorial:
- ispLEVER Starter or ispLEVER software with device support for the device used with your Lattice Semiconductor evaluation board
- Active licenses for Synplicity Synplify, Mentor Graphics Precision RTL Synthesis with VHDL support
- ispVM

This tutorial requires one of the following FPGA boards to examine the runtime operation of the tutorial design:
- LatticeEC Standard Evaluation Board (revision B)
- LatticeEC Advanced Evaluation Board (revision C)
- LatticeXP Standard Evaluation Board (revision B)
- LatticeXP Advanced Evaluation Board

In addition, this tutorial requires the following hardware to power and program an FPGA board:
- ispDOWNLOAD cable
- AC adapter
Accessing Online Help
You can find online help information on any tool included in the tutorial at any time by pressing the F1 key.

About the Tutorial Design
The tutorial design is a simple datapath-oriented design that includes a multiplexer, register bank, and a shift/load-type register. Data for the design is supplied by an internal 10-bit counter that provides a count pattern of 0-1023 on path “a” and 1023-0 on path “b.” The objective of the tutorial is to introduce a logic analysis module that will monitor signals “a,” “b,” and reg_out and demonstrate techniques to trigger and trace signals. Figure 1 shows the schematic of the tutorial design.

Figure 1: Tutorial Schematic

Table 1 summarizes the design’s functionality.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>sel</th>
<th>r_l</th>
<th>reg_out</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-63</td>
<td>1023-960</td>
<td>0</td>
<td>0</td>
<td>a</td>
<td>0-63</td>
</tr>
<tr>
<td>64-127</td>
<td>959-896</td>
<td>1</td>
<td>0</td>
<td>b</td>
<td>959-896</td>
</tr>
<tr>
<td>128-1023</td>
<td>895-0</td>
<td>1</td>
<td>1</td>
<td>b</td>
<td>q[9:0] = q[8:0,9]</td>
</tr>
</tbody>
</table>

The up/down counter logic controls the select (sel) signal, which determines which 10-bit datapath, “a” or “b,” is passed through the multiplexer. The internal logic specifies “a” when the counter module value is 0-63 and “b” if it is 64-127. The rotate/load (r_l) signal controls the Rotate module. It is active when signal “a” is greater than 127. Given the counter sequence, the initial state before the Rotate occurs is 897.
About the Tutorial Data Flow

Two flows are available in ispTRACY for instantiating ispTRACY logic analysis cores: the RTL instantiation flow and the EDIF netlist flow. This tutorial demonstrates the EDIF flow. The primary distinction between this flow and the RTL instantiation flow is that the EDIF flow automates the inclusion of the core logic into your design and eliminates a second synthesis process in merging the original design logic with the core logic. The EDIF flow is faster and involves less effort. The RTL instantiation of the ispTRACY core into your design, however, facilitates unified design documentation. For more information on these two flows, see the online help for the ispTRACY Core Linker and the ispTRACY Logic Analyzer. The flows are also described in detail in the "ispTRACY Core Generation and Logic Analysis" chapter of the FPGA Design Guide.

The flow chart shown in Figure 2 illustrates the sequence of ispLEVER tools used to define, link, and implement an ispTRACY core in the tutorial design. It begins with a predefined EDIF netlist of the tutorial design, vhdlhdesign.edf. IPexpress is used to define and generate a JTAG module containing an ispTRACY core called t3_la.ngo. A new version of the design (vhdlhdesign.ngo), including a reference to the ispTRACY core added, is processed through the normal Map Design, Place & Route Design, and Generate Bitstream Data processes to create a programming file.

Note

You can also generate ispTRACY cores by using MATLAB/Simulink, but this tutorial does not discuss this method. For information on this process, see the “DSP Design with MATLAB/Simulink for LatticeECP FPGAs” chapter in the FPGA Design Guide or refer to the MATLAB/Simulink online help topic.
Figure 2: Tutorial Design Flow for Defining and Linking ispTRACY Core

- **Build Database**
  - vhdlhdesign.edf
- **IPexpress**
  - .lpc file
  - .ngo file
- **ispTRACY Core Linker**
  - vhdlhdesign.ngo
  - t3_la.ngo
  - t3_la.core added
- **Map Design**
  - vhdlhdesign.ncd
- **Place and Route Design**
  - vhdlhdesign.bit
Task 1: Creating an EDIF-Type ispLEVER Project

In this task, you will create an EDIF-type project for the LatticeEC standard evaluation board device, add the tutorial design source file, import predefined constraints for location and timing preferences, and use the Build Database process to create a Lattice logical object file (.nog) in preparation for adding an ispTRACY core.

Note:
The design flow supports HDL-based projects in the Project Navigator; however, this approach requires resynthesis of your source files when an ispTRACY core is added to your design. Using an EDIF-based flow is a lower-impact method of adding internal ispTRACY cores to your design.

Although the tutorial tasks and figures illustrate the LatticeEC standard evaluation board, the procedures and preferences supplied with the tutorial are compatible with LatticeEC advanced or the LatticeXP standard/advanced builds of the FPGA board.
Creating a New Project with the Project Navigator

The first step is to create a new project in the Project Navigator.

To create a new project:

1. Start the ispLEVER system, if it is not already running.
2. In the Project Navigator, select **File > New Project** to open the Project Wizard dialog box.
3. In the Project Wizard dialog box, shown in Figure 4, select or specify the following:
   a. In the Project Name box, enter **VHDLhdesign**.
   b. In the Location box, enter the following directory:
      ```
      <install_path>\examples\tutorial\isptracy_tutor
      ```
   c. In the Design Entry Type box, select **EDIF**.
   d. In the Synthesis Tools box, select **Precision**.
   e. In the Simulator Tools box, select **ModelSim**.
   f. Click **Next**.

**Note:**
If you want to preserve the original tutorial design files, save the isptracy_tutor directory to another location on your computer before proceeding.

**Figure 4: Project Wizard Dialog Box**

4. In the Project Wizard - Select Device dialog box, shown in Figure 5, do the following:
a. In the Family box, choose **LatticeEC** or the appropriate device family for your build of the FPGA board.

b. In the Device box, choose **LFEC20E** or the appropriate device for your build of the FPGA board.

c. In the Speed grade box, choose **-4**.

d. In the Package Type box, choose **FBPGA484** or the appropriate package for your build of the FPGA board.

e. In the Operating Conditions box, choose **Commercial**.

f. Click **Next** to open the Project Wizard - Add Source dialog box.

---

**Figure 5: Project Wizard – Select Device Dialog Box**

5. In the Project Wizard - Add Source dialog box, click **Add Source** to activate the Import File (EDIF) dialog box.
6. In the File Name box in the Import File (EDIF) dialog box, shown in Figure 6, select vhdlhdesign.edf and click Open.

**Figure 6: Import File (EDIF) Dialog Box**

7. In the Project Wizard - Add Source dialog box, shown in Figure 7, click Next.

**Figure 7: Project Wizard – Add Source Dialog Box**
8. In the Project Wizard - Project Information box, shown in Figure 8, click Finish.

**Figure 8: Project Wizard – Project Information Dialog Box**
The new EDIF-type project is created with an EDIF netlist of the tutorial design added to the Sources in Project window, as shown in Figure 9.

**Note:**
Click on the part name to see the contents of the Processes for Current Source window.

![Figure 9: New Project in ispLEVER Window](image)

**Building the Design Files**
In this task, you will build the design files and generate an .ngd file. But first you will learn how to create a backup project revision that you can use to restore the original version of the project later in the tutorial.

To build the design files:
1. In the Project Navigator, highlight the device name (LFEC20E-5F484C) in the Sources in Project window.
2. Click on the Revision Control icon to turn on revision control.
3. Double-click on **Build Database**.

Information and warning messages appear in the Automake Log tab of the output panel of the Project Navigator. You can ignore these warnings in
IspLEVER creates an implementation in Project Rev01 of the revision panel. Rev01 will be restored later after the debugging session.

The Build Database process translates the synthesis output in EDIF to an NGD logical design database. Information and warning messages appear in the Automake Log tab of the output panel of the Project Navigator. You can ignore these warnings in this tutorial.

4. Turn off Revision Control by right-clicking in the Revision Control window at the right of the Project Navigator and choosing Turn Off from the popup menu.

The Project Navigator window should now look like the illustration in Figure 10.

**Figure 10: Project Navigator After Building Database**
Examining Design Resources

You can determine the specific physical resources applied to accommodate the tutorial design. You can use the Map Report for this purpose. Later you will compare this baseline report to the report containing the data generated by the additional logic required to represent the ispTRACY core and related trace memory.

To view the map report:

- In the Project Navigator, double-click on Map Report (vhdlhdesign.mrp).

The Design Summary section of the Map Report looks like the sample shown in Figure 11.

Figure 11: Design Summary Section of the Map Report

Design Summary
-----------------

Number of registers: 30
  PFU registers: 30
  PIO registers: 0
Number of SLICEs: 16 out of 9856 (0%)
  SLICEs(logic): 16 out of 7392 (0%)
  SLICEs(logic/RAM): 0 out of 2464 (0%)
  As RAM: 0
  As Logic: 0
Number of logic LUT4s: 22
Number of distributed RAM: 0 (0 LUT4s)
Number of ripple logic: 5 (10 LUT4s)
Number of shift registers: 0
Total number of LUT4s: 32
Number of external PIOs: 12 out of 360 (3%)
Number of PIO IDDR/ODDR: 0
Number of 3-state buffers: 0
Number of PLLs: 0 out of 4 (0%)
Number of Block RAMs: 0 out of 46 (0%)
Number of GSRs: 1 out of 1 (100%)
JTAG used: No
Readback used: No
Oscillator used: No
Startup used: No

Note

The Map Design process applies device-specific location and sysIO buffer types supplied in a pre-defined project preference file (vhdlhdesign.lpf). Lattice Semiconductor supplies the .lpf file with location (LOCATE) preferences appropriate for the LatticeEC Standard Evaluation Board (Revision B) User’s Guide. If you are using another build of the FPGA board, modify vhdlhdesign.lpf with the Edit Preferences (ASCII) tool. To modify the file for your device, locate the "# Pin assignments" section for your target device and uncomment the LOCATE preferences. To avoid conflicts, remember to delete or comment out the LOCATE preferences for all other target device pin assignments.
Task 2: Generating and Adding the ispTRACY Core

In this task, you will use the IPexpress tool to configure an ispTRACY core based on triggering conditions and the desired trace buffer. Then you will use the ispTRACY Core Linker to connect or link signals of your design to the core. The primary output of the Core Linker is a modified version of your design with one or more cores instantiated and the core logic ready for mapping, placement, and routing.

Figure 12 is a block diagram of the ispTRACY core. The primary interface consists of two buses: the trigger bus, TRIG_SIG\_n, and the trace bus, TRACE_SIG\_n. Depending on the core parameters that you specify in IPexpress, the core can accommodate many different trigger and trace scenarios and trace memory depth. The number of logic analyses, trigger logic options that you enable, and the size of the trace memory that you specify influence what additional FPGA resources will be required to accommodate this additional debugging logic.

Figure 12: ispTRACY Core

![ispTRACY Core Diagram]

where \( n \) = number of cores

To generate a JTAG module containing the ispTRACY core:

1. In the Project Navigator, choose Tools > IPexpress or click the button on the Project Navigator tool bar.
The main IPexpress main window opens, as shown in Figure 13.

**Figure 13: IPexpress Main Window**

2. In the upper left corner of the main window, click the + sign next to “Module,” then click the + sign next to “Architecture_Modules” to expand the module tree.

3. Click **Jtag**.
Most of the boxes in the IPexpress main window are now filled, as shown in Figure 14.

Figure 14: IPexpress Main Window Showing JTAG Core

4. In the IPexpress main window, do the following:
   a. In the Project Path box, verify that the path is set to the following:
      c:\isptools\examples\tutorial\isptracy_tutor
   b. In the File Name box, type t3_la.
   c. Click Customize.
The Lattice Module - JTAG dialog box now opens with the ispTRACY Cores tab of the Configuration tab selected, as shown in Figure 15.

**Figure 15: ispTRACY Cores Tab of the Configuration Tab of the Lattice Module - JTAG Dialog Box**

5. Select the **Configuration** tab, if it is not already selected.

6. In the Configuration tab, select the **ispTRACY Cores** tab, if it is not already selected.

7. Verify that the **Enable ispTRACY IP** option is turned on (it is turned on by default.)

8. Verify that the **JTAG interface logic** option is turned on (it is turned on by default.)
9. Set the parameters as given in the JTAG Core Parameters column of Table 2:

### Table 2: Specifying JTAG Core Parameters

<table>
<thead>
<tr>
<th>JTAG Core Parameters</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Cores: 1</td>
<td></td>
</tr>
<tr>
<td>Size Comparison Logic: <strong>ON</strong></td>
<td>“On” provides comparisons for:</td>
</tr>
<tr>
<td></td>
<td>equal to (=)</td>
</tr>
<tr>
<td></td>
<td>not equal to (&lt; &gt;)</td>
</tr>
<tr>
<td></td>
<td>greater than (&gt;)</td>
</tr>
<tr>
<td></td>
<td>greater than or equal to (&gt;=)</td>
</tr>
<tr>
<td></td>
<td>less than (&lt;)</td>
</tr>
<tr>
<td></td>
<td>less than or equal to (&lt;=)</td>
</tr>
<tr>
<td></td>
<td>“Off” minimizes the amount of trigger logic and provides equal-to or not-equal-to functionality.</td>
</tr>
<tr>
<td>Event Counter Size: <strong>NONE</strong></td>
<td>“None” indicates that a counter will not be created to store the number of trigger events that occur before the external trigger signal is activated.</td>
</tr>
<tr>
<td>Trigger Same as Trace: <strong>ON</strong></td>
<td>“On” indicates that the trigger and trace bus width is the same.</td>
</tr>
<tr>
<td>Trace Bus Size: <strong>12</strong></td>
<td>“12” indicates that up to twelve unique signals can be connected to the trace memory.</td>
</tr>
<tr>
<td>Trace Memory Depth: <strong>512</strong></td>
<td>“512” specifies that the depth of the memory and trigger mode is 512 samples. This option determines the amount of sample data that will be available for examination. It can be set to 512, 1024, 2048, or 4096, depending on what embedded block RAM resources are still available after your design logic has been allocated.</td>
</tr>
<tr>
<td>Sample After Trigger Mode Logic: <strong>ON</strong></td>
<td>“On” controls the number of samples captured per trigger. In this configuration, the Trace Memory Depth behaves as a circular, FIFO-like memory organization to store trace data. When trigger events occur, a “window” of state activity before and after the event is saved.</td>
</tr>
<tr>
<td>Number of Edge Trigger Signals: <strong>0</strong></td>
<td>“0” indicates that none of the trigger bus bits are edge-sensitive. The sum of the number of edge-trigger signals and the number of level-trigger signals must equal the trigger bus size.</td>
</tr>
</tbody>
</table>
### Table 2: Specifying JTAG Core Parameters

<table>
<thead>
<tr>
<th>JTAG Core Parameters</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger Input Logic: OFF</td>
<td>“Off” indicates that internal conditions rather than a signal or external I/O will specify when a trigger occurs.</td>
</tr>
<tr>
<td>Trigger Output Logic: ON</td>
<td>“On” indicates that an additional I/O pin will be added to the design to communicate with the ispTRACY Logic Analyzer.</td>
</tr>
</tbody>
</table>

The ispTRACY Cores tab of the Configuration tab of the Lattice Module - JTAG dialog box should resemble the illustration in Figure 16.

**Figure 16: Updated ispTRACY Cores Tab of the Configuration Tab of Lattice Module - JTAG Dialog Box**

10. Click **Generate**.

IPexpress synthesizes the core according to the parameters that you set and creates an EDIF netlist. Next, it translates the EDIF file into a Lattice .ngo object-format file in preparation for linking the core to the larger design. In addition, it generates a Lattice parameter configuration (.lpc) file, which you will need later in the tutorial in order to use the ispTRACY Logic Analyzer.
IPexpress activates the Generate Log tab of the Lattice Module - JTAG dialog box, as shown in Figure 17.

**Figure 17: Generate Log Tab of Lattice Module - JTAG Dialog Box**

11. Click **Close**.

12. Close the IPexpress window by choosing **File > Exit**.

**Linking the Core to the Design**

In this procedure, you will use the ispTRACY Core Linker to connect the design's internal signals to the ispTRACY core that you just generated.

*To link the ispTRACY core to your design's internal signals:*

1. In the Project Navigator, choose **Tools > ispTRACY Core Linker** or click the **button.
The ispTRACY Core Linker window appears, as shown in Figure 18.

Figure 18: ispTRACY Core Linker Window

2. Choose **File > Open**.

3. In the Open dialog box, select the **t3_la.lpc** file and click **Open**.

4. Select the **vhdlhdesign** folder in the upper left corner. All signals within the top of the design hierarchy now appear in the Selected Signals column.

5. In the ispTRACY Ports (right) column, open the **ispLA0** folder by clicking on the + next to it.

6. Click on the + next to the **Clock** folder in the Ports column.

7. Select **clk_int** in the Selected Signals column and **Sample_CLK_0** in the ispTRACY Ports column and click **Connect**.

   The ispTRACY Ports column is updated to show the association between the ispTRACY core’s sample clock and the design’s internal clock.
8. Click on the + next to the Trace folder in the Ports column, and use the technique given in the previous step to connect the remaining design and trace signals as shown in Table 3:

<table>
<thead>
<tr>
<th>Design Signals</th>
<th>Trace Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg_out(0)</td>
<td>TRACE_SIG_0(0)</td>
</tr>
<tr>
<td>reg_out(1)</td>
<td>TRACE_SIG_0(1)</td>
</tr>
<tr>
<td>reg_out(2)</td>
<td>TRACE_SIG_0(2)</td>
</tr>
<tr>
<td>reg_out(3)</td>
<td>TRACE_SIG_0(3)</td>
</tr>
<tr>
<td>reg_out(4)</td>
<td>TRACE_SIG_0(4)</td>
</tr>
<tr>
<td>reg_out(5)</td>
<td>TRACE_SIG_0(5)</td>
</tr>
<tr>
<td>reg_out(6)</td>
<td>TRACE_SIG_0(6)</td>
</tr>
<tr>
<td>reg_out(7)</td>
<td>TRACE_SIG_0(7)</td>
</tr>
<tr>
<td>reg_out(8)</td>
<td>TRACE_SIG_0(8)</td>
</tr>
<tr>
<td>reg_out(9)</td>
<td>TRACE_SIG_0(9)</td>
</tr>
<tr>
<td>sel</td>
<td>TRACE_SIG_0(10)</td>
</tr>
<tr>
<td>r_l</td>
<td>TRACE_SIG_0(11)</td>
</tr>
</tbody>
</table>

**Note:**

You can unlink existing linked signals by selecting the signals in the right pane and clicking **Disconnect.**

Refer to the schematic in "About the Tutorial Design" on page 3 for signal names.

**Note:**

To ensure that internal signal names are preserved through the synthesis process, you may need to apply special compiler directives to your design so the names are not changed. For more information, see the Mentor Graphics or Synplicity synthesis tool user guides.
The ispTRACY Core Linker window should resemble the illustration shown in Figure 19.

Figure 19: ispTRACY Core Linker Window Showing Connected Signals


10. Choose File > Set Active in ispLEVER Project.

A message box now appears with the following message:

```
t3_la.lpf has been generated. Please copy its contents into your preference file.
```

11. Click OK, and copy the contents of the t3_la.lpf core preference file into the project preference file:

   a. In the Project Navigator, choose the Window > Text Editor.

   b. Open vhdlhdesign.lpf and t3_la.lpf. (You may need to click on All Files in the Files of Type box to see these file names.)

   c. Copy the contents of t3_la.lpf and add them to the contents of vhdlhdesign.lpf.
12. In the ispTRACY Core Linker, choose **File > Exit**.

13. In the message box containing the following message, click **OK**:

   This module was set as active module in ispLEVER project.

Figure 20 illustrates the modified design with the t3_la module included. Trigger and trace signals are tapped from reg_out[9:0], and a new external port, trig_out_0, is added to the design’s interface.

**Figure 20: Modified Design with t3_la Module**

---

**Re-Examining Design Resources**

Before proceeding, you may want to examine the resources required to accommodate the ispTRACY core. As you did earlier, you can view the Map Report for this purpose.

*To view the Map Report:*

- In the Project Navigator, double-click on Map Report (vhdlhdesign.mrp).

The original design requires 14 LatticeEC slice resources. The Design Summary section of the report, shown in Figure 21, indicates that four block RAMs are applied to accommodate the trace buffer, and approximately 184 additional slice resources are applied to accommodate the trigger and trace logic of the ispTRACY core.
Figure 21: Design Summary Section of Map Report

Design Summary

Number of registers: 339
   PFU registers: 339
   PIO registers: 0
Number of SLICEs: 231 out of 9856 (2%)
   SLICEs(logic): 217 out of 7392 (3%)
   SLICEs(logic/RAM): 14 out of 2464 (0%)
      As RAM: 14
      As Logic: 0
Number of logic LUT4s: 186
Number of distributed RAM: 14 (28 LUT4s)
Number of ripple logic: 20 (40 LUT4s)
Number of shift registers: 0
Total number of LUT4s: 254
Number of external PIOs: 13 out of 360 (4%)
Number of PIO IDDR/ODDR: 0
Number of 3-state buffers: 0
Number of PLLs: 0 out of 4 (0%)
Number of Block RAMs: 1 out of 46 (2%)
Number of GSRs: 1 out of 1 (100%)
JTAG used: Yes
Readback used: No
Oscillator used: No
Startup used: No
The Project Navigator now looks like the picture in Figure 22.

Figure 22: Project Navigator After Mapping

Task 3: Programming the Evaluation Board

In this task, you will place and route the design and generate a bitstream (.bit) file to produce a programming file for an EC6, EC20, ECP20, or XP10 device on the appropriate evaluation board.

Using the guidelines of the appropriate board user’s guide, such as the LatticeEC Standard Evaluation Board (Revision B) User’s Guide, you will connect the ispDOWNLOAD cable between the JTAG header of the board and the parallel port of your system. You will run the ispVM programming environment to download the bitstream created in the last task to program the device.

The vhdlhdesign.lpf file supplied with the tutorial contains location preferences for the following evaluation boards:

- LatticeEC standard evaluation board (revision B)
- LatticeEC advanced evaluation board (revision C)
Programming and Logic Analysis Tutorial

Task 3: Programming the Evaluation Board

- LatticeXP standard evaluation board (revision B)
- LatticeXP advanced evaluation board

**Placing and Routing the Design**
First, you place and route your design.

To place and route the design:
- Double-click **Place & Route Design**.

The results are shown in Figure 23.

![Figure 23: Project Navigator After Placement and Routing](image)

**Generating the Bitstream**
Now you are ready to generate the bitstream (.bit) file.

To generate a bitstream (.bit) file for the evaluation board:
- Double-click **Generate Bitstream Data**.
IspLEVER creates a programming file, vhdlhdesign.bit, that is ready for downloading into the device. Figure 24 shows the Project Navigator after bitstream generation.

Figure 24: Project Navigator After Bitstream Generation
Connecting to the Evaluation Board

A LatticeEC standard evaluation board is shown in Figure 25. This tutorial uses the LatticeEC evaluation board as an example, but you can use any of the boards listed at the beginning of “Task 3: Programming the Evaluation Board” on page 26 with the appropriate device.

Figure 25: Standard LatticeEC Evaluation Board

To connect to the LatticeEC evaluation board:

1. Install a driver for the download cable.
2. Reboot your computer.
3. Attach the parallel port or USB ispDOWNLOAD cable to the parallel port or USB port of your system.
4. In the Project Navigator, select Tools > ispVM System or click the button on the toolbar.
5. Select Options > Cable and I/O Port Setup.
6. Click Auto Detect, then click OK.
7. Attach the 1 x 8 JTAG connector ispDOWNLOAD cable to JP1 (1x10) of the JTAG programming header of the evaluation board. Justify the alignment of pin 1 (VCC) of the header to the VCC lead of the cable.
8. Align the on-board oscillator so that pad V1 is driven. The 16-pin socket will allow connection to PLL clock pin V1 when the bottom of the oscillator is aligned to socket pins 8 and 9.
9. Plug in the AC adapter to a wall outlet, and plug the other end into the power jack at J31.

**Note**

You should follow the handling and power-up advice provided in the *LatticeEC Standard Evaluation Board (Revision B) User’s Guide* when using the evaluation board.

**Downloading the Program**

This task illustrates the procedure for downloading the programming bitstream to a LatticeEC standard evaluation board. The procedure is similar when you use the LatticeEC advanced or the LatticeXP standard or advanced builds of the FPGA board. Substitute the appropriate device for your FPGA board in the following steps.

To download the program to the LatticeEC evaluation board:

1. In the ispVM System interface, select **File > New**. A new chain configuration window appears.
2. Choose **ispTools > Scan Chain** or click the Scan toolbar icon. ispVM detects a single EC6E, EC20E, or ECP20E device in the chain and adds it to the list.
3. Select the first device in the New Scan Configuration Setup list.
4. In the popup box labeled Multi Match Device’s ID List, select **EC6E**, **EC20E**, or **ECP20E**.
5. Highlight the selected device—**EC6E**, **EC20E**, or **ECP20E**—right-click, and choose **Edit Device** from the popup menu.

The Device Information dialog box appears, as shown in Figure 26.

**Figure 26: Device Information Dialog Box**
6. Click the Select button of the Device section to open the Select Device dialog box.

7. Select the following:
   - Device Family: LatticeEC
   - Device: LFEC6E, LFEC20E, or LFECP20E, as appropriate for your revision of the LatticeEC standard evaluation board
   - Package: 484-ball fpBGA

8. Click OK.

9. Click the Browse button of the Data File section.

10. Select the vhdlhdesign.bit file, and click Open.

11. In the Operation box, select Fast Program, if it is not already selected.

12. Click OK to close the Device Information dialog box.

13. Choose Project > Download, or click the GO button on the toolbar.

   After a few moments, the download and programming activity will end. A green PASS button appears in the New Scan Configuration Setup dialog box, shown in Figure 27.

Figure 27: New Scan Configuration Setup Dialog Box

14. Select File > Save As to save the configuration setup as an .xcf file.

15. In the File Name box in the dialog box that appears, type in vhdlhdesign.xcf, and click Save.

**Task 4: Performing Logic Analysis**

In this task, you will use the ispTRACY Logic Analyzer to set up trigger conditions and view trace buffer data from the on-chip ispTRACY core operating within the device on the LatticeEC standard evaluation board. The trigger setup influences under what specific conditions and how the ispTRACY core trace signal states are displayed in the ispTRACY Logic Analyzer’s graphical user interface. In this task, you will explore just a few of the many ways to trigger and trace the system.

**To set up the triggering conditions for the ispTRACY core display:**

1. In the Project Navigator, choose **Tools > ispTRACY Logic Analyzer** or click on the button of the toolbar.
2. Choose **Device > Connection Setup**.
3. Check the settings in the Cable and I/O Port Setup dialog box, shown in Figure 28, to be sure that they are appropriate for your board and cable. Reset any if necessary, then click **OK**.

**Figure 28: Cable and I/O Port Setup Dialog Box**

4. In the ispTRACY Logic Analyzer window, choose **File > New**.
5. Specify the following in the New Project dialog box, shown in Figure 29:
   a. In the Project Name box, enter **VHDLhdesign**.
   b. In the Project Directory box, enter the following path:
      ```
      <install_path>\examples\tutorial\isptracy_tutor
      ```
c. Click the **Next** button.

**Figure 29: New Project Dialog Box**

6. In the Device Information dialog box, shown in Figure 30, click the **Browse** button.

7. In the Open dialog box, highlight the **t3_la.lpc** trace configuration file, and click **Open**.

8. In the Device Information dialog box, click **Finish**.

**Figure 30: Device Information Dialog Box**
Using One Shot Trace Mode

One Shot is an option in the Trace Mode section of the Trigger Setup tab of the ispTRACY Logic Analyzer. The way it works is illustrated in Figure 31. The trace memory provides a window onto the large number of potential data states. The tutorial uses the smallest memory organization of 512 words. You can use the Position option to determine the relative amount of past or future samples that you would like to examine relative to the trigger position.

Figure 31: Determining Number of Samples

The Pre-Trigger setting is helpful if you care mostly about data states that occurred after the trigger. With this setting, only 32 samples of data that occur before the trigger are stored; however, 480 samples of data that occur after the trigger are stored. The Post-Trigger setting provides the most trace data on states that occurred before the trigger event, and the Center-Trigger setting provides equal amounts of trace data.

In some cases, the trace memory may accommodate all the potential data states of your design. The ispTRACY Logic Analyzer simply positions the trigger in the window at the first occurrence of the trigger event without the Position option influencing the position.

To use One Shot mode:

1. Choose **Window > Show ispLA Window > Device0 > Device0 LA0**.
   A dialog box appears for LA0 with Trigger Setup, Event Patterns, Signal Analysis, and Data Listing tabs.
2. Select the **Trigger Setup** tab.
3. Specify the following trigger conditions for the ispTRACY core, as shown in Table 4:

**Table 4: Specifying Trigger Conditions**

<table>
<thead>
<tr>
<th>Trigger Setup</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Trace Mode:</strong></td>
<td><strong>One Shot</strong> with a “Pre-Trigger” position indicates that the trace memory will hold mostly samples of the trace bus signals that occur after the trigger event. Trace Mode enables you to capture more or less data before or after the trigger event.</td>
</tr>
<tr>
<td><strong>Pre-Trigger</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Compare Mode:</strong></td>
<td>Trigger events 0 (EV0) and 1 (EV1) indicate that the trigger comparison logic will use equivalency logic of “=” for the trigger condition.</td>
</tr>
<tr>
<td>■ EV0 Is Defined As Trigger Bus = Pattern_0</td>
<td></td>
</tr>
<tr>
<td>■ EV1 Is Defined As Trigger Bus = Pattern_1</td>
<td></td>
</tr>
<tr>
<td><strong>Trigger Condition:</strong></td>
<td>Set the <strong>After</strong> box to <strong>EV0</strong>. Set the field to the right of <strong>EV0</strong> to <strong>--</strong>.</td>
</tr>
<tr>
<td></td>
<td>Set the <strong>Wait for</strong> box to the right of the <strong>After</strong> box to <strong>EV1</strong>. Set the field to the right of <strong>EV1</strong> to <strong>--</strong>.</td>
</tr>
<tr>
<td><strong>Trigger Output Polarity: High</strong></td>
<td>“Wait for EV0, Wait for EV1” indicates that the trigger comparison will wait for Pattern_0 to occur before Pattern_1 causes a trigger event.</td>
</tr>
<tr>
<td></td>
<td>“High” indicates that the trigger signal coming from outside the core should be set to active high.</td>
</tr>
</tbody>
</table>
The Trigger Setup tab should resemble the illustration in Figure 32.

**Figure 32: Trigger Setup Tab**

4. Click the **Event Patterns** tab.

5. Holding down the Shift key, select reg_out(0) through reg_out(9) in the Trigger Bus/Signal Name column.

6. Click the right mouse button, and choose **Group Into Bus**. Bus0 appears in the list.

7. Click the – symbol next to bus0 to collapse the individual bits.

8. Select and click the right mouse button on bus0.

9. Select **Set Bus Radix**.

   The Bus Radix dialog box now appears, as shown in Figure 33.

10. Choose **Decimal** from the list.
11. Click **OK**.

**Figure 33: Bus Radix Dialog Box**

12. Specify the following trigger event patterns by repeatedly clicking on the Xs:

- **bus0, Pattern_0: 0512** (Before you enter 0512, place your cursor before the ???? pattern and press x. The ???? pattern becomes an XXXX pattern. Keeping the cursor placed before the XXXX pattern, type 0512.)
- **sel, Pattern_1: 0**
- **r_l, Pattern_1: 0**
The Event Patterns tab should resemble the illustration shown in Figure 34.

**Figure 34: Event Patterns Tab**

This combination of trigger setup and event patterns will cause the ispTRACY Logic Analyzer to wait until reg_out(9:0) reaches 512, then it will wait until r_1 and sel equal 0 before a trigger event is communicated to the ispTRACY Logic Analyzer.

13. Click the **Signal Analysis** tab.
14. Select reg_out(0) through reg_out(9) in the Trigger Bus/Signal Name column.
15. Click the right mouse button and choose **Group Into Bus**.
   
   Bus0 appears in the list.
16. Select **bus0**, and click the right mouse button.
17. Select **Set Bus Radix**.
18. Choose **Decimal** from the list.
19. Click **OK**.
20. Choose **Data > Set Clock Frequency**.
21. Specify **33.33** in the Frequency entry box, as shown in Figure 35.

**Figure 35: Specify Clock Frequency Dialog Box**

22. Click **OK**.

23. Select **Device > Run Current ispLA** or press the **RUN** button in the tool bar.

24. Click **OK** in the box notifying you that the process finished successfully.

25. Choose **Data > Zoom > Fit Window**.

The Signal Analysis window adjusts to show all the signal states of the trace memory, as shown in Figure 36. The trigger event is shown with the T marker at position 32. You can adjust the X and O marker to specific periods between events by clicking on a marker and dragging it to a new position. The status bar at the bottom of the tab displays the position in time of the X marker (2070.207 ns in this example), the O marker (4620.462 ns in this example, and the difference between them (2550.255 ns in this example).
The Pre-Trigger setting of the Position option in the Trace Mode section of the Trigger Setup tab has caused the trace buffer to fill with reg_out(9:0) activity primarily after the trigger event.

Figure 36: Signal Analysis Tab with T Marker at Position 32

26. Click the Trigger Setup tab and select **Center-Trigger** in the Position box.
27. Click the Signal Analysis tab, then press the **RUN** button.
The Signal Analysis display is updated, and the trigger marker appears at position 256 in the display, as shown in Figure 37.

**Figure 37: Signal Analysis Tab with T Marker at Position 256**

28. Click the Trigger Setup tab and select **Post-Trigger** in the Position box.
29. Click the Signal Analysis tab, then press the **RUN** button.
The display in the Signal Analysis tab is updated, and the trigger marker appears at position 480 of the display, as shown in Figure 38.

**Figure 38: Signal Analysis Tab with T Marker at Position 480**
Using Sample After Trigger Mode

Sample After Trigger is an option in the Trace Mode section of the Trigger Setup tab of the ispTRACY Logic Analyzer. It enables you to specify how many samples before the trigger and how many samples after the trigger you would like to examine. In this mode, the sampled trace bus data is stored to a trace buffer. When the trigger condition is met, the contents of the trace buffer are placed in the trace memory. The size of this trace buffer is defined by the number of samples before the trigger. After the trigger condition is met, the trace control block keeps storing the sampled trace bus data in the trace memory until the user-defined number of samples is stored. The trace stops only when the trace memory is full or you force it to stop. The number of samples before and after the trigger are independent.

In Figure 39, 32 samples per trigger are defined. As a result, up to 16 (512 divided by 32) unique event occurrences appear in the Signal Analysis display.

To use Sample After Trigger Mode:
1. Click the Trigger Setup tab.
2. Specify the options shown in Table 5:

**Figure 39: Defining Trigger Samples**

<table>
<thead>
<tr>
<th>Trace memory</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
</table>

**Table 5: Specifying Trigger Conditions**

<table>
<thead>
<tr>
<th>Trigger Setup Function</th>
<th>Trace Mode:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample After Trigger</td>
<td>Indicates the number of samples to take before and after the trigger point.</td>
</tr>
<tr>
<td>Samples - set to 32</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compare Mode:</th>
</tr>
</thead>
<tbody>
<tr>
<td>EV0 Is Defined As Trigger Bus = Pattern_0</td>
</tr>
<tr>
<td>EV1 Is Defined As Trigger Bus = Pattern_1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trigger Condition: Wait For</th>
</tr>
</thead>
<tbody>
<tr>
<td>EV0 OR EV1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trigger Output Polarity:</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
</tr>
</tbody>
</table>
3. Click the **Event Patterns** tab.
4. Set `reg_out(9:0)` to **X** (Don’t Care) in the Pattern_0 column.
5. Set `r_l` to **1** in the Pattern_0 column and `sel` to **1** in the Pattern_1 column.
6. Click the **Signal Analysis** tab.
7. Press **RUN**.
8. Choose **Data > Zoom > Fit Window**.

The screen now resembles the illustration in Figure 40.

---

**Figure 40: Signal Analysis Tab in Sample After Trigger Mode**

---

**Restoring the Original Version of the Project**

Restoring a version of your project that does not include the ispTRACY core is an option in a typical project, and you may want to retain the debugging hardware in the production version of your device. However, freeing the block RAM resources and LUT-based logic to expand your design and then removing the core might be desirable.

There are two distinct methods of restoring the original version of your project:

- The easiest and safest approach that Lattice recommends is to restore a prior revision of your project using the Project Navigator’s Revision Control features.
If you have not retained a prior revision of your project, you can remove the ispTRACY core from the current revision of the project by manually deleting all .tcy files from the local design directory and rerun (Force) the Build Database, Map Design, and Place & Route Design processes.

To exit the ispTRACY Logic Analyzer graphical user interface and restore the original version of the tutorial project:

1. In the ispTRACY Logic Analyzer, choose File > Exit.
2. In the ispVM window, choose File > Exit.
3. In the Project Navigator, turn on Revision Control.
4. Select Project Rev01 of the Revision Panel, click the right mouse button, and choose Set as Active Revision.

### Summary

You have completed the “Programming and Logic Analysis Tutorial.” In this tutorial, you have learned how to do the following:

- Generate an ispTRACY core and link it to the trigger and trace signals in the design by using IPexpress and the ispTRACY Core Linker, respectively.
- Generate a bitstream file for the target device on the LatticeEC standard evaluation board.
- Program a standard LatticeEC standard evaluation board with ispVM.
- Understand and configure the ispTRACY core trigger logic and trace memory array.
- Run the ispTRACY Logic Analyzer application to examine the sample captures in the internal trace memory array.
- Understand the resource impact of adding ispTRACY logic analysis resources to your design.

### Glossary

Following are the terms and concepts that you should understand to use this tutorial effectively.

**.bit file.** A .bit file is a binary-format configuration file containing the default outputs of the bit generation process. It is used to program a Lattice FPGA device.

**logic analysis core.** A logic analysis, or debugging, core is a precharacterized piece of logic (IP) specifically designed for logic analysis.

**.ncd file.** An .ncd file is a binary-format FPGA post-map physical design database file generated by the Map Design process of the Project Navigator.
The .ncd file includes mapping information and, potentially, placement and routing information.

**.ngd file.** An .ngd file is a binary-format FPGA pre-map logical design database file generated by the NGDBuild phase of the Build Database process in the Project Navigator. The NGD file represents the logical design information of the ASCII EDIF netlist.

**.ngo file.** An .ngo file is a binary-format FPGA pre-map logical design database file generated by the EDIF-to-NGD translation phase in the Build Database process in the Project Navigator. It contains all of the data in the input .ngd file, as well as information on the physical design produced by the mapping. The .ngo file typically represents one or more hierarchical branches of a larger logical design. In this tutorial, an .ngo file is produced for the ispTRACY logic analysis core.

**slice.** A slice is an architectural element within an FPGA consisting of two LUT4 lookup tables that feed two registers (programmed to be in FF or latch mode), and some associated logic that allows the LUTs to be combined to perform functions such as LUT5, LUT6, LUT7, and LUT8. It also includes control logic to perform set/reset functions (programmable as synchronous or asynchronous), clock select, chip-select and wider RAM/ROM functions. The registers in the slice can be configured for positive or negative and edge or level clocks. There are four interconnected slices per PFU block. Each slice in a PFU is capable of four modes of operation: logic, ripple, RAM, and ROM. Each slice in the PFF is capable of all modes except RAM.

**.tla file.** A .tla file the project file created by the ispTRACY Logic Analyzer. It contains the project name, core configuration trace and trigger signals, the trigger and event settings, and the ispTRACY chain acquisition data.

**trace.** A trace is one or more signal states that are monitored for debugging purposes. The ispTRACY Logic Analyzer enables you to monitor signal states that occur before and after a user-defined trigger pattern. IPexpress enables you to define the width and depth of a trace memory to hold the signal states.

**trigger.** A trigger is a specific state or range of states that cause a trace to be stored for review and debugging. The ispTRACY Logic Analyzer enables you to establish different scenarios for a trigger condition, given a set of trigger signals.

**.xcf file.** An .xcf file is a scan chain configuration file generated by ispVM for programming devices in a JTAG daisy chain. The .xcf file contains information about each device, the data files targeted, and the operations to be performed.

---

**Recommended Reference Materials**

The following reference materials are recommended to supplement this tutorial:

- *LatticeECP & LatticeEC Family Handbook*
- *LatticeXP Family Handbook*
- LatticeECP/EC Family Data Sheet
- LatticeXP Family Data Sheet
- Technical Note #1054, “Lattice ispTRACY Usage Guide”
- *LatticeEC Standard Evaluation Board (Revision B) User’s Guide*
- *LatticeEC Advanced Evaluation Board (Revision C) User’s Guide*
- *LatticeXP Standard Evaluation Board User’s Guide*
- *LatticeXP Advanced Evaluation Board User’s Guide*
- Lattice ispLEVER online Help for IPexpress, the ispTRACY Core Linker, and the ispTRACY Logic Analyzer