Probe Visualizer User’s Guide and Tutorial

ProtoLink Prototype Verification Platform
(2013.03)

Synopsys, Inc.

Printing

Printed on February 27, 2013.

Version

This manual supports the ProtoLink Prototype Verification Platform 2013.01 version. You should use the documentation from the version of the installed software you are currently using.

Copyright Notice and Proprietary Information

Copyright (c) 2012-2013 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at http://www.synopsys.com/Company/Pages/Trademarks.aspx.

All other product or company names may be trademarks of their respective owners.

Synopsys, Inc.
700 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

About This Book 1

Purpose 1
Audience 1
Book Organization 2
Conventions Used in This Book 3
Abbreviations 3
Glossary 4
Related Publications 5

Introduction 7

Overview 7
Architecture 8

User Interface 11

Common User Interface Features 11
Window Banner 11
Pull-down Menus 11
Mnemonic Keys 11
Bind Keys 12
On-line Help 12
Toolbars 12
Mouse Operation 12
Right Mouse Button Menus 12
ProtoSet User Interface 13
ProtoRun User Interface 14

Setup 15

Overview 15
Import 17
Configure 22
Hardware Selection 22
RTL Instrument 32
Converge 34

Setup for Post-Partition Flow 37

Overview....37
Import, Configure, RTL Instrument....39
plnproc....41
Converge, ProtoRun....42

Runtime 43

Hardware Connection....43
Software Setup....44
Setup Events and Triggers....52
Collect Samples....55
Collect Samples for Express Mode....55
Take Snapshot....58
Collect Samples for Full Visibility Flows....60
FV Dump with ProtoLink Controlled Clock....61
FV Dump with User Controlled Clock....64
Design Control....67
Probe ECO....69
Revision Management....72

Application Tutorials 73

Setup Flow....73
Create a New Project....73
Import Design....75
Configure....76
Instrument....78
Synthesize the Instrumented Design....78
Place and Route the Design....79
Converge....79
Runtime Flow....80
Hardware Preparation....80
Download FPGA Image....80
Open Runtime Database....80
Create a Simple Trigger....82
Activate System LA and Capture Waveforms....83
Add New Probes with SLA Probe ECO....84

About This Book

Purpose

This book is designed to allow you to quickly become proficient in the Probe Visualizer application of the ProtoLink Prototype Verification Platform. You should already be familiar with the ProtoLink platform before beginning this book.

The manual should be read from beginning to end, although you may skip any sections with which you are already familiar.

Audience

The audience for this manual includes engineers who are familiar with languages and tools used in the process of SoC design and verification utilizing FPGA devices and prototype boards.

This document assumes that you have a basic knowledge of the platform on which your version of the ProtoLink software runs: UNIX or Linux, and that you are knowledgeable in Verilog or VHDL, simulation software, digital logic design, FPGA-based prototyping, and hardware debug.

Book Organization

This *Probe Visualizer User's Guide and Tutorial* is organized as follows:

- *About This Book* provides an introduction to this book and explains how to use it.
- *Introduction* provides an overview of the Probe Visualizer product and introduces its capabilities and methodology.
- *User Interface* provides details regarding the interface, including toolbars, icons, and commands.
- *Setup* provides details regarding setting up the Probe Visualizer product.
- *Runtime* provides details regarding running the Probe Visualizer product.
- *Application Tutorials* gives step-by-step instructions on using the Probe Visualizer product.

Conventions Used in This Book

The following conventions are used in this book:

- *Italics font* is used for emphasis, book titles, section names, design names, file path, and file names within paragraphs.
- **Bold** is used to emphasize text, highlight titles, menu items, and other Synopsys terms.
- `Courier` type is used for program listings. It is also used for test messages that the Verdi system displays on the screen.
- **NOTE** describes important information, warnings, or unique commands.
- **Menu -> Command** identifies the path used to select a menu command.
- Left-click or Click means click the left mouse button on the indicated item.
- Middle-click means click the middle mouse button on the indicated item.
- Right-click means click the right mouse button on the indicated item.
- Double-click means click twice consecutively with the left mouse button.
- Shift-left-click means press and hold the <Shift> key then click the left mouse button on the indicated item.
- Drag-left means press and hold the left mouse button, then move the pointer to the destination and release the button.
- Drag means press and hold the middle mouse button on the indicated item then move and drop the item to the other window.

Abbreviations

- ECO - Engineering Change Order
- LA - Logic Analyzer
- TDM - Time Division Multiplexing
- SoC - System on a Chip

Glossary

This section describes some of the terms that are used throughout this document.

J-connector

The J connector is a 120-pin high speed interface connector from Samtec. It is generally used in most pre-fabricated FPGA boards. The part number of the J connector is Samtec QTH-060-01-L-D-A.

Related Publications

- *Installation and System Administration Guide* - provides information on installing the ProtoLink platform.
- *ProtoLink Command Reference Manual* - gives detailed information on the ProtoLink command set.
- *ProtoLink Tcl Reference* - gives detailed information on the ProtoLink Tcl command set.
- *Verdi User's Guide and Tutorial* - gives detailed information on using the Verdi™ Automated Debug System.
- *Siloti User's Guide and Tutorial* - gives detailed information on using the Siloti™ Visibility Automation System.
- *Release Notes* - for current information about the latest software version, see the Release Notes shipped with the product and the installation files in the distribution directories.

Introduction

Overview

The ProtoLink Prototyping Verification Platform improves the efficiency of verifying SoCs. The ProtoLink Probe Visualizer product is easy to use and accelerates the debug process with its unique capabilities. When connected, it provides rich real-time visibility into prototype boards. The waveforms of the required signals can be dumped without using a logic analyzer. In addition, the Probe Visualizer's debug environment is seamlessly integrated with the Verdi™ Automated Debug System from Synopsys. The debug process is then as easy as doing RTL verification.

The Probe Visualizer product enhances the visibility and debugging in prototype boards in the following aspects:

- Greatly improves debug productivity by integrating the Verdi system
- Reduces iteration time of adding new probes from possible hours down to minutes
- Increases the number of probes from 10s of signals to 1000s
- Provides millions of cycles captured to help locate random failures
- Enables debugging at RTL level
- Enables debugging across multiple FPGAs as if they were a single FPGA

Using the Probe Visualizer product for a pre-partitioned design on prototype boards makes it easier and faster to enable the visibility and debug for existing designs on prototype boards.

Architecture

The ProtoLink Probe Visualizer product consists of the following elements (refer to the *ProtoLink Probe Visualizer Architecture* and *Probe Visualizer and Prototype Board Connection* figures):

- The ProtoLink Interface Card: the card is designed to be connected to the workstation (the runtime host) at one end and connected to the prototype board (loaded with the design and the configured ProtoLink Probe Visualizer IP) on the other end. When connected, the Probe Visualizer runtime commands can be executed from the host workstation to probe desired signals and quickly dump an FSDB file for analysis.
- The Probe Visualizer software, which provides:
  - The setup command utility (*protoset*): to add probes, and then to generate the instrumented design and the Probe Visualizer IP for downstream tools.
  - The converge command utility (*converge*): to generate the runtime database for protorun.
  - The runtime command utility (*protorun*): to arm the prototype board with built in Probe Visualizer IP, set triggers and save dumped waveforms.
- Probe bus: the probe bus is the bus connecting the Probe Visualizer IP to the ProtoLink Interface Card.

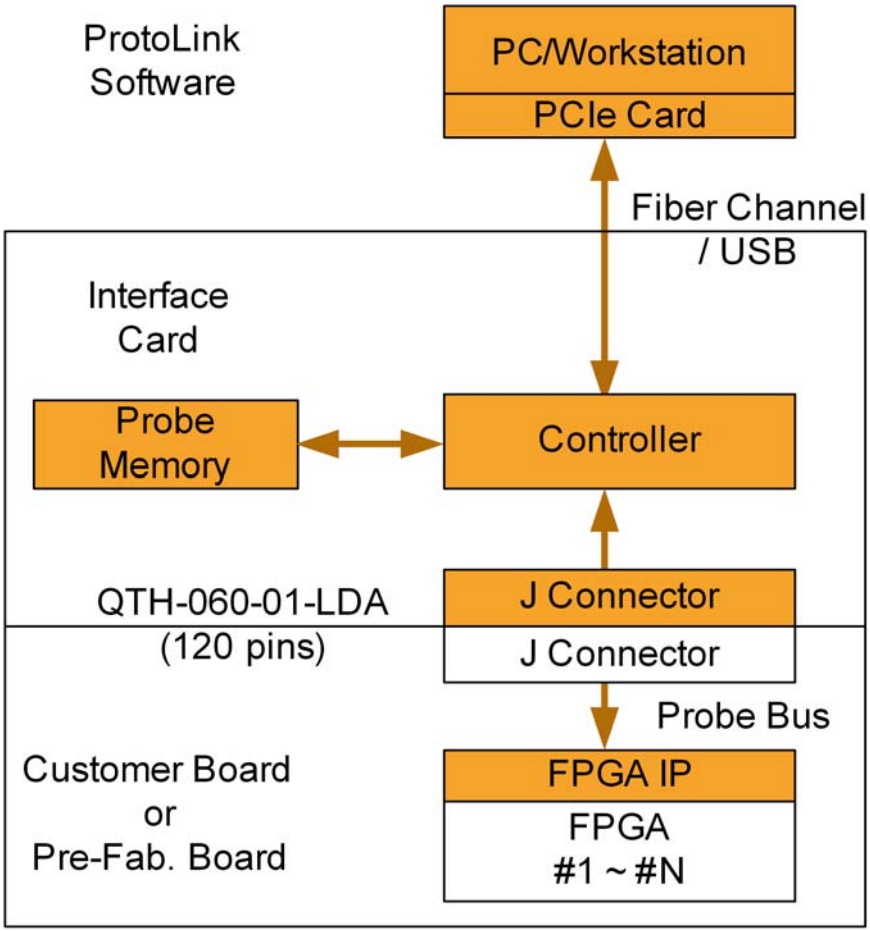

Figure 1: ProtoLink Probe Visualizer Architecture

The Probe Visualizer architecture supports:

- 125MHz system clock
  - Supports synchronous and asynchronous sampling
- Up to 4G byte probe memory
  - Holds up to 134M cycles (~ 1 sec)
- Up to 10K+ probes per FPGA per probe bus
  - 16 TDMs: can all be used in one run depending on user clock
  - 8 groups: only one group can be used in one run
- Up to 4 probe buses can be used for multiple FPGAs
  - Up to 6 probe buses if using 100MHz system clock

To complete the Probe Visualizer hardware set up, the ProtoLink Interface Card needs to connect to a runtime host workstation at one end and to the prototype board with a J connector or Mictor connector at the other end (as shown in the following figure).

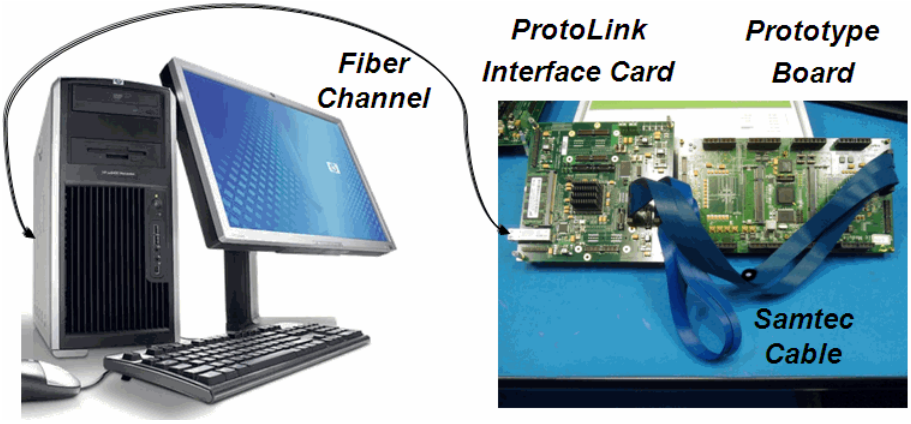

Figure 2: Probe Visualizer and Prototype Board Connection

The ProtoLink Interface Card uses two methods to connect to the workstation. The first method is to connect to the custom PCIe card inside the workstation through the fiber channel port. The second method is to connect to the USB port of the workstation directly. While the second method is easier to connect, the first method provides better data upload performance. Only one method of connectivity can be selected during a debug capture run. If both methods are connected at the same time, the default connection will be the fiber channel port; however, this can be overridden by manually selecting USB as the connection.

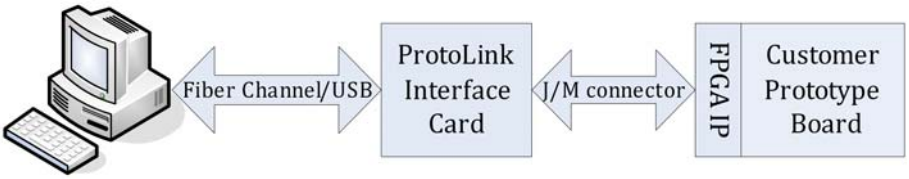

Figure 3: ProtoLink Interface Card Connections

User Interface

The ProtoLink platform has a large number of commands, including many that are invoked through mouse clicks or drags rather than selecting from pull-down menus at the top of each window. Read this chapter to become familiar with the interface conventions of the ProtoLink platform before proceeding further.

Common User Interface Features

The features described below are common to *ProtoRun* and *ProtoSet* windows.

Window Banner

The banner at the top of each window identifies the application and window number (such as <*ProtoRun:2*>).

Pull-down Menus

A pull-down menu bar is located just below the window banner. Each menu item contains several commands that display when the menu is selected. The ProtoLink platform supports the tear-off menu convention, which allows frequently used menus to be pulled aside and left open.

Mnemonic Keys

The window menus support **Meta** key invocation using mnemonics. The mnemonic for each item is indicated by an underline. For example, to display the **File -> Open** menu (meta -fo), press and hold the <Meta> key on the keyboard (the diamond key/<Alt> key on Sun keyboards or the <Alt> key on Windows' keyboards) and press the "f" key, then release the <Meta> key and press the letter "o" key.

Bind Keys

A command can be bound to either a keystroke or a mouse button. After the bind keys are defined, commands can be easily invoked with a keystroke or mouse click.

On-line Help

All application windows provide on-line help, which can be accessed through the **Help** menu on the main toolbar.

Toolbars

A row of icons appears beneath the pull-down menu bar. These icons provide access to frequently used commands for the current window.

Mouse Operation

The mouse is most often used to select objects by clicking the left mouse button. A range of objects can be selected by dragging with the left mouse button over the objects or by using the <Shift> key along with clicking-left. To add or remove individual objects to or from the selection, use the <Ctrl> key along with left-click.

The ProtoLink platform also makes use of drag-and-drop to move information from one window to another. A drag-and-drop is normally performed by pressing the middle mouse button to select the object, holding the button as the mouse is moved to a new location, and then releasing the button to “drop” the object into a new location.

Right Mouse Button Menus

Right-clicking an object will display a menu with commands appropriate for that object type.

ProtoSet User Interface

An example *ProtoSet* window is shown below:

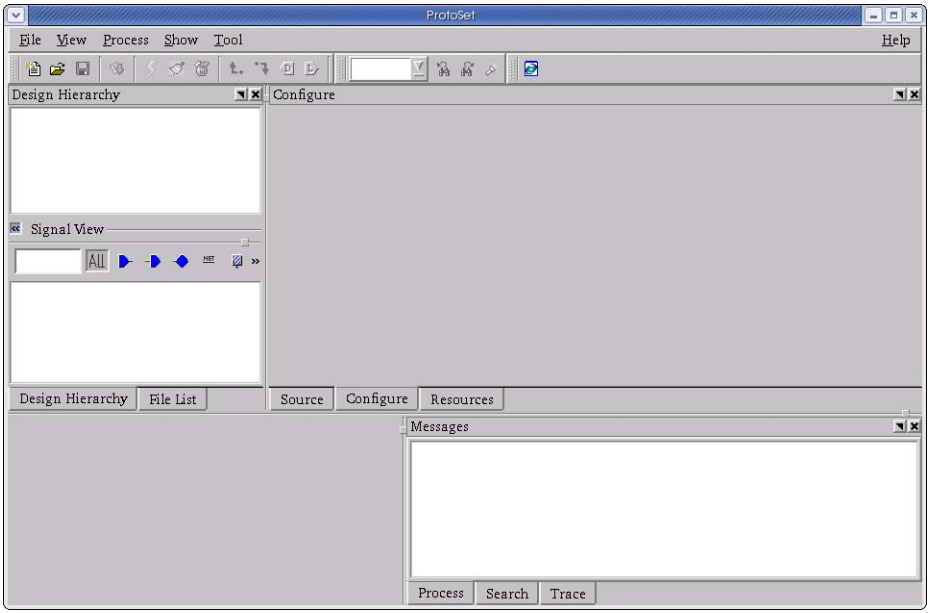

Figure 1: ProtoSet Example WIndow

ProtoRun User Interface

An example *ProtoRun* window is shown below:

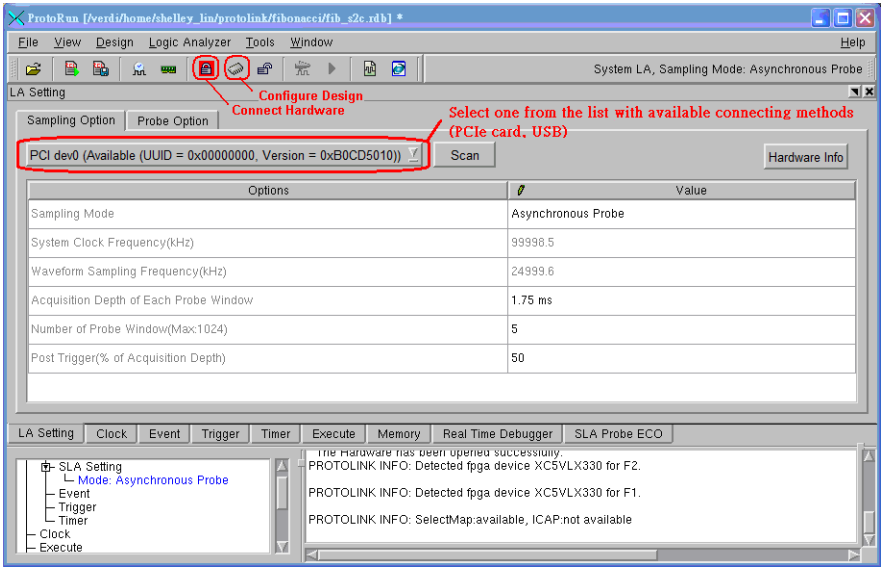

Figure 2: ProtoRun Example WIndow

Setup

Overview

The flow chart for the Probe Visualizer setup for pre-partitioned mode is shown in the following figure.

The orange-colored box indicates the Probe Visualizer setup steps that are performed within the *ProtoSet* window. The yellow-colored box indicates the Probe Visualizer Converge steps to generate the runtime database. The light blue-colored box indicates the Probe Visualizer runtime steps that are performed within the *ProtoRun* window.

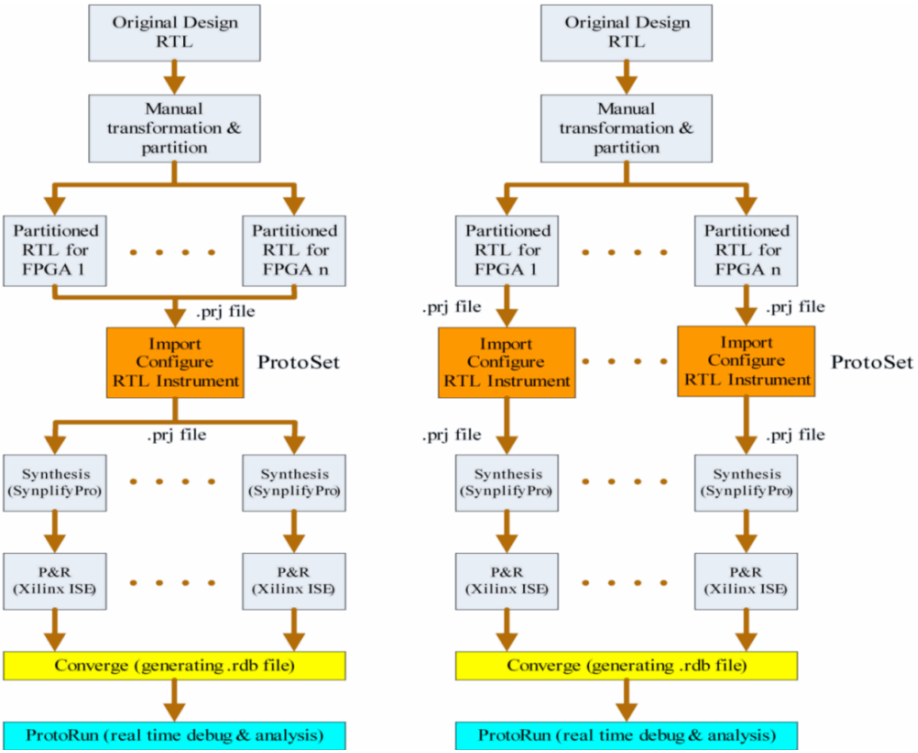

Figure 1: Probe Visualizer Setup Flow

The traditional flow for SoC prototyping starts with an RTL design that is partitioned and mapped to multiple FPGAs on the board. Based on the

partitioned design, the corresponding project file (.prj or .xst file) will be prepared with the RTL source file list and the synthesis constraint information. Tools such as Synplify Pro, Xilinx XST, or Xilinx Vivado can be used for synthesis to generate an EDIF file that will then be implemented for Place & Route in the Xilinx Integrated Software Environment (ISE) tools. During this step, some black boxes in the design will have to be replaced by the corresponding EDIF file, a common practice for handling IP.

With the Probe Visualizer product, an extra process for setting up (using the `protoset -Probe` command) is needed before running synthesis. The setup includes the following three steps: **Import**, **Configure**, and **RTL Instrument**.

As shown in the *Probe Visualizer Setup Flow* figure, users can run one *ProtoSet* process for one FPGA (one synplify project file) at a time and run multiple *ProtoSet* processes for multiple FPGAs. When design modifications are made to the RTL that only affect some but not all FPGAs, rerunning the *ProtoSet* process is only needed for those modified FPGAs.

Users can also provide multiple project files to the *ProtoSet* window to do setup for all FPGAs together. When design modifications are made to RTL that only affect some but not all FPGAs, rerunning the *ProtoSet*.process will check and highlight in red the files with a timestamp that is later than the imported timestamp at file view. The import process will be forced to rerun for the modified files only.

Either way, the synthesis and P&R flow are always run per FPGA and users need to run Converge to generate one runtime database for ProtoRun.

Import

The *ProtoSet* window will be opened when the following command is executed:

```
protoset -Probe
```

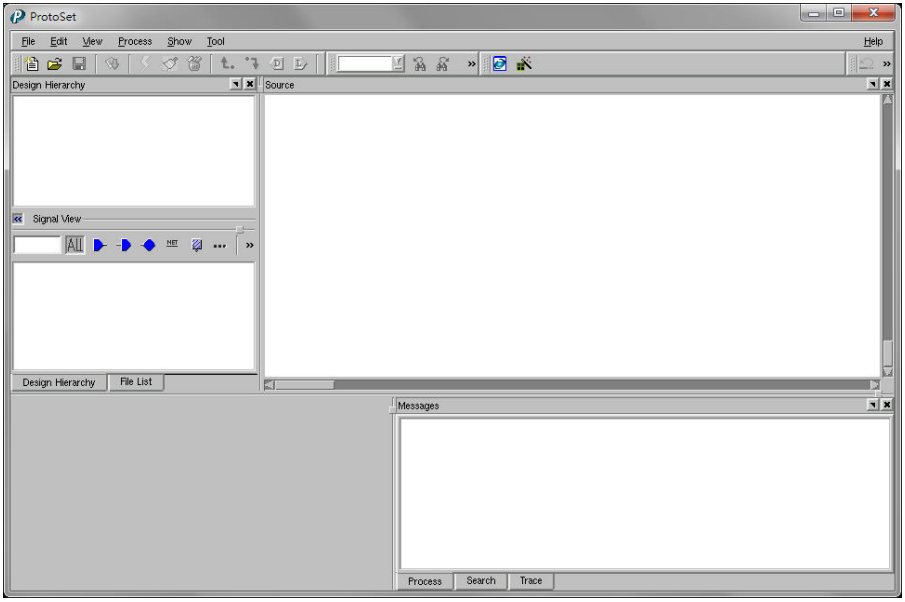

Figure 2: ProtoSet Window

Create a new ProtoSet project with the **File -> New Project** command or the **New Project** toolbar icon or open an existing ProtoSet project with the **File -> Open Project** command or the **Open** toolbar icon. When creating a new project, ProtoSet will create the necessary links and copy the necessary database files from the product installation location to the new project's directory.

Import is the first step in the ProtoSet process. This step is for importing the design from the project file (.prj file) originally prepared for Synplify Pro or from pre-compiled design libraries (pre-compiled from the Synplify Pro, Xilinx XST or Quartus project files).

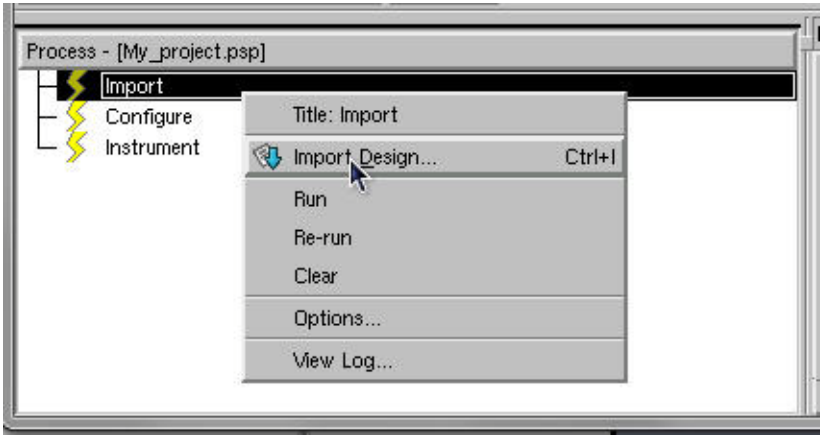

Figure 3: ProtoSet Process - Import Design Step

To import the design in the Import step, select the **Import Design** command from the **Process** pull-down menu, the right mouse button command menu, or select the **Import Design** toolbar command to open the *Import Design* form.

The design can be imported from a synthesis project file or a pre-compiled library.

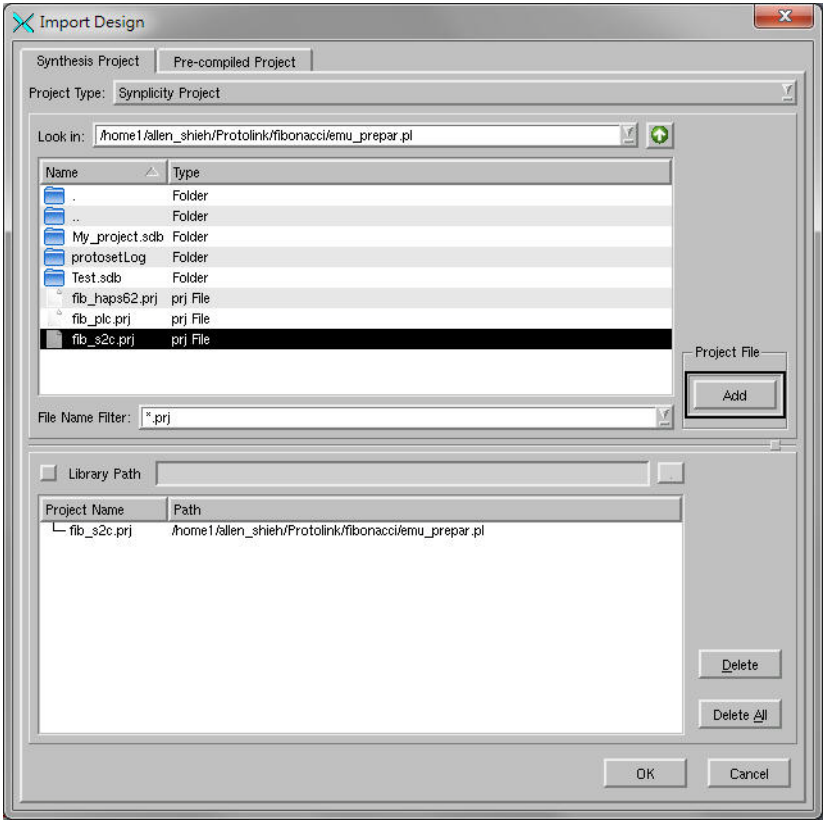

Figure 4: Import Design from Project

For importing the design from the Synplify Pro project file (.prj file), select the **From Project** tab in the *Import Design* form. Specify all the related project files and click the **OK** button to import the design.

To import the design from a pre-compiled library, you will need to pre-compile the Xilinx XST project file, the Synplify Pro project file, or the Quartus project file to .ppl files in advance. The steps are:

1. Generate Makefile files from Synplify Pro project files (.prj file), Xilinx XST project files (.xst file) or Quartus project files (.qsf file) with the Probe Visualizer *plcomgen* utility. For example:

   <u>For Synplify Pro</u>:

   ```
   plcomgen -prj <prj_file_1> -o <makefile_1>
   plcomgen -prj <prj_file_2> -o <makefile_2>
   …
   ```

   <u>For Xilinx XST:</u>

```
plcomgen -f xst -prj <prj_file_1> -o <makefile_1>
plcomgen -f xst -prj <prj_file_2> -o <makefile_2>
…
```

For Quartus:

```
plcomgen -f qua -prj <prj_file_1> -o <makefile_1>
plcomgen -f qua -prj <prj_file_2> -o <makefile_2>
...
```

2. Run make to compile the design to generate *.ppl files. For example:

```
make -f <makefile_1>
make -f <makefile_2>
…
```

Then import the pre-compiled design libraries via the **Pre-compiled Project** tab in the *Import Design* form. Specify all related *ppl library files and click the **OK** button to import the design (see the *Pre-compiled Project* figure).

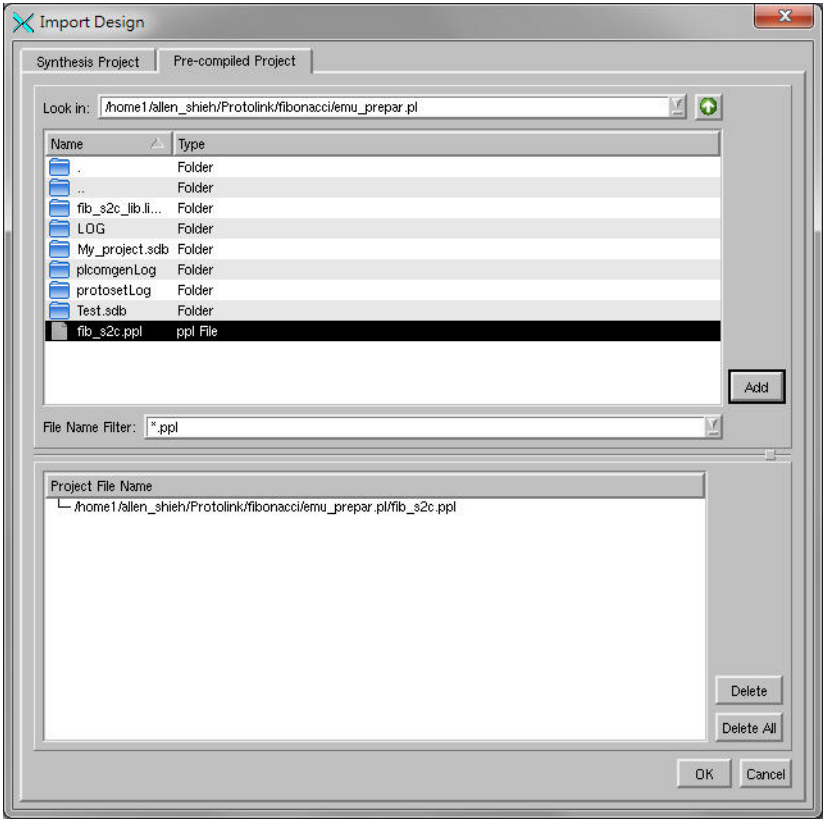

Figure 5: Pre-compiled Project

After successfully importing a design, the Import step is done. The status is changed from (initial) to (successful). This means the Configure step can be initiated. The other statuses include (failed) and (running).

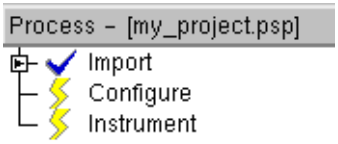

Figure 6: ProtoSet Process Status

The compiled database from the Import step can be used for both the Probe Visualizer product and the Verdi system so the Verdi system can be directly invoked for debug after collecting the waveforms. There is no need to recompile the design for the Verdi system.

Configure

Configure is the next step. Provide the information regarding the target prototype board and the current chip assignment (RTL/FPGA) and then configure the probe resources and specify the signals for probing.

Hardware Selection

The first step of configuration is to select the hardware platform or the prototype board. A few hardware platforms are already included with the tool. Alternatively, a customized hardware platform can be created for prototyping.

The Probe Visualizer product can work with any prototype platform if the board file can be found in the library. For example, the *Platform Selection* figure shows the selection of HAPS54 as the hardware platform in the **Hardware** section.

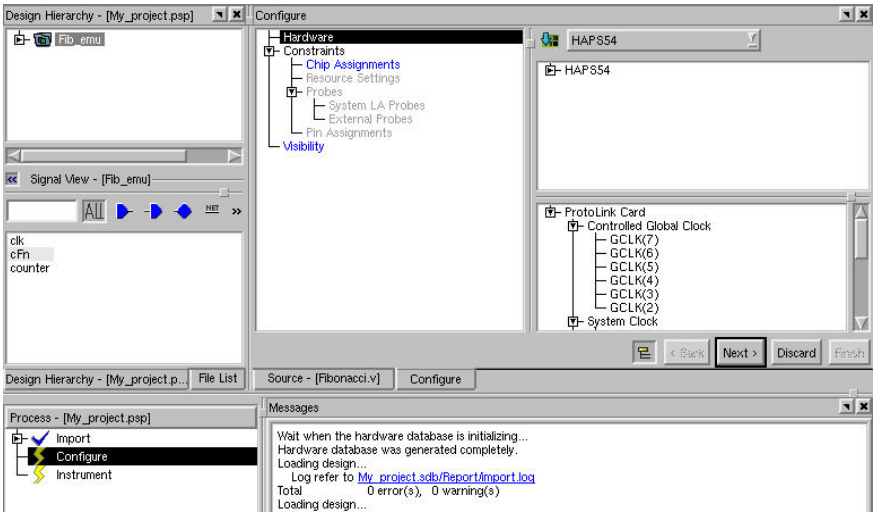

Figure 7: Platform Selection

If the platform has a different configuration from the default platform, click the **Import Platform** icon in the *Platform Selection* figure to import another configured platform file.

To modify the pre-defined platform to meet your specific need, you can click the **Platform Configuration Wizard** icon on the toolbar to open the *Platform Configuration Wizard* form to create a customized platform based on the pre-defined platform.

In the *Platform Configuration Wizard* form (see the *Platform Configuration Wizard* figure), select a pre-defined platform from the **Source Platform** field and specify a name for the new platform in the **New Platform Name** field. The **Platform Export Path** field is used to export the platform file to the specified directory. Click the **Next** button to continue, and the **Platform Hierarchy** section will be shown in the form where the speed grade can be changed if the default value is not good.

Click the **Next** button again and the **Connector Assignments** section will be shown. In the **Connector Assignments** section, you can modify the connection between the ProtoLink J-connectors and your FPGA chips. Specify the required probe bus width in the **Max. Probe Bus Width** column and indicate whether to insert a snapshot bus in the **Snapshot Bus** column. The snapshot bus is used for capturing the snapshot of registers. The **Snapshot Bus** flag is *on* if a snapshot bus is defined for the selected platform. However, for each chip, only one **Snapshot Bus** flag is allowed to be turned on. Click the **Next** button to continue and the **Clock Configurations** section will be shown in the form. Modify the clock settings if required. Click the **Next** button again to go to the **Conclusion** section and then click the **Finish** button to complete the creation of the customized platform. The newly created platform can be found in the specified export path and can be imported via the **Import Platform** icon.

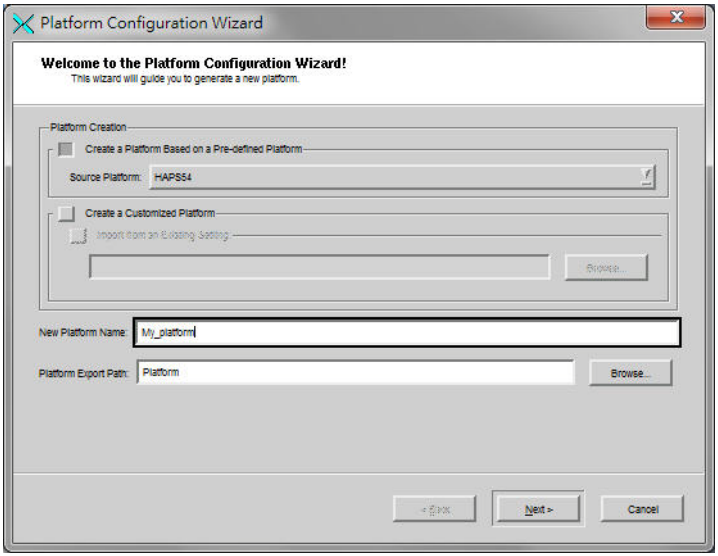

Figure 8: Platform Configuration Wizard

In the Chip Assignment step (see the *Chip Assignment* figure), select the RTL top and assign to each FPGA.The design instance can be dragged from the **Design Hierarchy** pane and dropped into the **Instance** field. Then specify the mapping

FPGA by double-clicking the **Assign To** field to bring up the name list of all FPGAs contained in the platform. In the *Chip Assignment* figure, the uB is specified as the FPGA to hold the “Fib_emu” design.

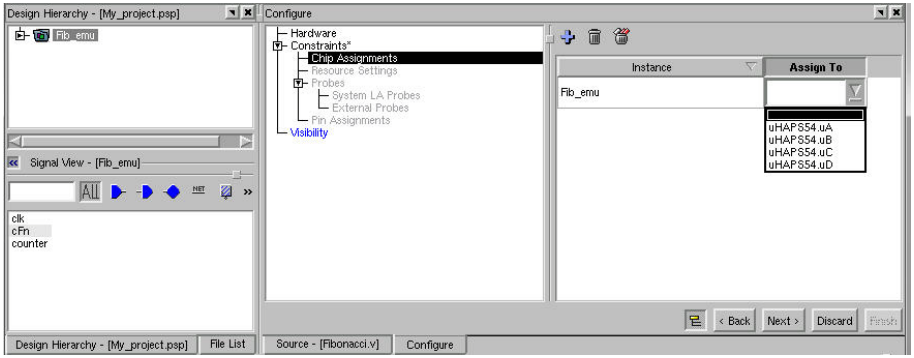

Figure 9: Chip Assignment

The probe resources are based on three factors: Probe Bus width, Probe Group, and Probe TDM (Time Division Multiplexing). The probe bus information can be browsed in the **Hardware** page. For example, as shown in the *Probe Bus* figure, there is a 64-bit probe bus available in HAPS54 uB FPGA. The Probe Visualizer product also provides 8 probe groups and 16 probe TDMs so that the total maximum signals that can be probed are 8K (64 x 16 x 8 = 8K). This is, however, limited by the routing resources available on the FPGA. The actual available probes per FPGA are therefore design dependent.

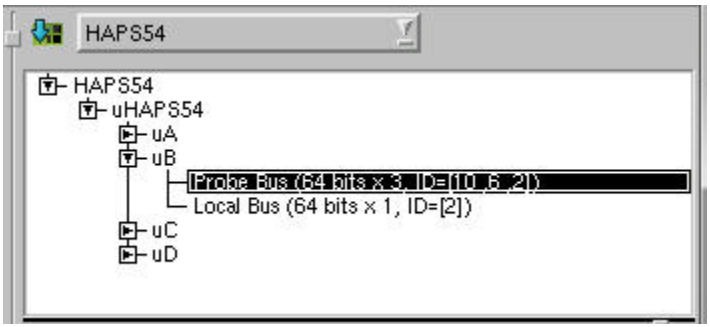

Figure 10: Probe Bus

The three probe resource factors are independent and can be used in any combination. The probe bus width is determined by the board file. The ProtoLink Probe Visualizer product supports a 32-bit probe bus which requires 8 FPGA I/O pins or a 64-bit probe bus which requires 16 FPGA I/O pins. The ProtoLink Probe Visualizer product also needs at least two additional control pins for both 32-bit and 64-bit probe buses. Contact your local AE to request the *Design User*

Prototype Board to Connect with ProtoLink Interface Card (Probe Capability Only) application note for more details about the ProtoLink Probe Visualizer hardware.

The probe group is configured to select one group per run. Therefore, two runs are needed to collect the results of two probe groups. On a fast running prototype board, one run only takes seconds so multiple runs can be quickly accomplished and waveforms collected from multiple probe groups. Each group can be configured manually or a probe list file provided.

The probe TDM can be configured automatically by providing just the required signals because all signals will be probed during the same run. TDM is used to multiplex multiple samples (up to 16) in a single clock cycle. As a result, the user clock that can be sampled will need to be slowed down accordingly.

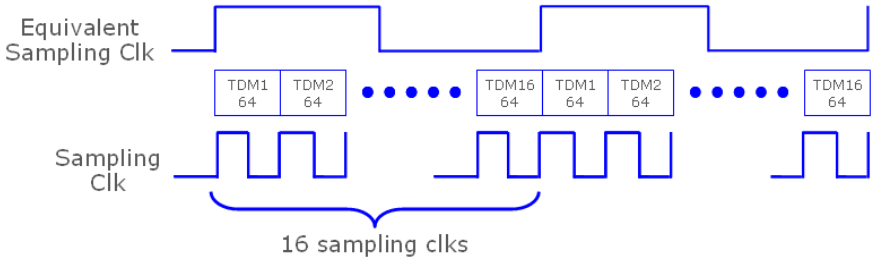

Figure 11: Probe TDM

Up to 125MHz sampling clock is used to do either asynchronous or synchronous sampling of the signals.

Synchronous sampling employs a user clock as the reference clock to sample the signals. The advantage of using synchronous sampling is that the user clock can run faster than the asynchronous sampling. Synchronous sampling only catches the signal changes after the clock edges, so it is beneficial for a fully synchronous design. For example, if the positive user clock edge triggers the signal changes, the changes can be sampled by the negative user clock edge.

The advantage of using asynchronous sampling is that possible signal glitch problems can be detected because the signals are sampled in between the edges of the user clock. Asynchronous sampling can also be used to sample the clock if needed. On the other hand, the user clock would have to be slowed down to insure that an accurate waveform can be captured or shown. Based on the Nyquist Theorem, at least two samples are required per user clock and it is suggested to take three samples per user clock. In other words, the fastest user clock should be around 33.3MHz (100/3) for a 100MHz sampling clock.

A reference guide to choose between an Asynchronous or Synchronous probe is:

- Choose Synchronous probing scheme when:
  - There is a reference clock and the frequency of the system clock can be evenly divided (resulting in a whole number) by the reference clock. For example, if the reference clock is 50MHz, then a 100MHz system clock can be divided by the 50MHz reference clock.
- Choose Asynchronous probing scheme when:
  - The user clock is less than or equal to “system clock / 3” to be able to view the waveform with quality. For example, if the system clock is 100MHz, then the user clock is 33.3MHz or slower.

Based on a 125MHz sampling clock, 64-bit probe bus, and the 4G byte probe memory provided by the ProtoLink Interface Card, the effects of different probe TDM numbers and sampling schemes (synchronous/asynchronous) can be found in the following table.

TDM	Synch Probe (Max User Clock MHz)	Asynch Probe (Max User Clock MHz)	No. of Probes (1 Probe bus per FPGA)	No. of Cycles Captured (Sync Probe)	No. of Cycles Captured (Asynch Probe)	Acquisition Depth (ms)
1	125.00	41.67	64	134,217,728	44,739,242	1,073.74
2	62.50	20.83	128	67,108,864	22,369,621	1,073.74
4	31.25	10.42	256	33,554,432	11,184,810	1,073.74
8	15.63	5.21	512	16,777,216	5,592,405	1,073.74
16	7.81	2.60	1024	8,388,608	2,796,202	1,073.74

NOTE: A maximum of 4 probe buses are supported for 125 MHz sampling clock (6 probe buses for 100 MHz sampling clock).

Assume the case with the following conditions:

- The sampling type is synchronous probe.
- The sampling clock is 125 MHz.
- The TDM is 4.
- 4G Bytes of storage memory is installed on the protolink card.

In this case, the maximum number of probe buses is 4 (due to 125 MHz sampling clock) and 1G Byte is available for each probe bus.

The number of probes available in one cycle
= 4 (TDM) * 64 (no. of probes per probe bus) = 256

The number of cycles captured = 1 * 2**30 * 8 / 256 = 33,544,432

The acquisition depth is independent of the TDM value and can be calculated as follows:

Acquisition Depth (in seconds)

= 1 Gbytes/probe bus / 64 bits/probe bus / 125 MHz
= ((1 * 2**30 * 8)/64)/(125 * 10**6)
= 1073.74 (ms)

Select the **Resource Settings** step (see the *Resource Settings* figure) to configure the Probe Visualizer resource for each FPGA. Set the reference clock to the first FPGA if synchronous probe is selected.

When the reference clock is set, the **Base System Clock Frequency (kHz)**, the **Clock Multiplier**, the **System Clock Frequency (kHz)**, the **Reference Clock Frequency (kHz)**, and the **Recommended Max. TDM** fields will be as shown in the *Resource Settings* figure. Specify the frequency for the reference clock in the **Reference Clock Frequency (kHz)** field and then Probe Visualizer will calculate the value and show the recommended maximum TDM number in the **Recommended Max. TDM** fields. If the specified reference clock frequency value is not valid, the field will be highlighted in red. Refer to the *Messages* pane for the error message if this occurs.

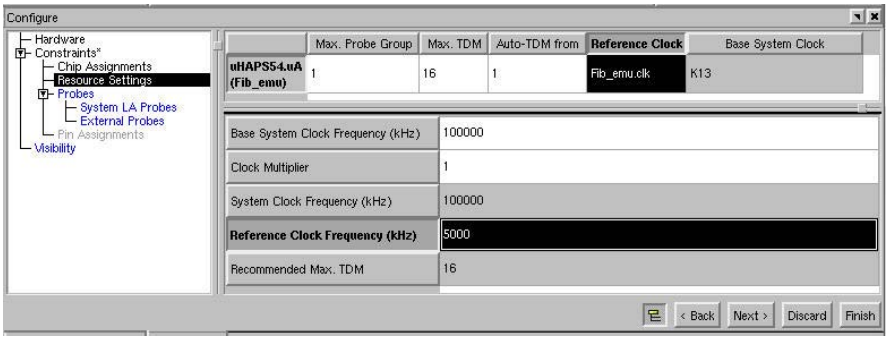

Figure 12: Resource Settings

For some platforms that do not specifically identify the system clock pins, the desired system clock can be manually specified for Probe Visualizer IP for each FPGA. Contact your local AE to request the *Design User Prototype Board to Connect with ProtoLink Interface Card (Probe Capability Only)* application note and refer to the *Clock Example* section for more details.

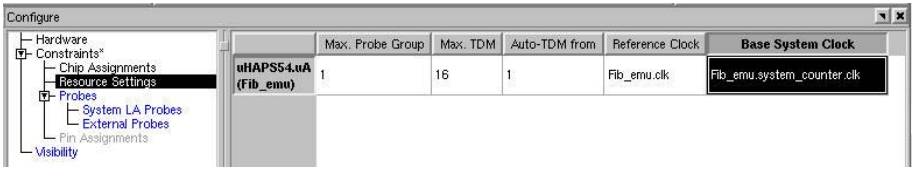

Figure 13: Set System Clock for the F1 FPGA

The *Add Signals for Probing and Specify Probe Bus, Group, and TDM* figure shows the **System LA Probes** section. This section is for adding signals for probing and for configuring each probe signal's probe bus, probe group, and TDM slot number (TDM Auto implies the tool will automatically assign a slot). A specific probe bus can be specified in the **Probe Bus** field if there are multiple probe buses available on the related FPGA. The signals can be dragged from the signal view pane of the *ProtoSet* window or from any window of the Verdi system (such as source code, waveform, or schematic) directly and dropped in the **Signal Name** field. The **Import from File** and **Export to File** icons can be used to import the probe list from file or export the current probe list to file.

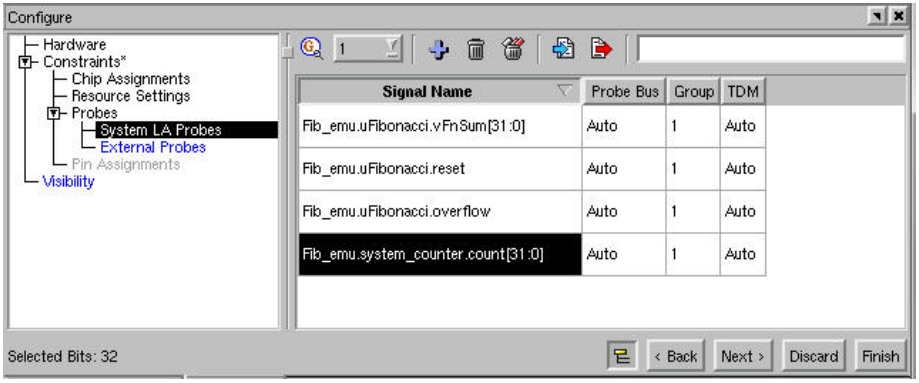

Figure 14: Add Signals for Probing and Specify Probe Bus, Group, and TDM

Turn on the **View -> Resources** toggle command to add the **Resources** tab for checking the available probe resources as shown in the following figure.

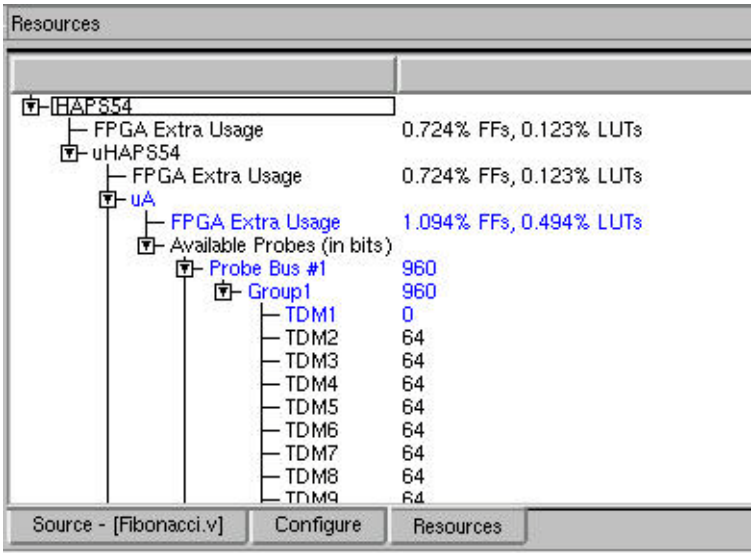

Figure 15: Available Probe Resources

The Probe Visualizer product also supports specifying the external probe (to connect to an external Logic Analyzer) and the pin assignment. Select the **External Probes** and **Pin Assignments** categories to configure respectively.

The **Force/Release** section is available for configuration as shown in the *Force/Release Section* figure if the snapshot bus is available in the selected platform. To get more details about the snapshot bus, contact your local AE to request the *Design User Prototype Board to Connect with ProtoLink Interface Card (Probe Capability Only)* application note.

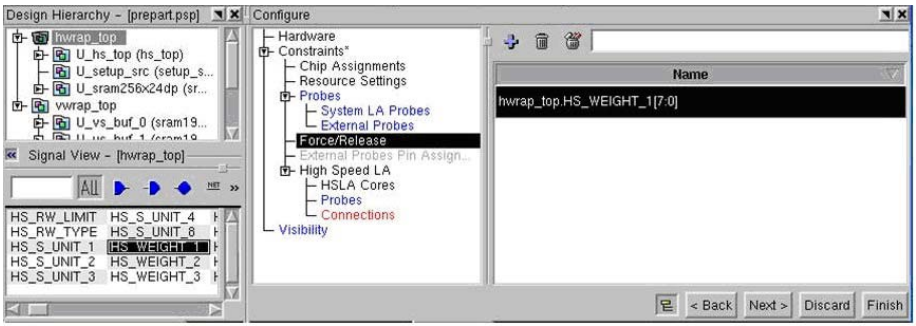

Figure 16: Force/Release Section

This function is very useful when users want to control specific signals to have specific values during runtime. Users will need to drag and drop signals from the *Signal View* window to the **Force/Release** list window or enter the signal name

directly in the **Force/Release** signal name field. Only those signals shown in the **Force/Release** list window can be forced and released during runtime.

Select the Visibility step to edit an essential signal list. The list will be preserved from being optimized by the synthesizer and thus the signals can be probed dynamically using Probe ECO. Also, the list is required for the Full Visibility flow. Refer to the *Take Snapshot* section for more details.

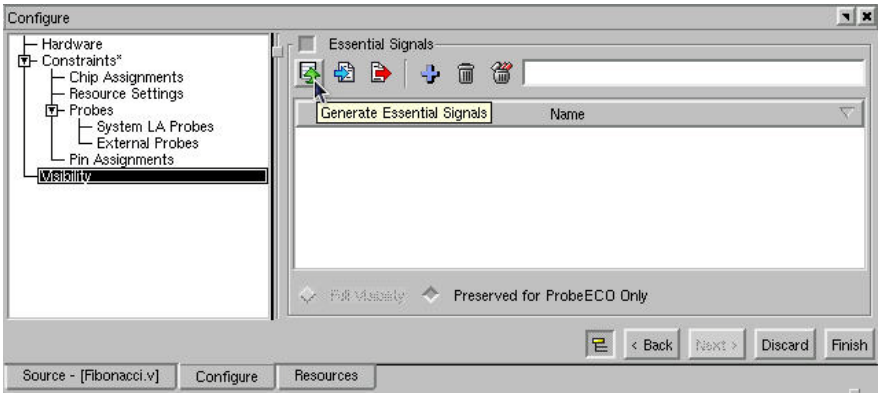

Figure 17: Visibility Section

The ideal source of the essential signal list is from the Siloti system. The Siloti system can automatically analyze a design to determine the essential signals and infers values for signals not captured.

In the **Visibility** section, the Siloti system (Siloti license required) can be directly invoked to generate the essential list by clicking the **Generate Essential Signals** icon. The *Generate Essential Signals* form containing the script content to execute the Siloti Essential Signal Analysis utility is invoked. Make any modifications, if required, and then click the **OK** button to proceed.

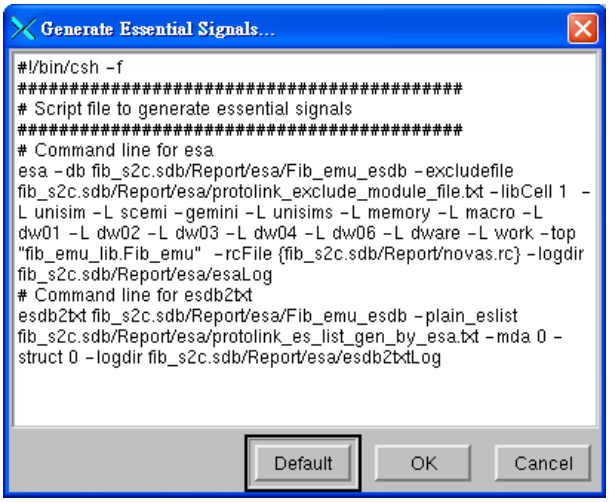

Figure 18: Generate Essential Signals Form

When done, the generated essential signals will be added back to the list in the **Visibility** section as shown in the following figure. Instead of using the **Generate Essential Signals** button, a manually compiled list can be imported from the **Import from File** icon. To save the current list to a file, select the **Export to File** icon. To remove or add essential signal to the list, select the **Delete Selected**, **Delete All**, or **New** icons respectively.

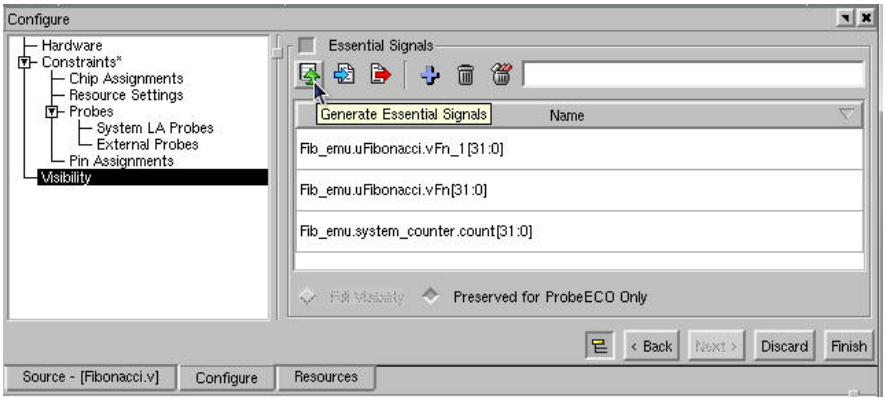

Figure 19: Visibility Section with Generated Essential Signals

To use the Full Visibility flow, the *plnproc* utility needs to be run before Converge as described in the *Take Snapshot* section. Click the **Finish** button to complete the Configure step.

RTL Instrument

In the **Process** pane, select **Instrument** and then **Run** on the right mouse button command menu to start the instrumentation.

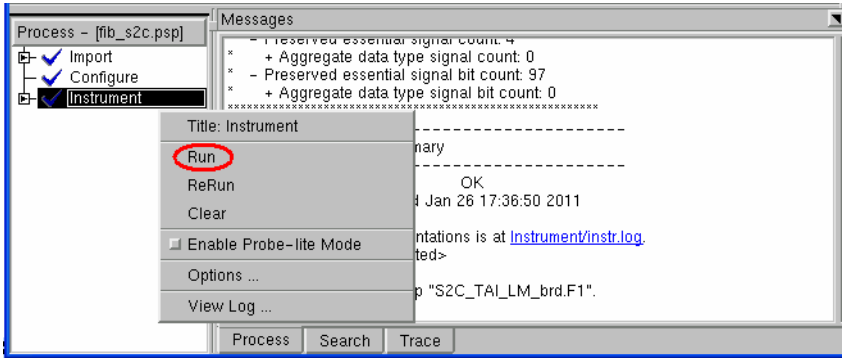

Figure 20: Start RTL Instrumentation

Based on the configuration, ProtoSet will insert the generated Probe Visualizer IP and connections to the design. The instrumented design will be saved in the `Instrument` directory by default and the modified project file is generated. Also the Probe Visualizer setup database (.sdb file) and the corresponding ProtoLink Setup Project file (.psp file) are updated.

If users want to specify a non-default instrument output directory, select the Options command from the pull down menu shown in the *Start RTL Instrumentation* figure.

After the ProtoSet setup process is completed, it's possible to go back to the original flow to run synthesis and P&R. However, for synthesis, the ProtoSet generated project file needs to be provided (located in the `<instrument output>/<chip>` directory) into Synplify Pro, Xilinx XST, or Quartus.

NOTE: For P&R (using the Xilinx ISE tool), the path of the ProtoLink IP needs to be specified with the **–sd** option of the Xilinx *ngdbuild* utility (the Translate step) as follows:

```
ngdbuild -sd <instrument output>/<chip>/EDIF …
```

For the Quartus flow, different methods are used for different synthesizers based on the flow requirements. To understand the details, follow the first page of the Application Tutorials and look into the following directory:

```
<working_dir>/demo/probe/fibonacci/project.altera_qua (for
using Quartus synthesizer and P&R)
```

```
<working_dir>/demo/probe/fibonacci/project.altera_synp (for
using Synplify Pro and Quartus P&R)
```

Alternatively, use the *plAutoRun* command utility provided by the Probe Visualizer product to automatically run synthesis, P&R, and converge for Xilinx FPGA. For example:

```
> plAutoRun -psp s2c.psp
```

For more details about the command utility, use the –h option to print out the information.

Converge

The final step to generate the runtime database (`<project_name>.rdb` directory) shown on *Probe Visualizer Setup Flow* is Converge.

The converge command utility will automatically invoke the Xilinx *xdl* command utility to generate the required .tab file. So make sure the tool path of Xilinx ISE is set properly to allow the *converge* command utility to invoke the *xdl* program successfully.

The steps to perform Converge are:

1. If a chip file does not exist yet, use the following command to generate a template chip file.

   ```
   converge -psp <project_file> -cft <output_chip_file>
   ```

   For example:

   ```
   > converge -psp fib_s2c.psp -cft chip_file.s2c
   ```

   A sample of the created chip file:

   ```
   @converge rc file Version 1.0
   [S2C_TAI_LM_brd.F1]
   SYN_DIR =
   SYN_FILES = Instrument.s2c/S2C_TAI_LM_brd.F1/./FPGA_1/
   fpga_1.edf Instrument.s2c/S2C_TAI_LM_brd.F1/EDIF/FPGA_BM.edf
   SYN_PREFIX =
   PNR_DIR = /verdi/home/user/protolink/fibonacci/emu_prepar.pl/
   Instrument.s2c/S2C_TAI_LM_brd.F1/./FPGA_1
   PNR_PREFIX =
   [S2C_TAI_LM_brd.F2]
   SYN_DIR =
   SYN_PREFIX =
   PNR_DIR =
   PNR_PREFIX =
   ```

   Basically, a chip file is used to provide the location of where synthesis and P&R results are stored.

2. Modify the chip file if required.
3. Use the following command to generate the runtime database.

   ```
   converge -psp <project_file> -cf <chip_file>
   ```

   where <chip_file> is the chip file generated from steps 1 and 2. The **-cf** argument is used here to specify the chip file as an ‘input chip file’ whereas, in step 1, the **–cft** argument is used to specify the ‘output chip file’.

For example:

```
> converge -psp fib_s2c.psp -cf chip_file.s2c
```

A runtime database will be available after this step to be used by ProtoRun.

If not all FPGAs have been converged together and each FPGA has been converged to generate an individual runtime database, an extra *rtdbmerge* step is needed to merge all the runtime database. For example:

```
> rtdbMerge –rdb 1.rdb –rdb 2.rdb –rdb 3.rdb –outrdb merged.rdb
```

Converge can be run with the selected FPGAs only by filling in the SYN/PNR results of the selected FPGAs only in the chip file. The generated DB will only have information for the selected FPGAs and only the selected FPGAs can be debugged.

After Converge is complete, a report named *<design_top>_user_net_eco_info.rpt* is placed under the *<project>.sdb/Report/* directory. This report summarizes the signals or nets that will not be available for Probe ECO. The report includes a column for the signal name, whether it is available for ECO or not, and a simple reason if the signal is not available for ECO. There is also a simple statistic report at the end of this report file. For example:

```
> cd <path>/counter.sdb/Report
> more pattern_gen_user_net_eco_info.rpt
    user_signal ECO_Capable NG_Reason
    pattern_gen.OUSIG[16] YES NONE
    pattern_gen.OUSIG[15] YES NONE
    pattern_gen.OUSIG[14] YES NONE
    ...
    pattern_gen.reset NO NotFoundInXDL
    pattern_gen.UROM0.addr[0] YES NONE
    ...
    user signal total number          : 2100
    signal number of ECO Capable      : 2093
    signal number of No CurrObj       : 0
    signal number of Not found in xdl : 7
    visibility percentage:            : 99.7%
```

Converge also has a graphical user interface mode which can be used from inside the *ProtoRun* window. To do so, invoke the *ProtoRun* window first by:

```
> protorun –probe
```

Then invoke the **Tools -> Invoke Converge_GUI** command from the *ProtoRun* window to open the Converge GUI window as shown in the *Converge GUI* figure.

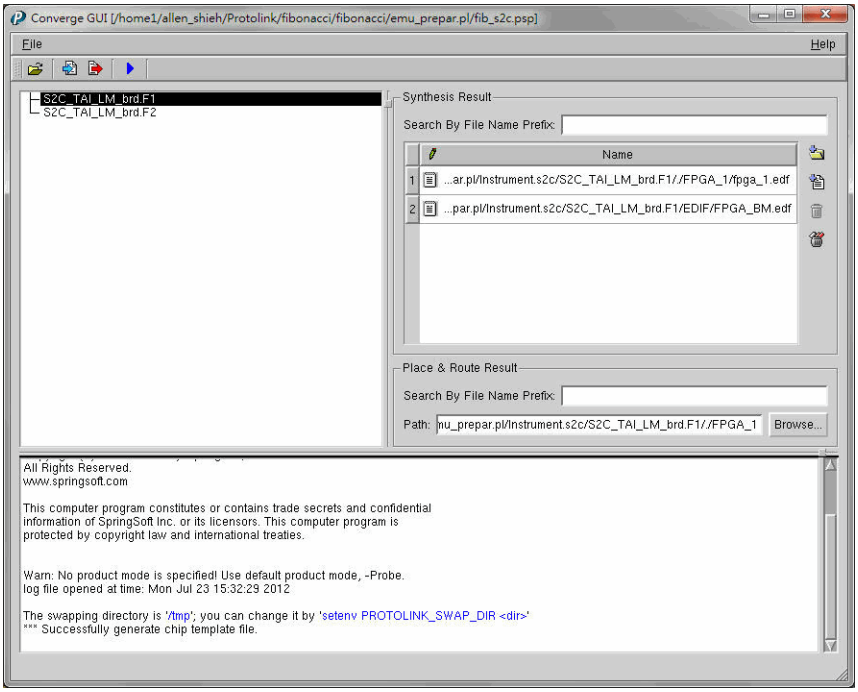

Figure 21: Converge GUI

Invoke the **File -> Open Project** as shown in the *Start RTL Instrumentation* figure to import the .psp file. Set the synthesized results file and path in the **Synthesis Result** section and the P&R results file and path in the **Place & Route Result** section as shown in the *Converge GUI* figure.

Click the **Run Converge** button to execute converge.

Setup for Post-Partition Flow

Overview

As shown in the *Probe Visualizer Setup Flow* figure in the *Setup* chapter, the normal setup flow for Probe Visualizer is assuming that per FPGA RTL is ready to be imported. However, some usage scenarios do not allow per FPGA RTL to be processed by Probe Visualizer. One scenario is that a gate-level partition tool is used so there is no per FPGA RTL available. The other scenario is that the RTL partition tool is generating a special database that can only be processed by the synthesizer but not Probe Visualizer.

To handle these scenarios, Probe Visualizer supports a special **-probeg** mode as shown in the following figure.

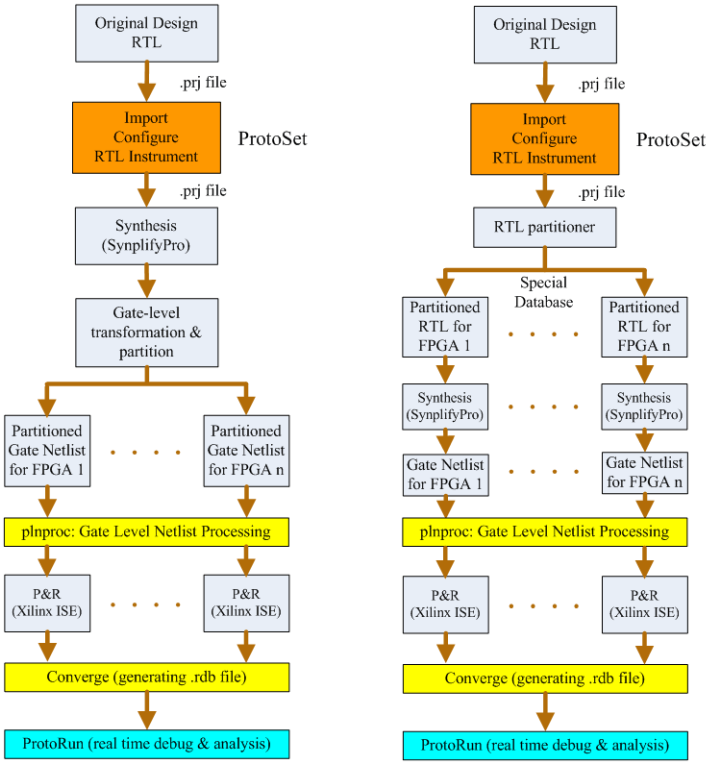

Figure 1: Probe Visualizer Setup Flow

Probe Visualizer will process the original RTL without partition first (using the `protoset -probeg` command). Then the additional step that requires the *plnproc* command utility is needed before going to P&R. This *plnproc* utility is also used in the Full Visibility flow as described in *Take Snapshot* section and *Collect Samples for Full Visibility Flows* section. In other words, the Full Visibility flow will be automatically taken care if used in this process shown in the *Probe Visualizer Setup Flow* figure.

After finishing P&R for all FPGAs, the Converge and Protorun processes are used the same way as described in the *Converge* section and *Runtime* chapter.

Import, Configure, RTL Instrument

Basically these steps are the same as described in the *Setup* chapter. However, because all probes will be selected before doing partition, Protoset will not show the information that can not be decided before partitioning. For example, external probe, clock setting, probe bus, TDM etc. will not be shown in the **-probeg** mode. Some screenshots of the **-probeg** mode are shown below.

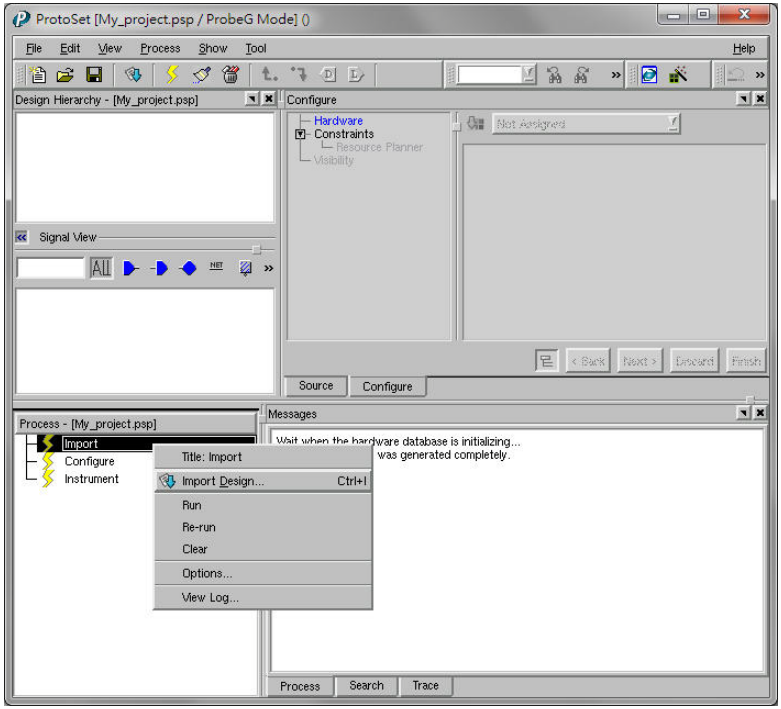

Figure 2: ProtoSet Process - Import Design Setup Step (highlight the differences in -probeg mode)

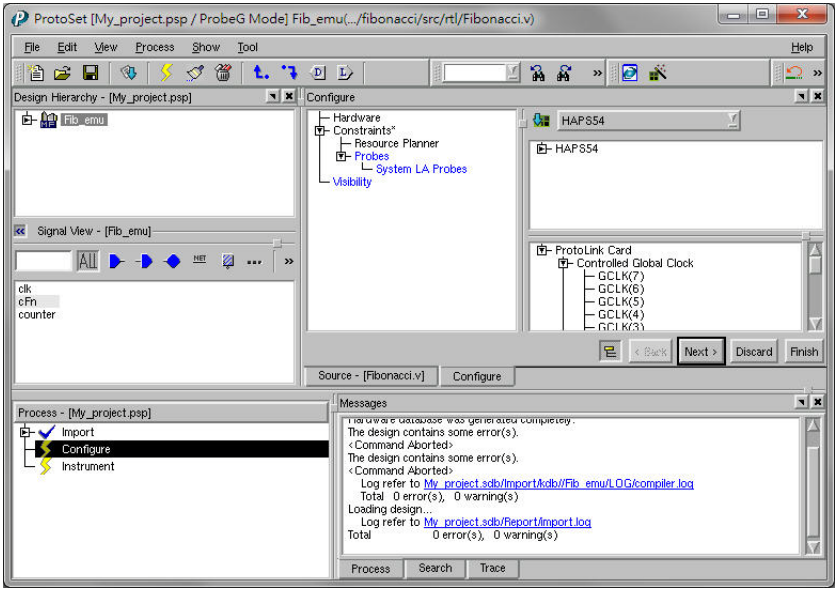

Figure 3: Configure Process (highlight the differences in -probeg mode)

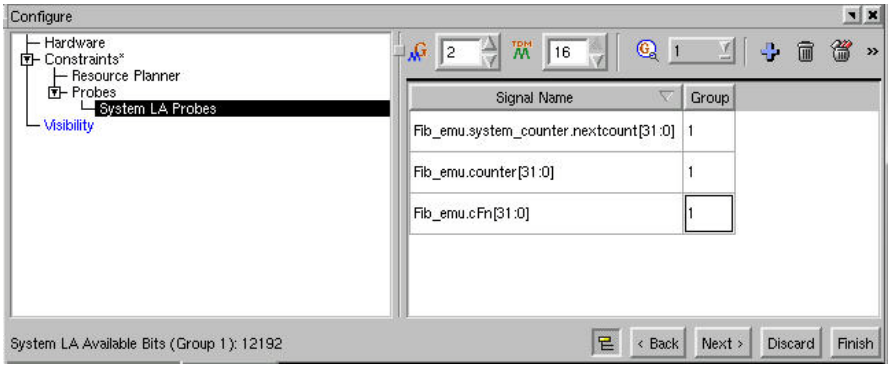

Figure 4: Add Signals for Probing and Specify Probe Group (highlight the differences in -probeg mode)

plnproc

This step is a must-have in the –probeg mode even if Full Visibility flow is used. Additional information is also needed to provide at this stage like the clock setting. For example:

```
> plnproc -psp fib_probeg.psp -cf chip_file -sys_clk
PL_S2C_TAI_LM_brd_F1.pin32 S2C_TAI_LM_brd.F1 -sys_clk
PL_S2C_TAI_LM_brd_F2.pin32 S2C_TAI_LM_brd.F2 -ref_clk
PL_S2C_TAI_LM_brd_F1.pin32
```

The chip file used here is the same chip file as mentioned in Converge section. The module information is needed as shown in the red colored text in the following example to help Probe Visualizer know the top module of each FPGA after the partition.

```
@converge rc file Version 1.0
[S2C_TAI_LM_brd.F1]
MODULE = PL_S2C_TAI_LM_brd_F1

SYN_FILES = Instrument/Fib_emu/Partition/Fib_emu/
S2C_TAI_LM_brd_F1/rev_1/fpga.edf
PNR_DIR = fib_probeg.sdb/INSTR_PNR/S2C_TAI_LM_brd.F1
[S2C_TAI_LM_brd.F2]
MODULE = PL_S2C_TAI_LM_brd_F2

SYN_FILES = Instrument/Fib_emu/Partition/Fib_emu/
S2C_TAI_LM_brd_F2/rev_1/fpga.edf
PNR_DIR = fib_probeg.sdb/INSTR_PNR/S2C_TAI_LM_brd.F2
```

Converge, ProtoRun

The Converge and Protorun processes are used the same way as described in the *Converge* section and *Runtime* chapter.

NOTE: Because all probes are setup before partition, it is possible the number of probes in each FPGA are not balanced. It is also possible some FPGAs may have too many probes without enough routing resources. If that happens, to redo the partition will some proper guidelines may be needed.

Runtime

The Probe Visualizer runtime is used to set up events and trigger conditions, collect samples of the probed signals, save waveforms to the workstation and handle the probe engineering change order (ECO) flow. These will be explained in the following sections.

NOTE: If a Windows 7 or Windows XP (32 bit) runtime host is being used, refer to the release notes for details about the limitations with the platforms.

Hardware Connection

Before runtime, the hardware connection should be complete. The ProtoLink Interface Card uses two methods to connect to the workstation:

- The first method is to connect to the ProtoLink PCIe card inside the workstation through the fiber channel port.
- The second method is to connect to the USB port of the workstation directly.

The steps to complete the connection for the first method are:

1. Install the ProtoLink PCIe card on the workstation with an x8 or x16 PCIe slot.
2. Use the Samtec cable to connect the J-connectors on the prototype board and ProtoLink Interface Card.
3. Connect the fiber channel cable from the ProtoLink Interface Card to the ProtoLink PCIe card on the workstation.

The steps to complete the connection for the second method are:

1. Connect the ProtoLink Interface Card’s USB port to the workstation.
2. Use the Samtec cable to connect the J-connectors on the prototype board and ProtoLink Interface Card.

The *protoInfo* command utility can be used to output the hardware information. For example:

```
> protoInfo

The device is connected to the PCIe (0x11).

The hardware (A ProtoLink) was detected.
The device is connected to the PCIe (0x11).
The PLCard (0) probe memory size is 4GB.

The hardware (ProtoLink) can be used now.
ProtoLink API version = 23-11212012
Device Controller version = fc201002
ProtoLink Controller version = b0cd5112 - build 11302011
```

If a machine other than the runtime host workstation was used to perform Probe Visualizer setup, synthesis, P&R and converge, then the generated runtime database (<project_name>.rdb directory) needs to be copied to the runtime host workstation.

FPGA is programmable so the final FPGA programming files (e.g. bit files) need to be downloaded via the conventional method (e.g. JTAG) to the prototype board before running the software. Alternatively, the FPGA programming file download script can be specified in the *Preferences* form of the *ProtoRun* window, and the specified download script will be invoked (with the bit stream files as arguments) to complete the FPGA download when the *protorun* **Configure** command is executed.

If other external interfaces are required, the connection needs to be secured properly to make sure the hardware is ready to run.

Software Setup

After the hardware is ready, invoke the command `protorun -Probe` to open the *ProtoRun* window (as shown in the *Connect Hardware and Reset Probe LAs in ProtoRun Window* figure) and specify a target connecting channel among all the available channels on the host machine. Multiple PCIe cards and a USB are allowed. The **Scan** command can be used to scan all available connections for selection. The information will be shown on the message pane.

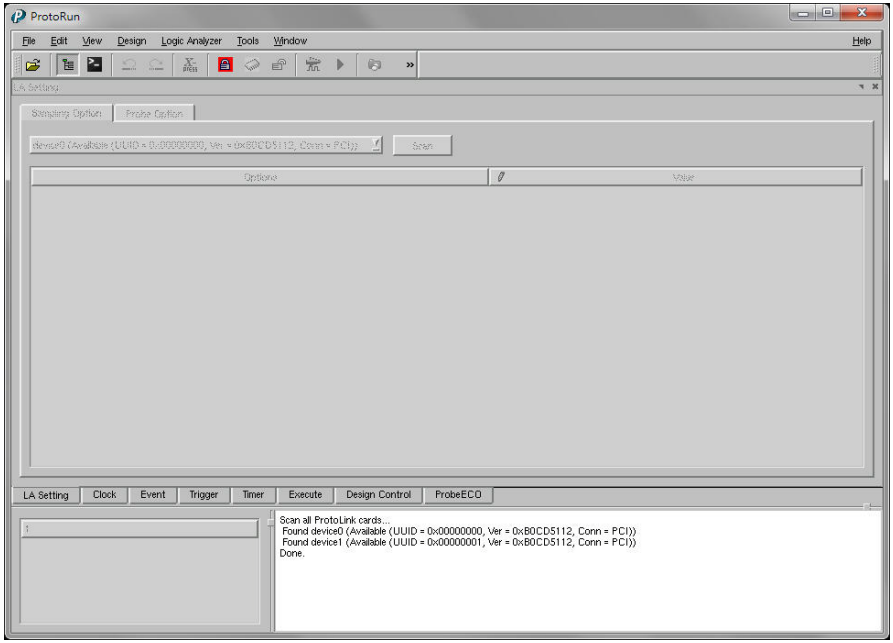

Figure 1: Connect Hardware and Reset Probe LAs in ProtoRun Window

Then, select the corresponding runtime database (the <project_name>.rdb directory in the *Open ProtoLink Runtime Database* form. In the *ProtoRun* window, select the **Logic Analyzer -> Connect** command (or click the **Connect Hardware** toolbar icon) to lock the hardware. Finally, select the **Logic Analyzer -> Configure** command (or click the **Configure Design** toolbar icon) to reset all probe logic analyzers inside the FPGAs.

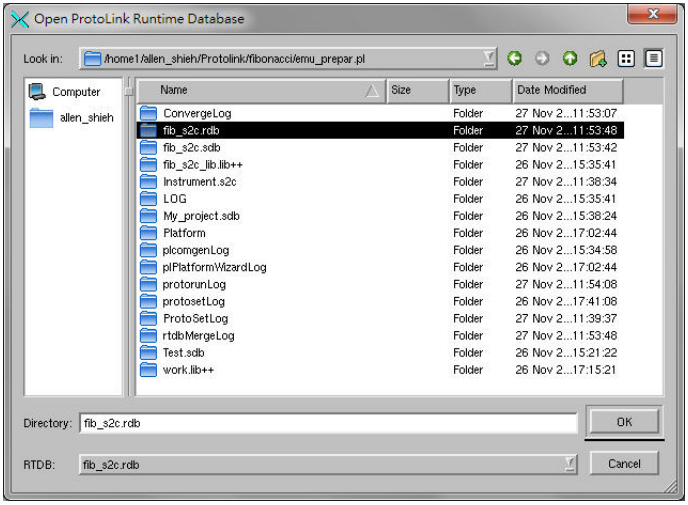

Figure 2: ProtoRun Open ProtoLink Runtime Database Form

NOTE: If the FPGA bit stream files are downloaded via *protorun*, make sure the image download script is set in the *Preferences* form of the *ProtoRun* window before selecting the **Configure** command. Refer to the *Preferences* chapter of the *ProtoLink Command Reference* for more details.

Before setting up events and triggers, configure the logic analyzer settings in the **LA Setting** tab as shown in the *Synchronous LA Settings* figure.

The **Sampling Mode** needs to be selected first. The **Asynchronous Probe** is the default option. When the reference clock has been set in the *ProtoSet* window, the **Synchronous Probe** options are available in the **Sampling Mode** field. The TDM number is automatically calculated when any of the **Synchronous Probe** options are selected.

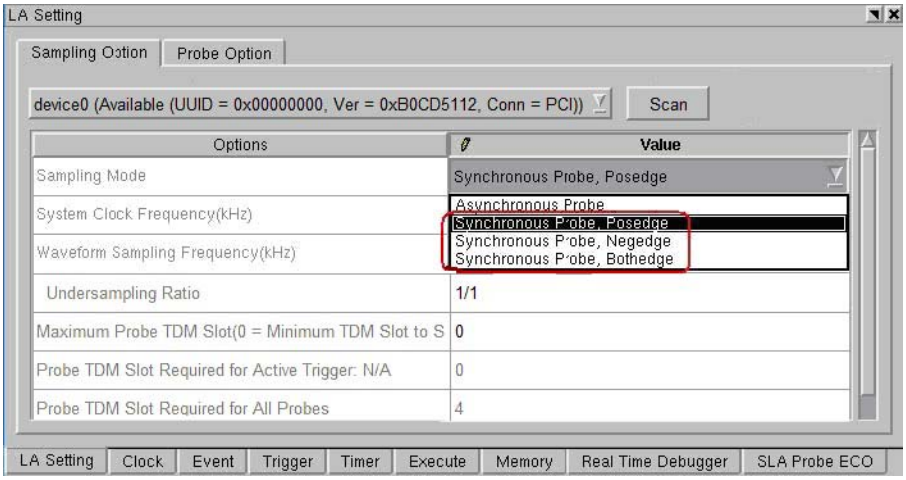

Figure 3: Synchronous LA Settings

After the **Sampling Mode** is decided, the system clock frequency and the waveform sampling frequency will be calculated. You can continue to decide the **Undersampling Ratio** option. The option is used to drop sampled waveform data in a ratio in order to reduce memory usage. You can specify the ratio from 1/1 (default) to 1/16. For example, if 1/3 is specified, two out of the three sampled data points will be discarded.

The **Maximum Probe TDM Slot** (for Asynchronous probes only) is used to reduce the TDM so the speed of the design can be increased if fewer critical signals need to be seen. For example, put the most critical 64 signals on TDM 1 and then set **Maximum Probe TDM Slot** to 1 to only see those signals. The other two fields following the option: the **Probe TDM Slot Required for Active Trigger** and the **Probe TDM Slot Required for All Probes** fields will reflect the change to the **Maximum Probe TDM Slot** option.

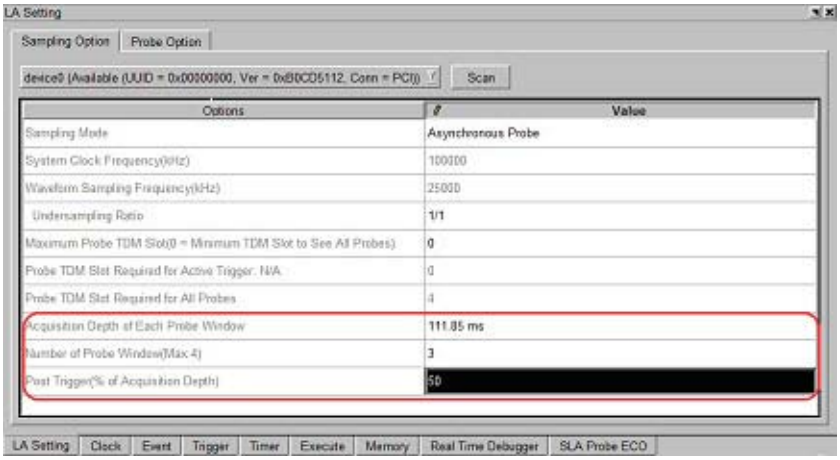

Figure 4: LA Setting Tab - Sampling Time Window

In the *LA Setting Tab - Sampling Time Window* figure, the **Acquisition Depth of Each Probe Window** option is the sampling time window with a maximum value of 1073.74ms based on the 125 MHz sampling clock and the 4G byte probe memory. Several reduced time windows and the last option **Qualifier** are provided for the desired one to be selected. If the **Qualifier** option is chosen, the acquisition will be controlled by the value changes (positive edge to turn on acquisition; negative edge to turn off acquisition) of the **Qualifier** event specified in the **Trigger** tab. Actually, when the **Qualifier** option is selected, the content of the **Trigger** tab will be reduced to allow a trigger to be composed of the **Qualifier** only as shown in the *Reduced Trigger Tab when Qualifier Option is Turned On* figure.

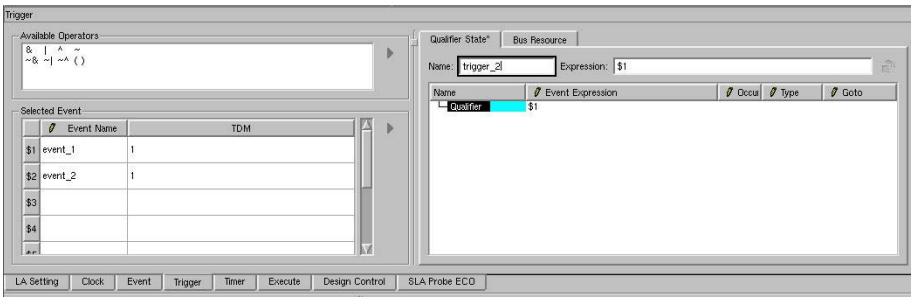

Figure 5: Reduced Trigger Tab when Qualifier Option is Turned On

The **Number of Probe Window** option field in the *LA Setting Tab - Sampling Time Window* figure is for setting the probe window number. The probe window number determines the number of consecutive trigger points to capture (capturing one trigger point after another) within an Arm session. Therefore, the

number is dependent on the memory resource and the valid value is from 1 to a value dependent on the selected acquisition depth.

The **Post-Trigger** option field in the *LA Setting Tab - Sampling Time Window* figure is used to define the percentage of the sampling time window to hold the samples after the trigger. This is adjustable so the best fit can be decided based on the design and trigger conditions.

The *Probe Option Tab in LA Setting* figure shows the **Probe Option** tab of the **LA Setting** tab, each probe bus can be enabled or disabled and a mapping group determined. However, the bus's final assigned group and status is determined by whether the probe bus is used in the trigger condition.

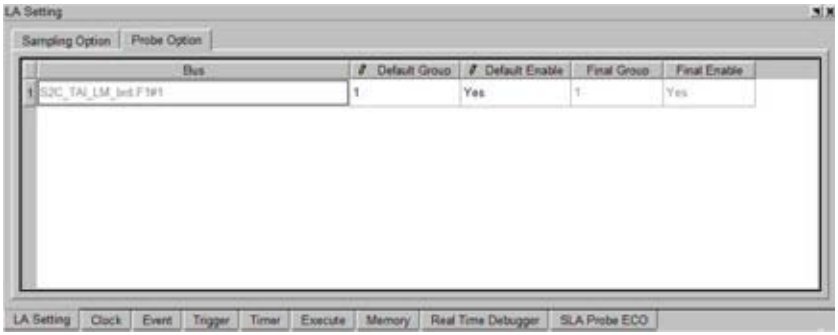

Figure 6: Probe Option Tab in LA Setting

Default Group: Use the default probe group to sample from the probe bus. This default value will be overridden if the System LA trigger uses probe values from a different group on the probe bus. The resulting probe group is shown in the **Final Group** column of the **Probe Option** tab.

Default Enable: Enables default probing from up to six probe busses for each ProtoLink controller. This default will be overridden if the System LA trigger uses probes from a probe bus that are not enabled by default. After the trigger's required probe busses are considered, the first enabled-by-default probe busses in the list will be included until the maximum of six per ProtoLink controller is met. The final enabled probe busses are listed in the **Final Enable** column of the **Probe Option** tab.

The controller generates a few controllable clocks (GCLK) and supplies them to the board. These clocks may be configured in terms of changing the frequency and waveform before (optionally) consuming them inside the design.

The **Clock** tab helps configure the protolink controllable clocks.

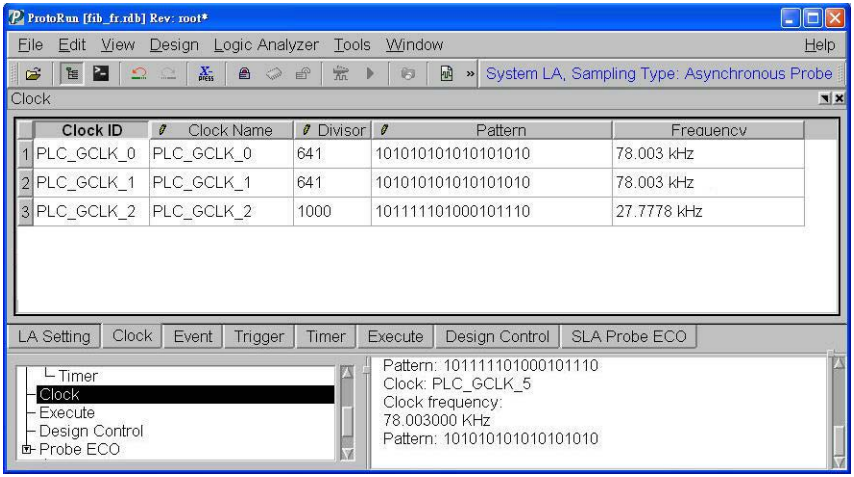

Figure 7: ProtoRun Clock Tab

In the above example are three controllable clocks that can be configured and optionally used to drive the design. Both the clock frequency and waveform shape can be changed. The **Divisor** column indicates the factor by which the system clock frequency may be reduced. The **Pattern** column determines both the shape and the frequency based on the rising edges in the pattern and the pattern length.

The resultant frequency is calculated as follows:

Frequency = (system clock freq. / divisor / length of pattern) * number of rising edges of the pattern.

The "number of rising edges of the pattern" should include the changes introduced by LSB and MSB since the pattern will be applied repeatedly. For example:

1. "0101" ==> "0101 0101 0101..." ==> 2 rising edges
2. "1010" ==> "1010 1010 1010..." ==> 2 rising edges
3. "101" ==> "101 101 101..." ==> 1 rising edge
4. "010" ==> "010 010 010..." ==> 1 rising edge
5. "011010" ==> "011010 011010..." ==> 2 rising edges
6. "101100" ==> "101100 101100..." ==> 2 rising edges

In the above figure, the system clock frequency is 100 MHz. The pattern for PLC_GCLK_2 is "101111101000101110“ that contains 5 rising edges including the LSB->MSB over a length of 18.

Hence, the resultant frequency is calculated as follows:

Frequency = 100MHz / 1000 / 18 * 5 = 27.7778 KHz.

Setup Events and Triggers

Just like when the logic analyzer is used, the built-in LAs inside the FPGAs will start taking samples when **Arm** (the hardware starts running) is clicked. Therefore, events and triggers need to be setup before **Arm** to define the time or scenario to stop taking samples.

The Probe Visualizer also provides an **Express Mode** icon on the toolbar for users who only use simple events as triggers and can skip the steps to edit an event and trigger first. If you are such a user, you can go to the *Collect Samples for Express Mode* section directly.

In the *ProtoRun* window, click the **Event** tab as shown in the *Event Tab* figure and then click the **New Event** button to create a new event. New events can be created based on probed signals listed in the **Probe Signal** section. Basically, any probed signal can be used to compose an event as long as it belongs to the same probe group. This is because signals from different probe groups are collected based on different runs so only signals from the same probe group can be seen during a single run. The current editing event can be saved as new trigger by selecting the **Save As New Trigger** button which opens the *Save as New Trigger* form as shown in the *Save As New Trigger Form* figure.

Figure 8: Event Tab

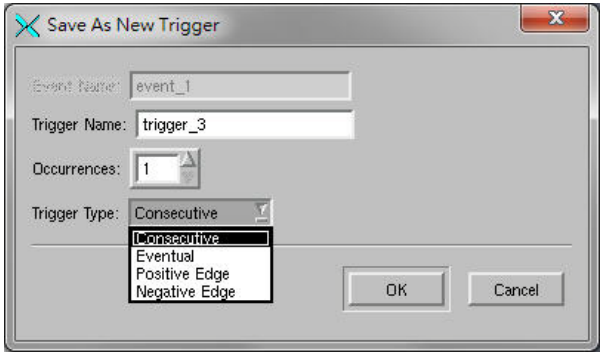

Figure 9: Save As New Trigger Form

The *Trigger Tab* figure shows the **Trigger** tab. A trigger can be composed of a maximum of sixteen states and a qualifier (optional). In each state, four condition branches are supported. These states will be checked consecutively so make sure to put them in the correct sequence. A different number of occurrences and trigger types can be selected for each selected event to form the required state. The qualifier is for specifying a qualifier event expression to control the dumping on (value 1) and off (value 0) within a probe window.

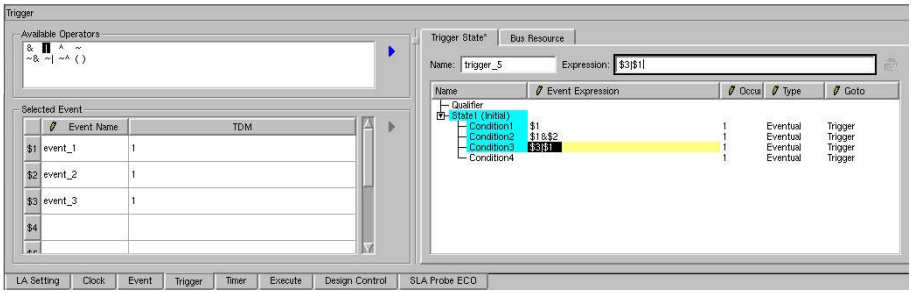

Figure 10: Trigger Tab

During a single run, up to 6 probe buses can be seen so the trigger condition can only use events containing signals from FPGAs containing those 6 probe buses.

The event-based trigger condition may not occur for each run so a timer can be set as active in the **Timer** tab as shown in the *Timer Tab* figure to define another stop condition based on elapsed time. The time specified is the actual time elapsed after Arm. The system clock frequency shown in the *LA Setting Tab - Sampling Time Window* figure is used as a reference to calculate the actual time.

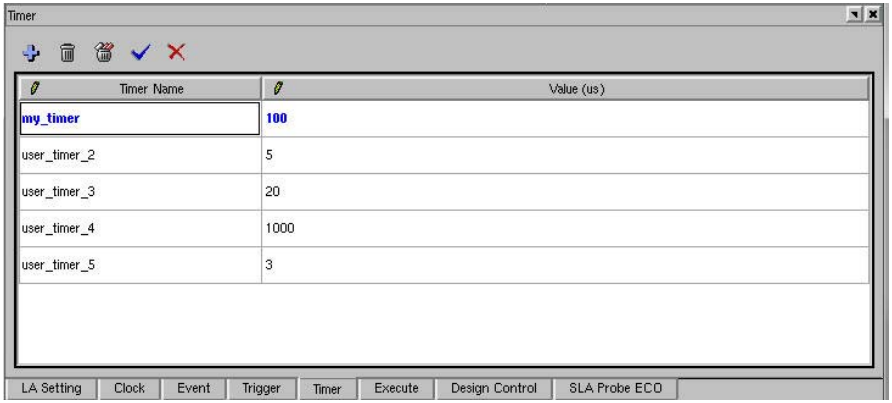

Figure 11: Timer Tab

In some scenarios, people may want to use external Logic Analyzer to sample some extremely high speed signals and also use the ProtoLink Probe Visualizer at the same time to sample other regular speed signals. To make sure the trigger coming from either the external Logic Analyzer or the Probe Visualizer can also trigger the other, the ProtoLink Interface Card provides pins for connecting with an external Logic Analyzer to send its own trigger out or receive a trigger from an external Logic Analyzer.

Collect Samples

Either **Probe Trigger** or **Timer** can be enabled as the stop condition for sampling. If both are enabled, the one that occurs first will stop the sampling. Turn on the **Auto Dump** option as shown in the *Execute Logic Analyzer* figure to save FSDB waveforms whenever triggered. You can also choose to save FSDB waveforms more selectively by turning off the **Auto Dump** option as saving FSDB waveforms on the workstation may take a few minutes.

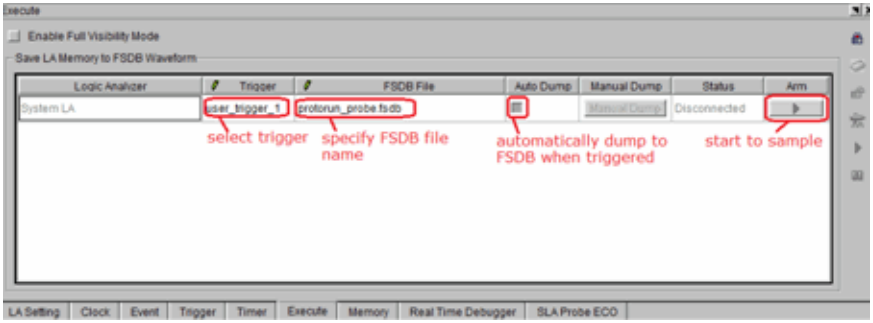

Figure 12: Execute Logic Analyzer

Collect Samples for Express Mode

The Probe Visualizer also provides an **Express Mode** icon on the toolbar as shown in the *ProtoRun Express Mode* figure for users who only use simple events as triggers and can skip the steps to edit an event and trigger first. After turning on the **Express Mode** icon, everything except the **LA Setting** and **Execute** tabs are hidden.

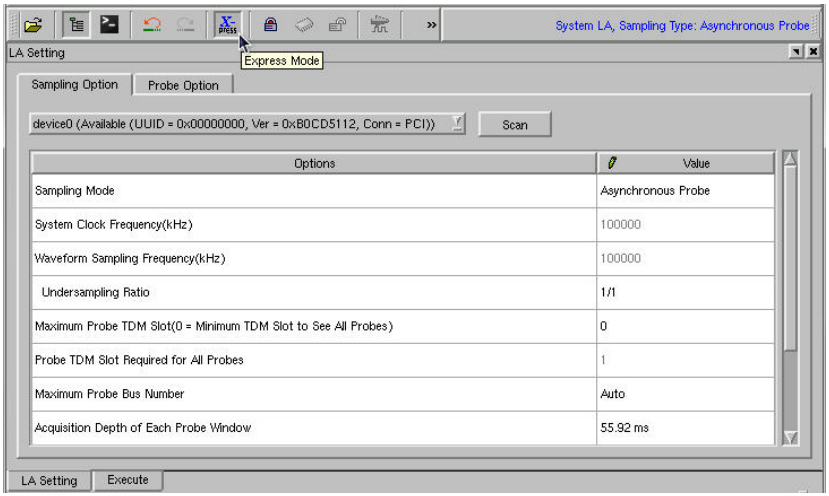

Figure 13: ProtoRun Express Mode

Select the **Execute** tab.

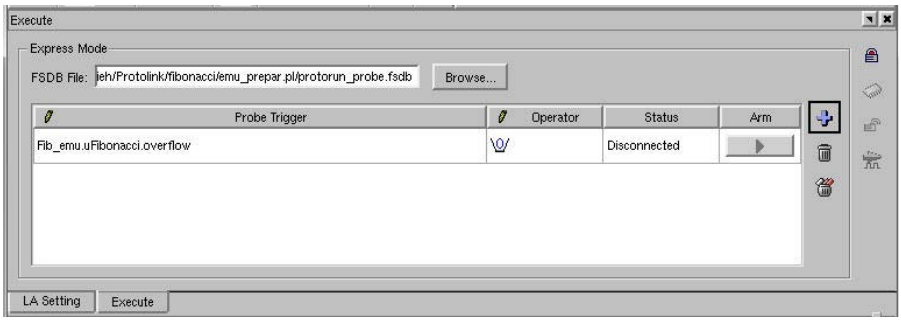

Figure 14: Execute Tab in Express Mode

Select the **New Probe Trigger** button to open the *Trigger Editor (Express Mode)* form to select a probe.

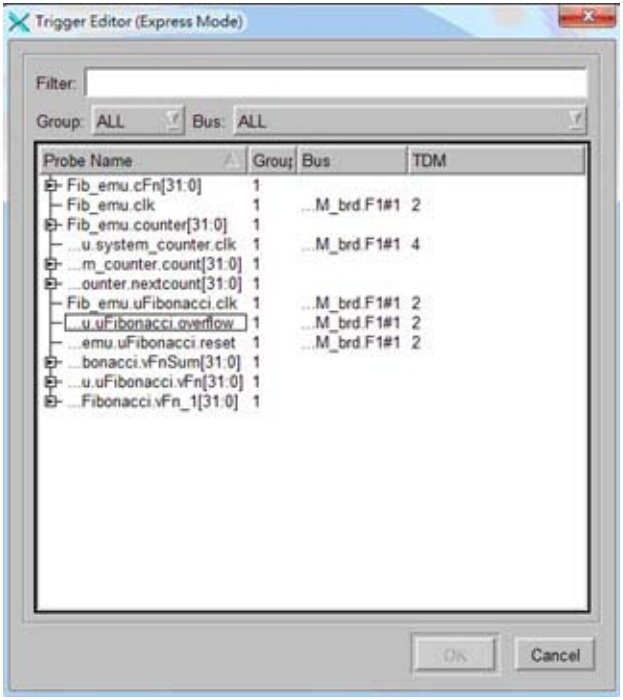

Figure 15: Trigger Editor (Express Mode) Form

Click the **OK** button to close the *Trigger Editor* form. On the **Execute** tab, edit the **Operator** field of the probe trigger.

After the required probe triggers are edited, select the **Logic Analyzer -> Connect** and **Logic Analyzer -> Configure** commands (or click the **Connect Hardware** and the **Configure Design** toolbar icons) if you have not done so, and select the **Arm** button for any of the available probe triggers to start triggering.

When the trigger or timer has been reached to stop sampling, select the **Logic Analyzer -> Disconnect** command (or click the **Disconnect Hardware** toolbar icon) to release the prototype board hardware. Now it is time to start using the Verdi system to view the waveform and debug the RTL. You can invoke the Verdi system in a separate shell window or invoke it directly from inside the runtime software.

Take Snapshot

The Probe Visualizer product uses the Snapshot bus to capture FPGA internal states to text or FSDB files. To use the **Take Snapshot** function:

1. Enter an essential list in the Visibility section of the Configure step during the ProtoSet setup process.
2. Right after synthesis and before P&R, execute the *plnproc* command utility as follows:
   a. Use the following command to generate a template chip file if it does not already exist. For example:

```
> plnproc -psp s2c_fv.psp -cft chip_file.s2c
```

   b. Modify the chip file if required to indicate the name of the instrumented EDIF file.
   c. Use the following command to insert PNR cell (FF) to latch the essential signals with registers. For example:

```
> plnproc -psp s2c_fv.psp -cf chip_file.s2c
```

The above steps are used to instrument the synthesized netlist (EDIF) to ensure the essential signals are preserved during P&R. They are also similar to the prerequisite steps for the full visibility flow (see the *Collect Samples for Full Visibility Flows* section).

NOTE: If the **OUTPUT_EDIF** field is blank, the instrumented EDIF file will be created as follows:
`<Instrument dir>/GATE_TRAN/<top module>.edf`

After the steps are done, the essential signals will be latched with registers and snapshot(s) can be captured on these registers any time during the Arm session in the *ProtoRun* window.

At any time during the Arm session, either stopped or running, select **Design -> Take Snapshot** to dump values of the essential signals to a text or FSDB file. The output file name can be configured in the **Snapshot** page of the *Preferences* form. An example of the content of the dumped snapshot text file is as follows:

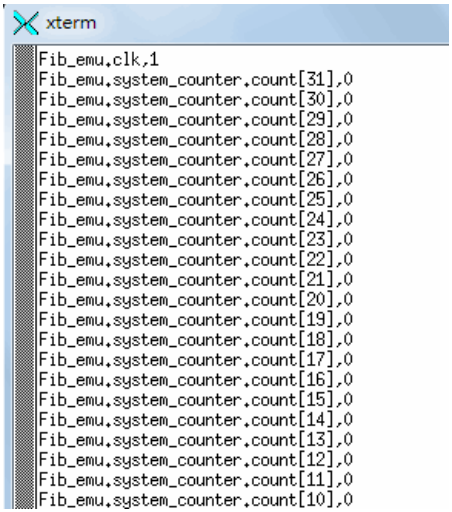

Figure 16: Snapshot Text File

Collect Samples for Full Visibility Flows

The Full Visibility flow helps to capture FPGA internal states to FSDB files. With the flow, the essential signals will be latched with registers and snapshot(s) of these registers can be captured from any paused (triggered) time point in *ProtoRun* window. The non-essential parts can be derived with the Siloti system. Therefore, the fewest resources (latched registers for the essential signals) can be used to achieve full visibility (derived with the Siloti system).

To use the flow:

1. Enter an essential list in the Visibility section of the Configure step during the ProtoSet setup process.
2. Right after synthesis and before P&R, execute the *plnproc* command utility as follows:
   a. Use the following command to generate a template chip file if it does not already exist. For example:

```
> plnproc -psp s2c_fv.psp -cft chip_file.s2c
```

   b. Modify the chip file if required.
   c. Use the following command to insert PNR cell (FF) to latch the essential signals with registers. For example:

```
> plnproc -psp s2c_fv.psp -cf chip_file.s2c
```

A Full Visibility (FV) dump snapshots the FPGA register states on a time point to the FSDB file. Each FV dump needs to take several milliseconds (ms). Refer to the following figure for an illustration of the clock waveform affected by FV dumps.

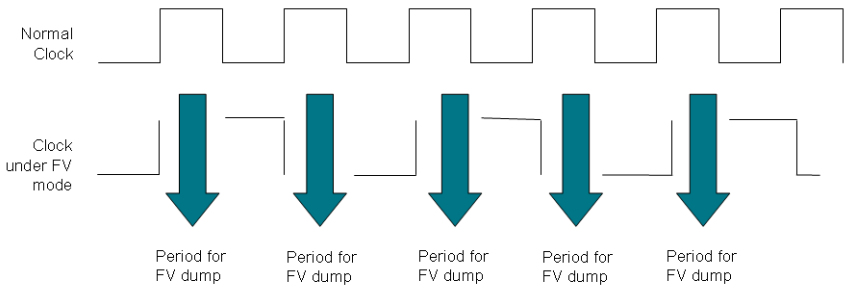

Figure 17: Clock Waveform Affected by FV Dumps

The typical usage of the FV dumping is that a target cycle/time that has bugs is known and the FPGA register states near the cycle/time to inspect the issue need to be dumped. The rough steps are:

1. Edit a trigger or timer and run Arm to stop on the desired time point.
2. Then enable the Full Visibility flow and configure the required number of FV dumps. Run Arm to dump FPGA states cycle by cycle.

Refer to the *FV Dump with ProtoLink Controlled Clock* section for more details.

To perform the FV dumps based on a controlled clock (user controlled clock) instead of the ProtoLink controlled clock, the steps will be different than the above. Refer to the *FV Dump with User Controlled Clock* section for the detailed steps.

FV Dump with ProtoLink Controlled Clock

Refer to the following figure for an illustration of an FV dump with a ProtoLink controlled clock.

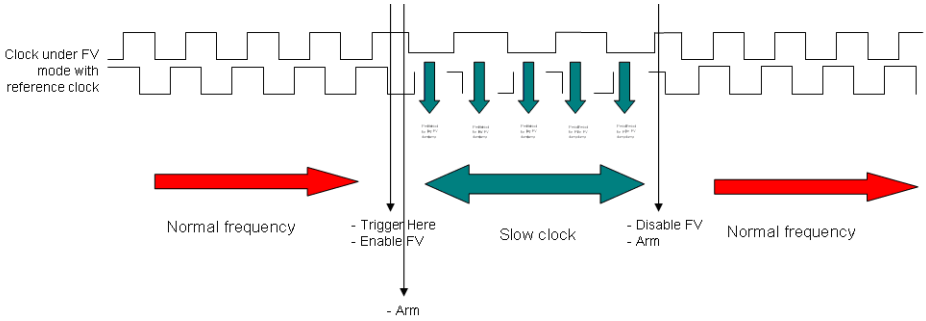

Figure 18: FV Dump with ProtoLink Controlled Clock

The requirements of cycle-by-cycle FV dumping using a ProtoLink controlled clock are:

1. The design clock is bound to a ProtoLink controlled clock.
2. The design clock can be stopped.

The steps for FV dumps with a ProtoLink controlled clock are:

1. Invoke the *ProtoRun* window (`protorun -Probe`) and open the runtime database (.rdb directory).
2. Edit a trigger in the **Event** and **Trigger** tabs. This trigger is for stopping before the target cycle to perform the FV dump.
3. Select the **Logic Analyzer -> Connect** command (or click the **Connect Hardware** toolbar icon) to connect the hardware.

4. Go to the **Execute** tab and select the **Enable Full Visibility Mode** option.

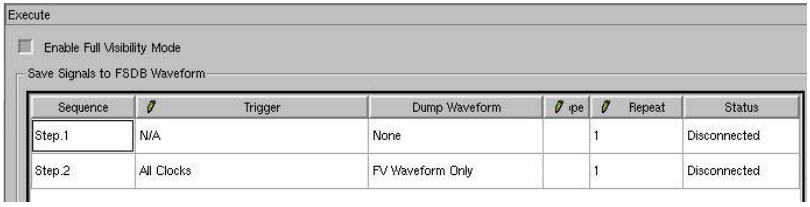

Figure 19: Full Visibility Mode Option

1. In the **Execute** tab, do the following:
   a. Select the **Properties** command to bring up the *Preferences* form.
   b. Specify the output directory of the FSDB file for FV dumps in the **Output Directory** field.

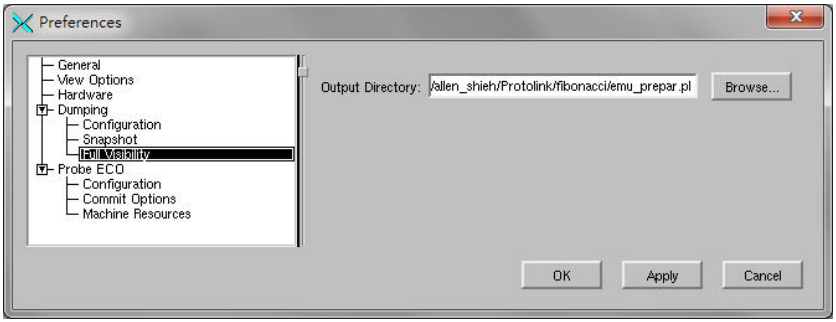

Figure 20: Full Visibility Mode Preferences

   c. Initially, there are two rows contained in the **Save Signals to FSDB Waveform** list in the **Execute** tab. The “Step.1” row is for specifying a trigger that can stop at the first cycle for FV dumping. The “Step.2” row is for specifying the reference clock for FV dumping and specifying the required number of FV dumping (captured on both clock edges) in the **Repeat** field.

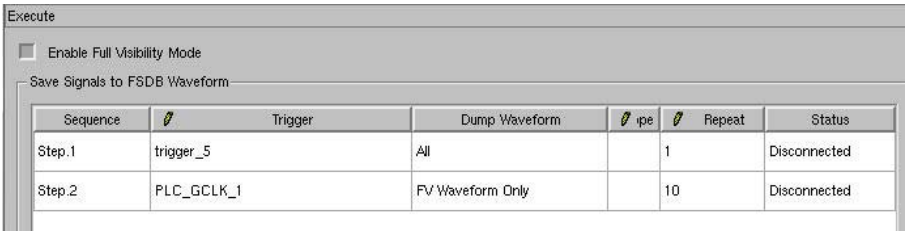

Figure 21: Save Signals to FSDB Waveform

   d. Extra rows can be added to conduct consecutive rounds of FV dumping in the same Arm session.

e. Select the **Logic Analyzer -> Configure** command (or click the **Configure Design** toolbar icon) to reset all Probe Visualizer embedded logic analyzers inside the FPGAs.

f. Select the **Logic Analyzer -> Arm** command (or click the **Arm** toolbar icon) to start probing and the FV dumping.

The FV dump results (*fsdb.vf*) are output to the specified **Output Directory** on the **Full Visibility** page of the *Preferences* form. The *fsdb.vf* file is a virtual file referring to multiple FV dump FSDB files. The *fsdb.vf* file can be opened in the Verdi system and then **Get All Signals** can be performed to browse the results. Refer to the following figure.

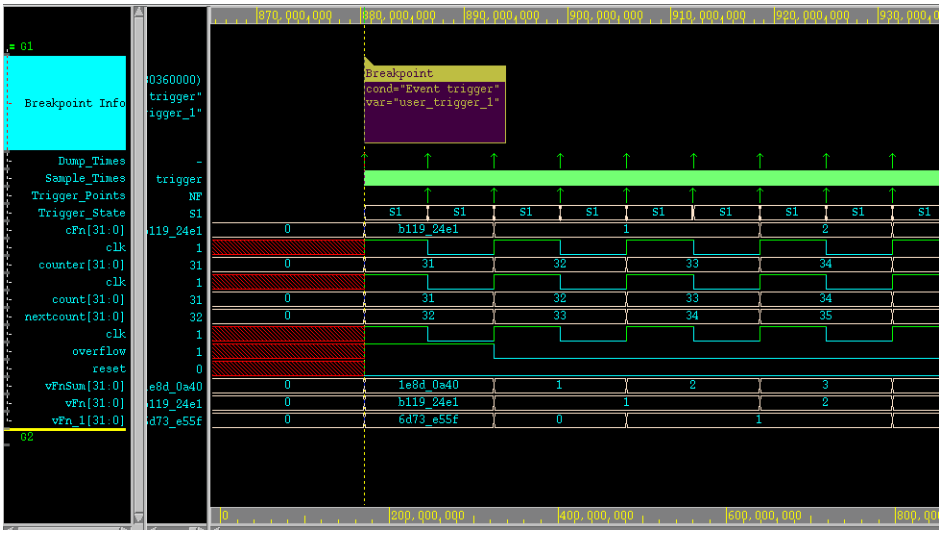

Figure 22: Results of FV Dumping with ProtoLink Controlled Clock

FV Dump with User Controlled Clock

Refer to the following figure for an illustration of the FV dump with a user controlled clock.

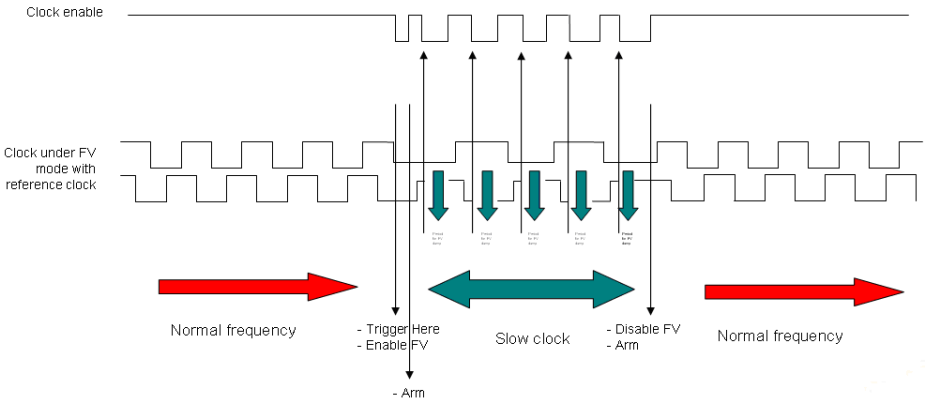

Figure 23: FV Dump with User Controlled Clock

The requirements of cycle-by-cycle FV dumping using user controlled clock are:

1. The design clock should be gated with the CLK_ENB signal generated from the ProtoLink controller.
2. The design clock can be stopped.

To perform FV dumps with a given clock, ensure that the clock is controlled by the CLK_ENB signal generated from the ProtoLink controller. Contact your local AE for the *Design User Prototype Board to Connect with ProtoLink Interface Card (Probe Capability Only)* application note for more details.

NOTE: The ProtoLink platform supports up to a 10MHz user controlled clock.

The steps for FV dumps with user controlled clock are:

1. Invoke ProtoRun (`> protorun -Probe`) and open the runtime database (.rdb directory).
2. Edit the first trigger in the **Event** and **Trigger** tabs. This trigger is for stopping before the target cycle to perform the FV dump.
3. Edit the second trigger in the **Event** and **Trigger** tabs. This trigger will trigger before the second and later cycles to perform the FV dump. Usually, the second trigger is a clock edge trigger of the user controlled clock (controlled by the CLK_ENB signal generated from the ProtoLink controller).

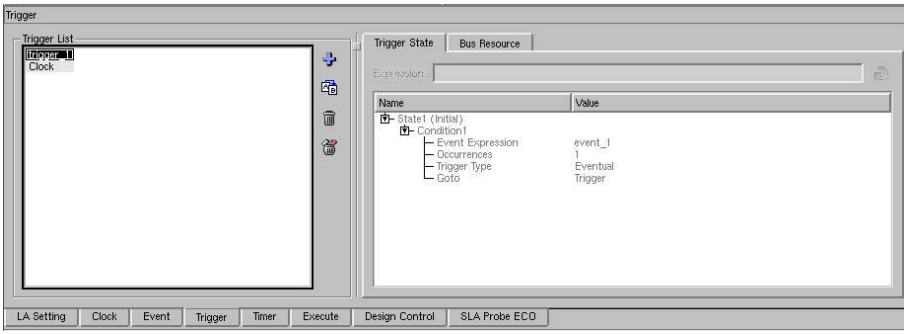

Figure 24: Triggers

1. Select the **Logic Analyzer -> Connect** command (or click the **Connect Hardware** toolbar icon) to connect the hardware
2. Go to the **Execute** tab and select the **Enable Full Visibility Mode** option.

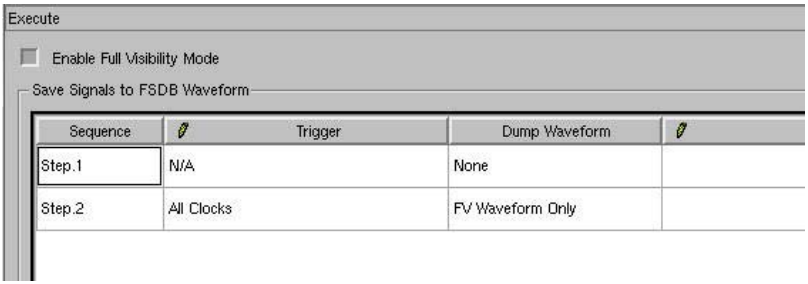

Figure 25: Full Visibility Mode Option

1. In the **Execute** tab, do the following:
   a. Select the **Properties** command to bring up the *Preferences* form.
   b. Specify the output directory of the FSDB file for FV dumps in the **Output Directory** field.

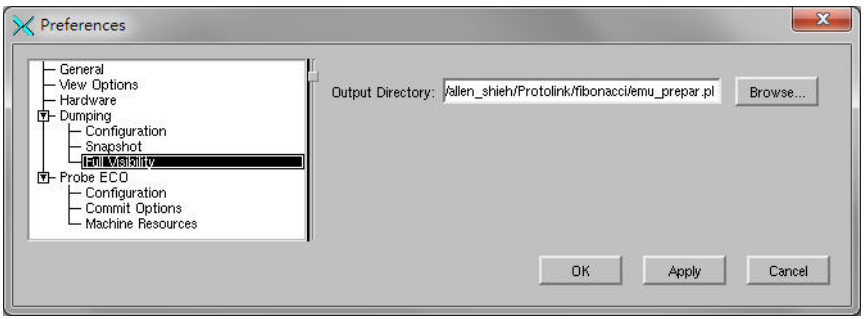

Figure 26: Full Visibility Mode Preferences

c. Initially, there are two rows contained in the **Save Signals to FSDB Waveform** list in the **Execute** tab. The “Step.1” row is for specifying a trigger that can stop at the first cycle for FV dumping. The “Step.2” row is for specifying a clock edge trigger of the user controlled clock and specifying the required number of FV dumping in the **Repeat** field.

d. Extra rows can be added to conduct consecutive rounds of FV dumping in the same Arm session.

e. Select the **Logic Analyzer -> Configure** command (or click the **Configure Design** toolbar icon) to reset all Probe Visualizer logic analyzers inside the FPGAs.

f. Select the **Logic Analyzer -> Arm** command (or click the toolbar icon) to start probing and the FV dumping. The clock_enb signal will assert high when armed and go to low when the first trigger triggers. Then, the specified number of FV dumps will be captured on each occurrence of the clock trigger.

The FV dumping results (*fsdb.vf*) are output to the specified **Output Directory** on the **Full Visibility** page of the *Preferences* form. The *fsdb.vf* file is a virtual file referring to multiple FV dump FSDB files. The *fsdb.vf* file can be opened in the Verdi system and then **Get All Signals** can be performed to browse the results. Refer to the following figure.

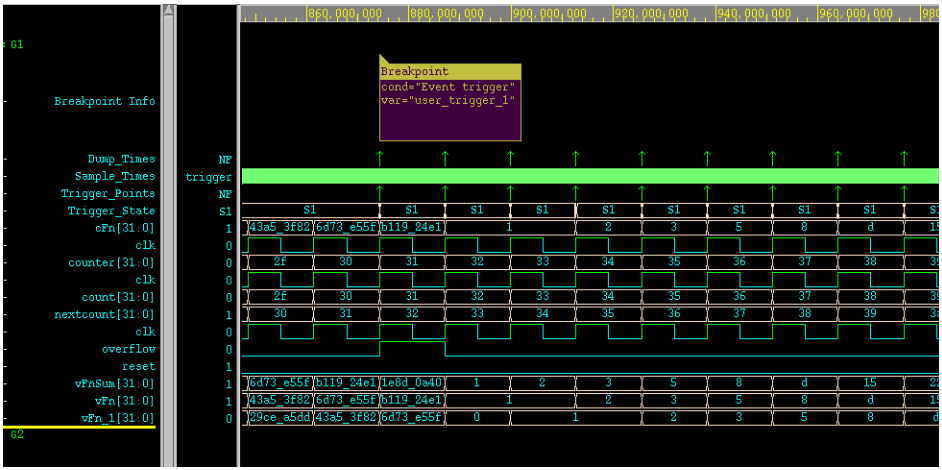

Figure 27: Results of FV Dumping with User Controlled Clock

Design Control

To control the design contents during runtime like changing the memory contents or controlling specific signals to specific values, users can use Design Control tab shown in the following figure.

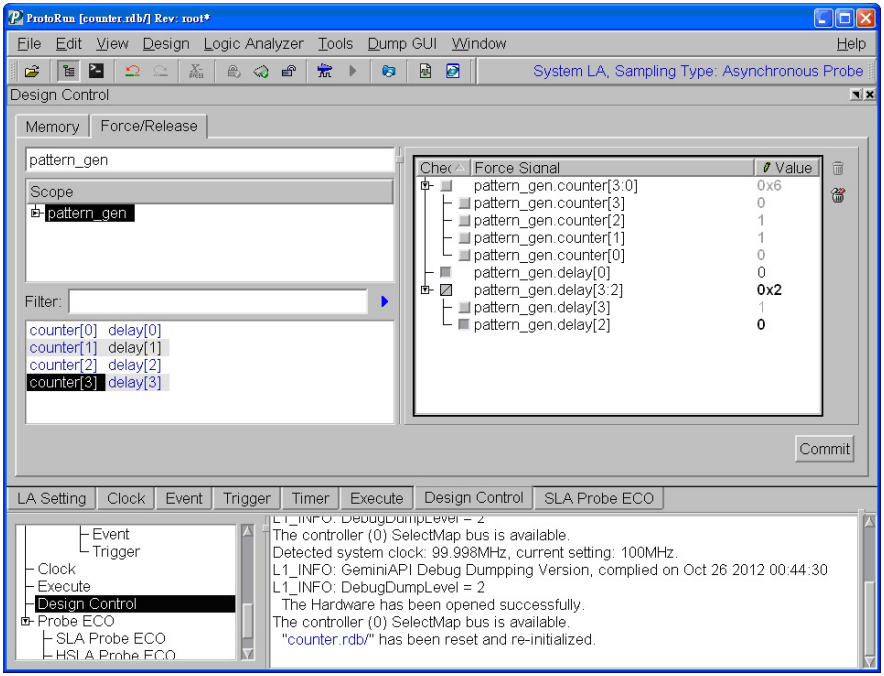

Figure 28: Design Control Tab

The **Force/Release** tab will let users select the required signals for the **Force Signal** list pane and control the values/actions during runtime. If the **Check** column of the specific signals in the list pane are checked, it implies those signals will be forced to the values as shown after clicking the **Commit** button. Otherwise, it implies those signals will be released to go back to the running values after clicking the **Commit** button. Partial check can only be seen on a bus node if not all of its child nodes are in the same states. All force/release changes will be done together when the **Commit** button is clicked. The force/release list can also be saved into runtime database using the **File -> Save** command.

Sometimes users may want to reuse some of the settings like the trigger conditions and force/release signals across multiple runtime databases. Users can use the **File -> Import Settings/Export Settings** command as shown in the *Import/Export Settings Form* figure to import or export those setting into a runtime setting file (.rsf) so it can be reused across multiple runtime databases.

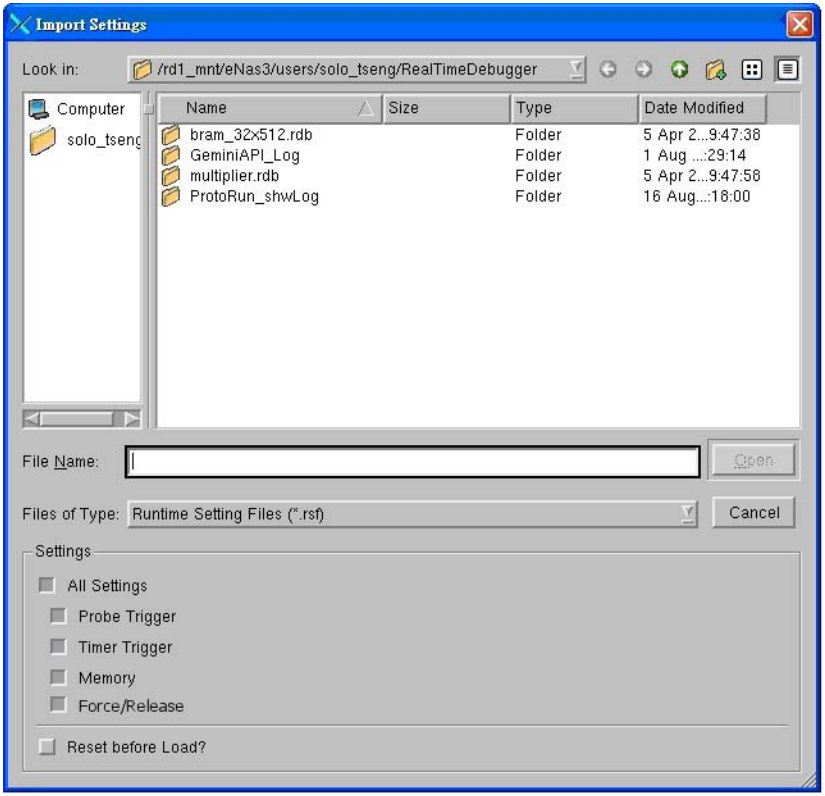

Figure 29: Import/Export Settings Form

Probe ECO

Extra signals are often required during the debug session. Usually this can only be completed by going back to add those extra signals to the probed signal list and going through the entire setup process again which can take several hours, leading to a one-day turnaround time. Using the Probe Visualizer product, a built-in Probe ECO process can do this quickly to reduce the required time to minutes to save time.

The Probe ECO process is possible because all available signals are recorded with their physical location inside the FPGA when running ProtoLink Converge. With the knowledge from the runtime database (.rdb file), the software can incrementally generate a partial routing script for Xilinx ISE or Quartus to reduce the time needed for regenerating an FPGA programming file with the new probes.

The synthesis and P&R process may sometimes optimize desired signals. To avoid this, a list of signals that should be preserved can be provided for Probe ECO. The Probe Visualizer setup flow will make sure those signals are not optimized and can still be seen in the **SLA Probe ECO** tab as shown in the *SLA Probe ECO Tab* figure. To select the correct signals, the Essential Signal Analysis engine from the Siloti system can also be used to generate the list to obtain the highest possible coverage.

The following figure shows the **SLA Probe ECO** tab where Probe ECO can be executed. The **Current Probe** tab lists the existing probed signals and the **Modified Probe** tab lists the new probed signals to be added. The required signals from the signal window can be dragged and dropped to the **Modified Probe** tab section. The signal window contains both **RTL** and **Gate-Level** tabs. The **Gate-Level** tab includes signals that could not be mapped/found in RTL. Signals can be dragged and dropped directly from the Verdi system based on the debug results to save time finding the required signals.

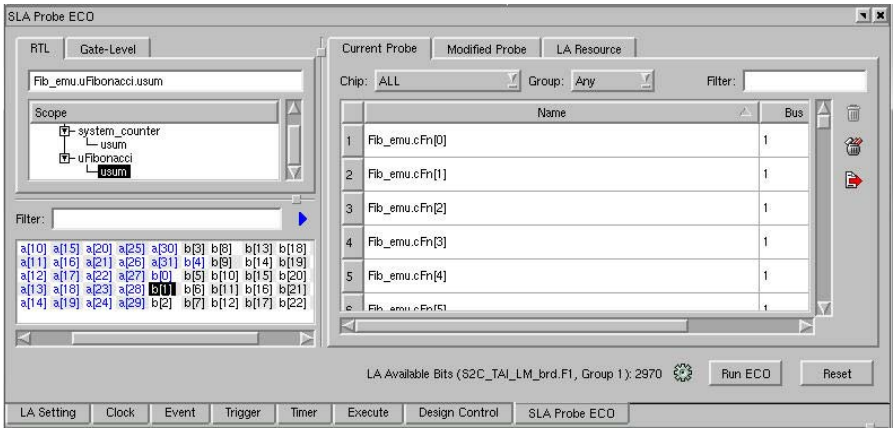

Figure 30: SLA Probe ECO Tab

Sometimes a current probe needs to be removed to leave space for newly added signals. This can be accomplished by selecting those signals in the **Current Probe** tab and clicking the **Delete** icon. Also, in both the **Current Probe** tab and the **Modified Probe** tabs, the **Export Probe** icon can be selected to export the probe list to a text file in CSV format. In the **Modified Probe** tab, the **Import Probe** icon can be selected to import a probe list from a CSV file.

After all addition and deletion actions have been completed, click the **Run ECO** button to generate the new FPGA programming file. If extra options need to be added for the bitgen process, select the **Probe ECO -> Commit Options** section of the *Preferences* form (invoked by selecting the **Tools -> Preferences** command) and edit the bitgen options. The Probe Visualizer product supports submitting the Run ECO job to multiple machines in distributed and parallel mode to save time. This can be configured in the **Probe ECO -> Machine Resources** section of the *Preferences* form. Refer to the *Preferences* chapter of the *ProtoLink Command Reference* for more details.

If ProtoSet and ProtoRun have been running on different machines, the Probe ECO step can also be run on the setup machine as the Probe ECO step does not require the connection to the hardware. This is commonly seen because the setup machine is usually more powerful and has the ISE (or Quartus) tool (required by Probe ECO) properly installed.

With the newly generated FPGA programming file, download the file first, and then click **Configure**, and **Arm** to get the new waveform with the updated probed signals.

The Probe ECO process can also be applied to **External Probes** which are used to connect to the external logic analyzer to monitor some high speed signals from

external IO pins. The same partial routing method can be applied to reroute the signals to external pins to perform the external Probe ECO process as shown in the *External Probe ECO Tab* figure.

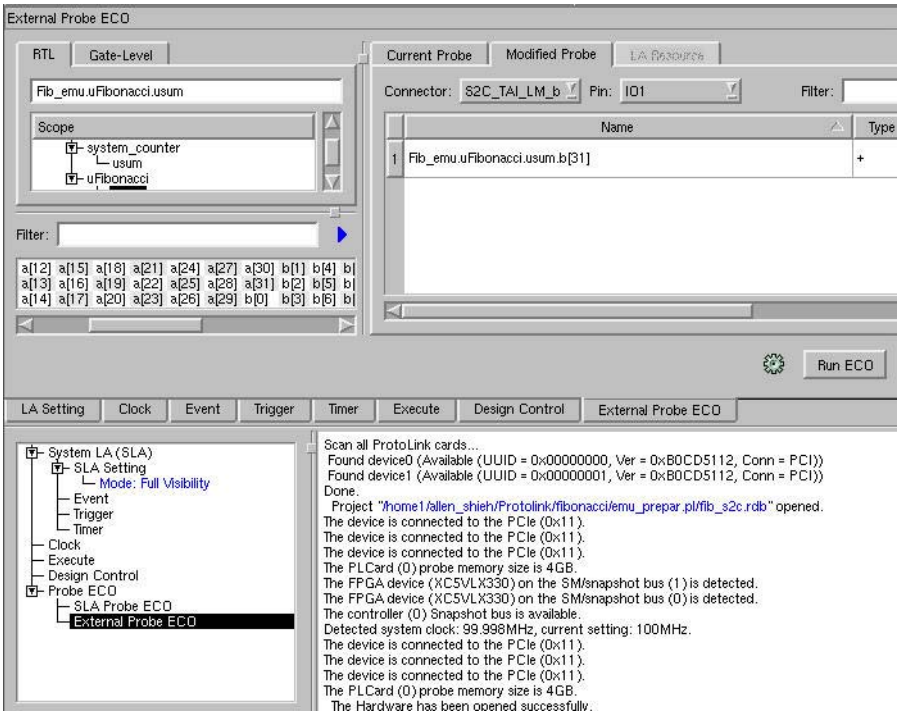

Figure 31: External Probe ECO Tab

Run Probe ECO in Batch Mode

Probe ECO can be performed in batch mode with the **-runProbeECO** command line option of the *protorun* utility. Refer to the *protorun* utility description in the *ProtoLink Command Reference* for details. For example:

```
> protorun -rdb test1.rdb -rev rev1.0 -runProbeECO -SLA
new_probes.csv -export current_probes.csv
```

The *protorun* utility will add new probes from “new_probes.csv” and run Probe ECO to perform the relevant changes and output the current probe list to “current_probes.csv”.

After executing Probe ECO in batch mode, check the log file “plrDumpLogWin.log” under the “ProtoRunLog” directory to check for any related warnings or errors.

Revision Management

In the *ProtoRun* window, the runtime databases can be saved as new revisions and managed in the opened project. Therefore, incremental changes can be saved to different revisions and an appropriate one picked from the saved revisions to run the next probe session. The revisions are actually snapshots of the current runtime database and thus include the complete data of event, trigger, LA settings and Probe ECO.

For example, a trigger was set and the waveform dumped. Now a different trigger needs to be set to capture waveforms for a different scenario. The trigger can be changed and saved to a new revision (via the **File -> Save As** command and select **New Revision** as the saving type). Later on, the setting with the previous trigger can be restored by selecting the previous revision in the *Revision Management* form (opened with **File -> Revision Management**).

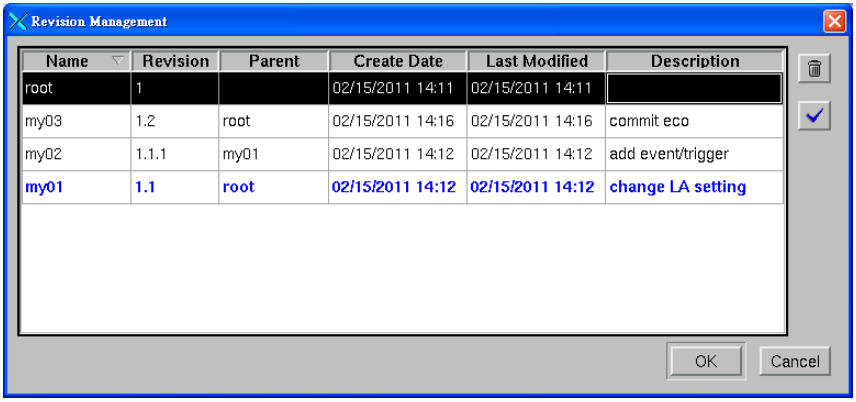

Figure 32: Revision Management Form

Application Tutorials

The tutorial data resides in the *<$PROTOLINK_INST_DIR>/demo* directory. Copy the files from this directory into your working directory.

If you have not yet prepared your own tutorial data, follow the procedures below in the UNIX environment:

```
> mkdir <working_dir>
> cp -r <$PROTOLINK_INST_DIR>/demo/ <working_dir>
> cd <working_dir>/demo/probe/fibonacci/project
```

Setup Flow

Create a New Project

1. Open the *ProtoSet* window.

   ```
   > protoset -Probe
   ```

2. In the *ProtoSet* window, select the **File -> New Project** command to open the *New Project File* form.

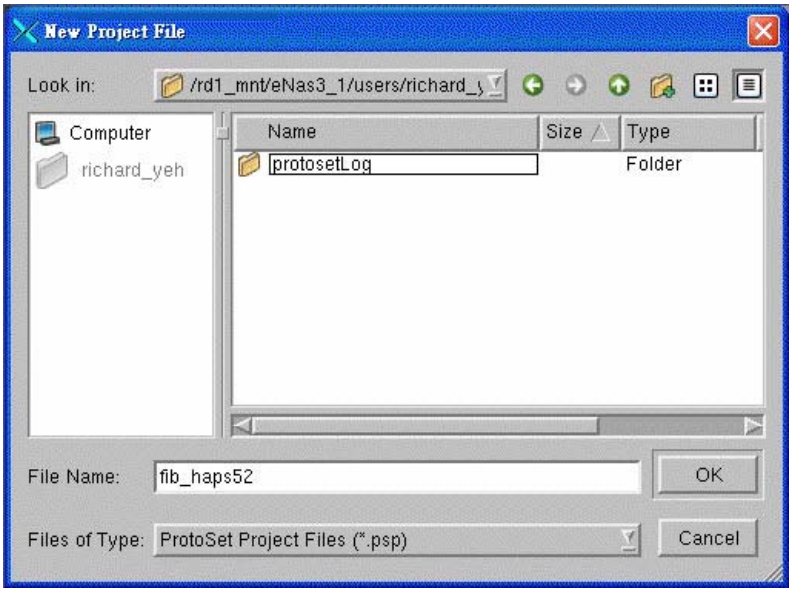

Figure: New Project File Form

3. In the **File Name** field, enter the project name as *fib_haps52*.
4. Click the **Open** button to create the new project.

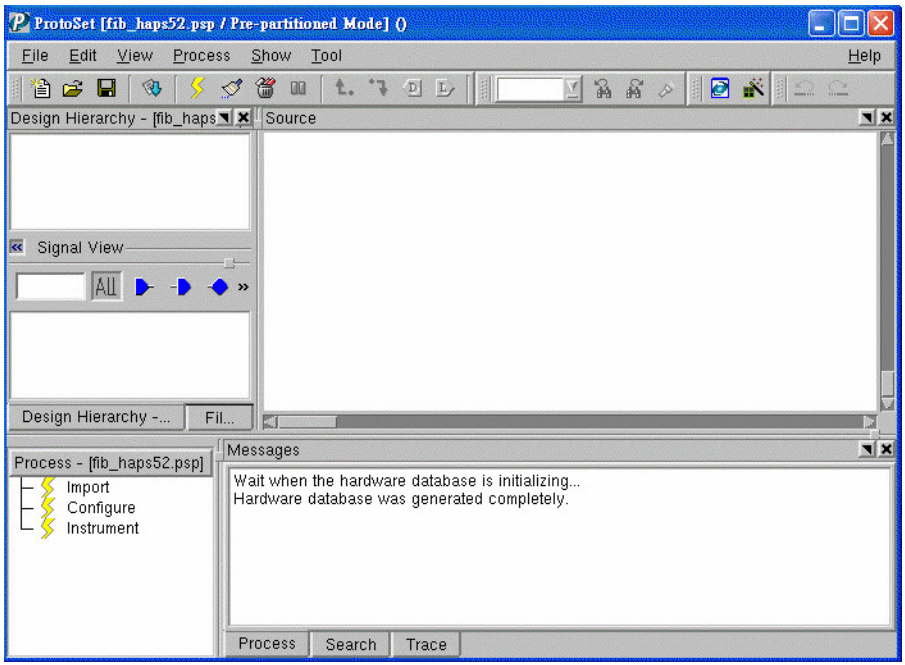

Figure: ProtoSet Window with Project Loaded

Import Design

1. In the *ProtoSet* window, select the **Process -> Import Design** command to open the *Import Design* form.

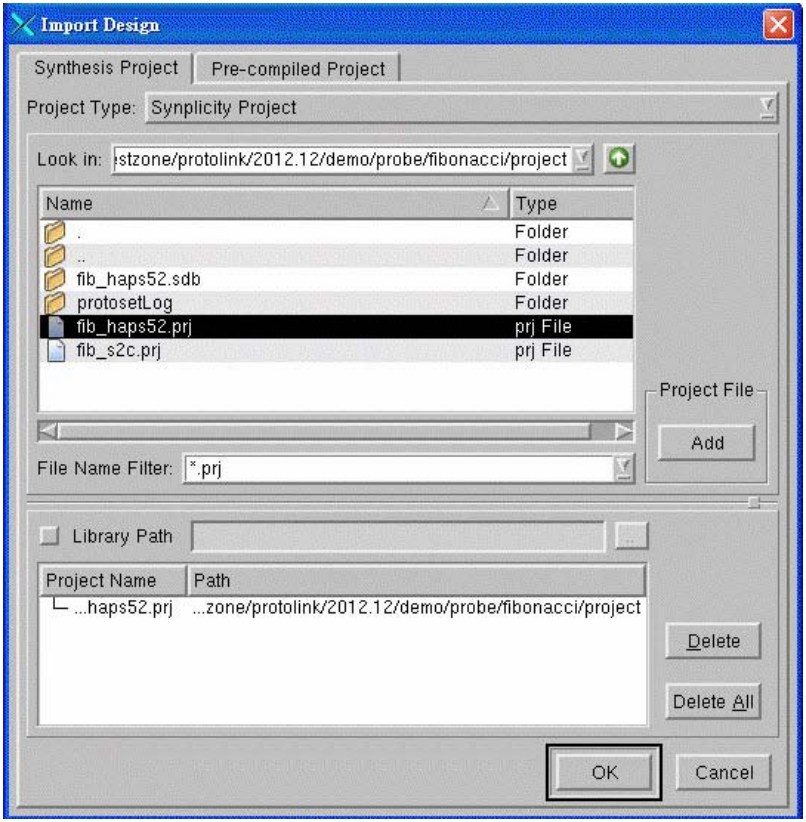

Figure: Import Design Form

2. In the lower pane, select the Synplify project file *fib_haps52.prj* and click the **Add** button.
3. Click the **OK** button to import the design from the Synplify project file.

Configure

1. Click the Configure process in the Process View to show the Configure View.
2. In the **Hardware** page, select the *HAPS52* platform from the pre-defined selection list.

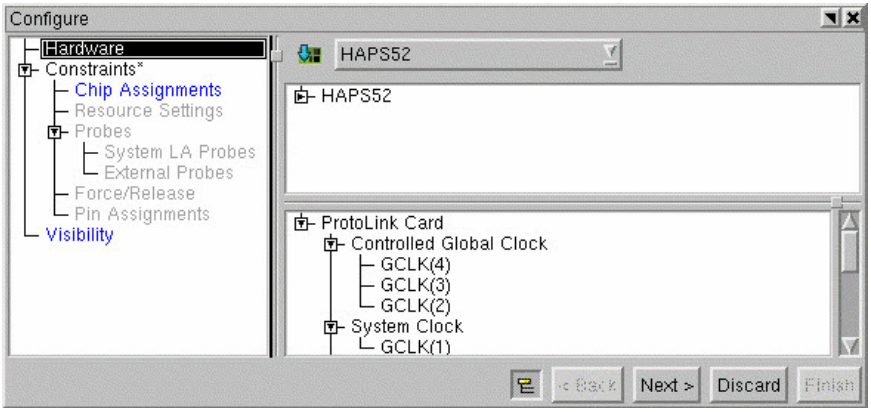

Figure: Hardware Page - Platform Selection

3. Click the **Next** button to switch to the **Chip Assignments** page.
4. Drag the *Fib_emu* module from the Design Hierarchy View and drop into the **Chip Assignments** page.
5. Double-click the **Assign To** field and select the *uHAPS52.uA* instance.

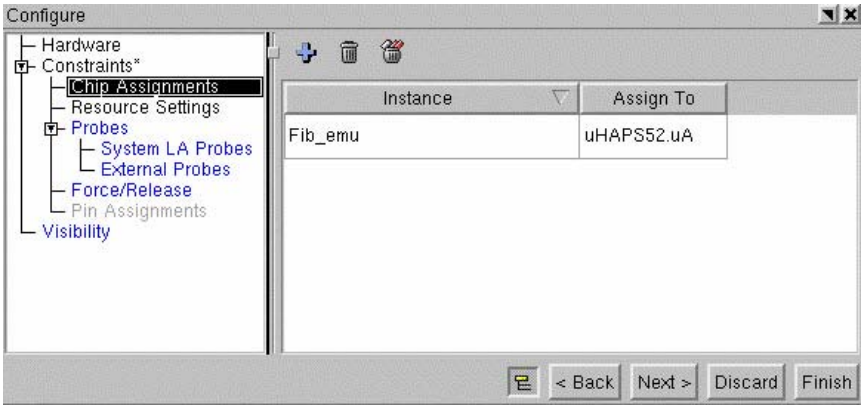

Figure: Chip Assignments Page

6. Click the **Next** button to switch to the **Resource Settings** page.

7. In the **Resource Settings** page, set the **Max. Probe Group** field to 3.

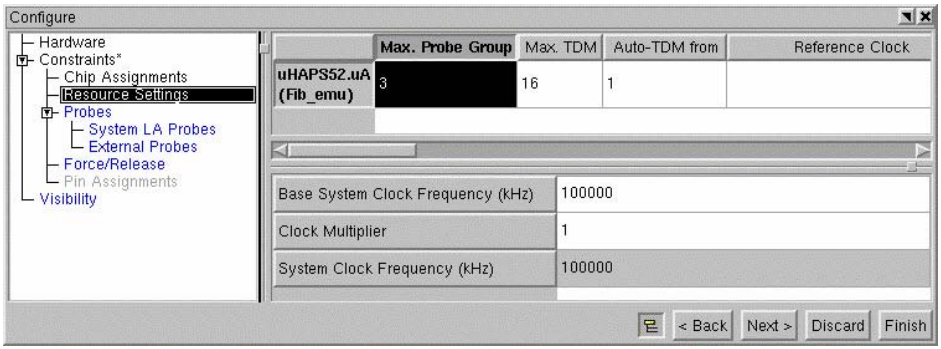

Figure: Resource Settings Page

8. Click the **Next** button to switch to the **System LA Probes** page.
9. Add probes by dragging the signals in the following figure from the Signal View and dropping into the **System LA Probes** page.

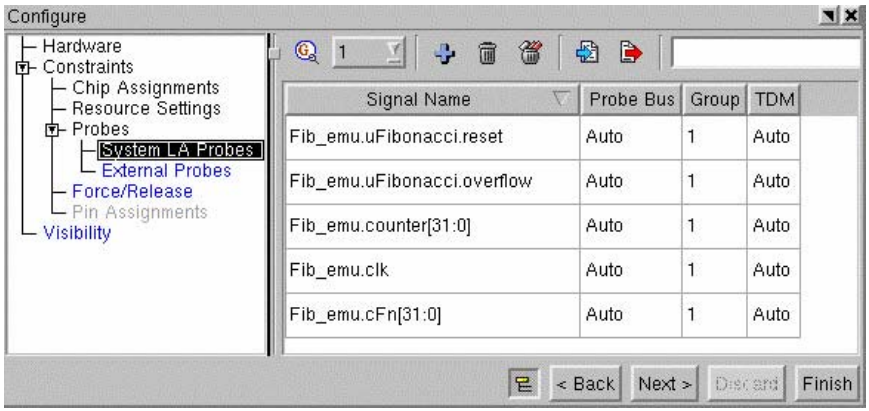

Figure: System LA Probes Page

10. Click the **Finish** button to complete the Configure process.
11. When prompted with the Resource Constraints file name, click the **Save** button to accept the default file name.

Instrument

1. Right-click on the Instrument process and select the **Options** command.
2. In the *Instrument Options* form, set the **Output Directory** to *Instrument.haps52* and click the **OK** button.
3. Right-click on the Instrument process and select the **Run** button.

   At the completion of the Instrument process, all processes are marked with a check icon.

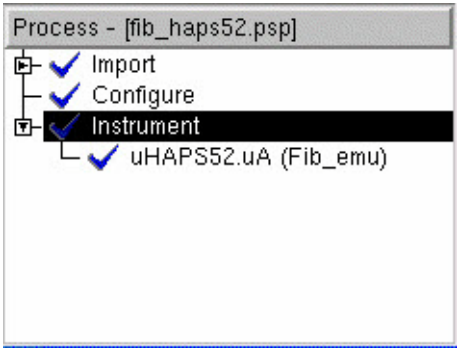

Figure: Instrument Process Finished

4. Select the **File -> Exit** command to exit the *ProtoSet* window.

Synthesize the Instrumented Design

Execute the following command to synthesize the instrumented design:

```
> make -f Makefile.haps52 synthesis
```

This command will invoke synplify_pro to synthesize the instrumented project file located at *Instrument.haps52/uHAPS52.uA/fib_haps52.prj*.

NOTE: Before executing the command, the tool path and license file for Synplify Pro must be set up.

Place and Route the Design

Execute the following command to place and route the design:

```
> make –f Makefile.haps52 pnr
```

This command will invoke Xilinx ISE tools (ngbuild, map, par, and bitgen) to implement the FPGA.

NOTE: Before executing the command, the tool path and license file for Xilinx ISE must be set up.

Converge

1. Execute the *converge* utility with the **-cft** option to generate a chip file template called *chip_file.haps52*, which is an input file to the *converge* utility.

   ```
   > converge -psp fib_haps52.psp -cft chip_file.haps52
   ```

   The generated chip file can be used without modification in this case because the synthesis and place and route directories match the default ones. The *chip_file.haps52* file can be reused for later converge invocations.

2. Execute the *converge* utility to generate the runtime database *fib_haps52.rdb*.

   ```
   > converge -psp fib_haps52.psp -cf chip_file.haps52
   ```

Runtime Flow

Hardware Preparation

This tutorial is based on the HAPS-52 prototype board and the default configuration is used. Refer to the *HAPS-52 Integration* application note for more information on the hardware configuration (contact local Synopsys technical support for the application note). To exercise Probe Visualizer runtime, it is required to properly connect the prototype board, the ProtoLink Interface Card, and the runtime machine.

Download FPGA Image

Program the FPGA ‘A’ with the bit file in the runtime database at *fib_haps52.rdb/PNR/uHAPS52.uA/fpga_1.bit*. Refer to section *Configuration* in the *HAPS Motherboard HAPS-52* manual for more information regarding downloading the design onto an FPGA.

Open Runtime Database

1. Open the *ProtoRun* window:

   ```
   > protorun -Probe
   ```

2. In the *ProtoRun* window, select the **File -> Open Project** command to open the *Open ProtoLink Runtime Database* form.

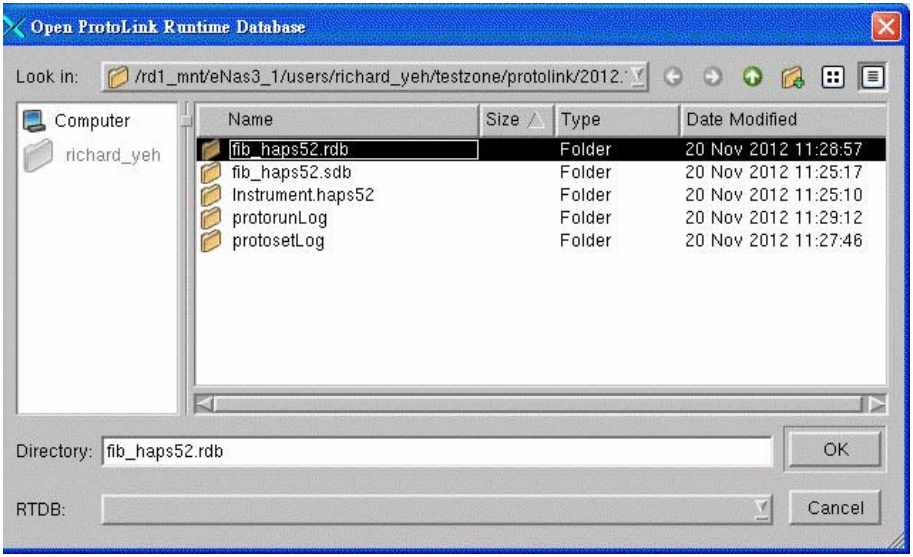

Figure: Open ProtoLink Runtime Database

3. In the *Open ProtoLink Runtime Database* form, specify *fib_haps52.rdb* in the **Directory** field and click the **Open** button.

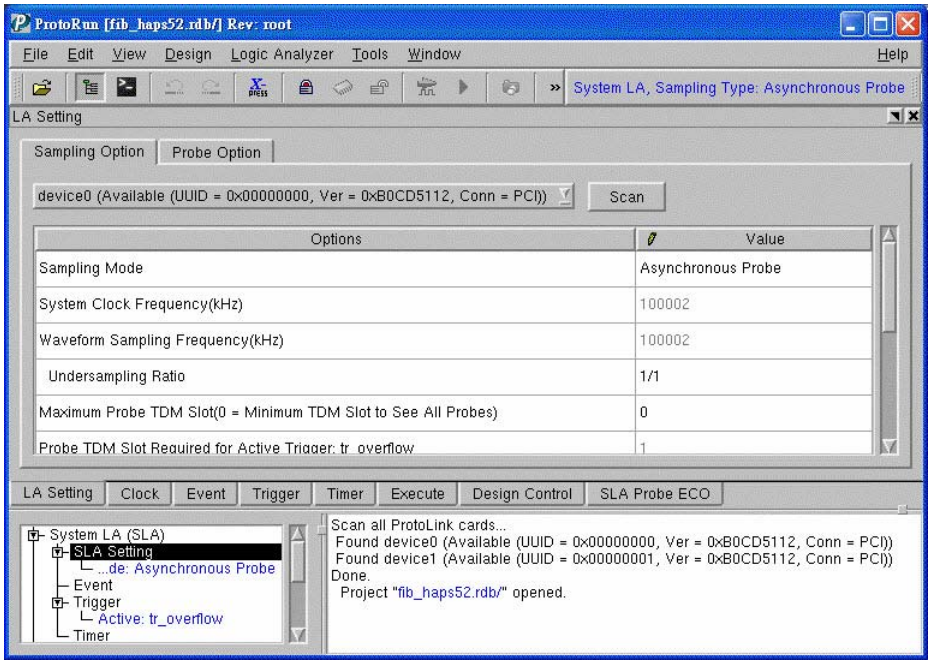

Figure: ProtoRun Window

Create a Simple Trigger

1. Select the **Event** tab.
2. Click the **New Event** button to create a new event. Double-click the event in the **Event List** field and change the name to *ev_overflow.*
3. Drag the *Fib_emu.uFibonacci.overflow* signal from the available probes in the **Net Probe** section and drop into the **Selected Probe** section.
4. Set the **Operator** field to ‘1’.

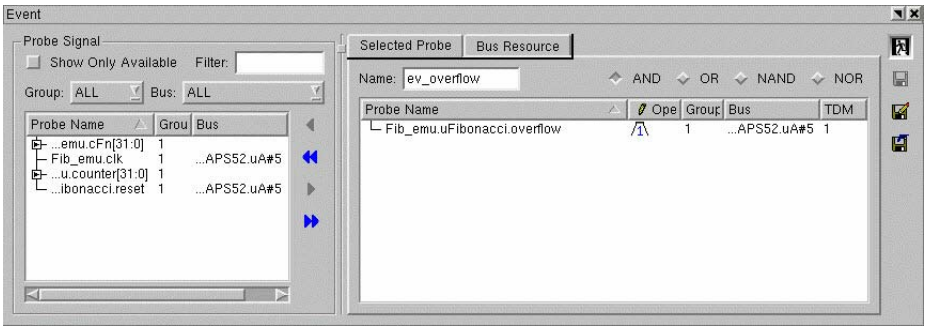

Figure: Create a New Event

5. Click the **Save As New Trigger** button to open the *Save As New Trigger* form.

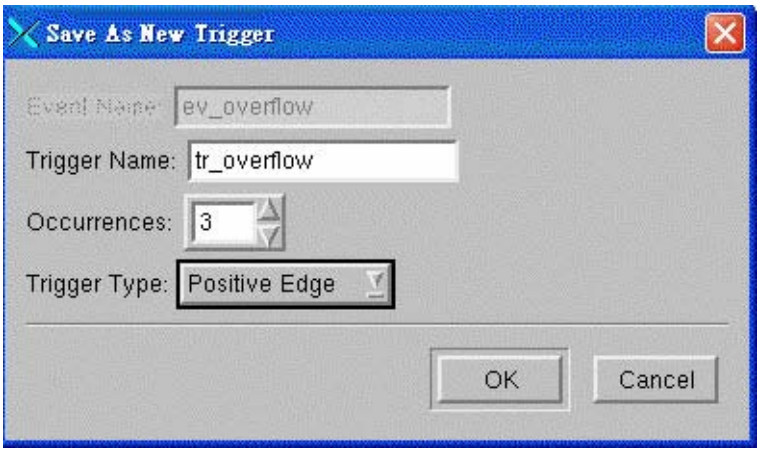

Figure: Save As New Trigger Form

6. In the *Save As New Trigger* form, set the **Trigger Name** to *tr_overflow*, the **Occurrences** to ‘3’, and the **Trigger Type** to **Positive Edge**.
7. Click the **OK** button to create the trigger.

Activate System LA and Capture Waveforms

1. Select the **Execute** tab.
2. Select the *tr_overflow* trigger in the **Trigger** field.

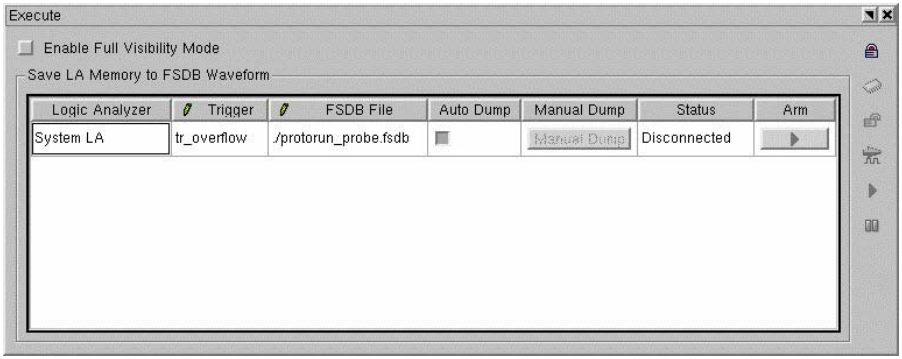

Figure: Execute Tab

3. Click the **Connect** button on the toolbar to connect to the ProtoLink Interface Card.
4. Click the **Configure** button on the toolbar.
5. Click the **Arm** button on the toolbar to activate the System LA.
6. Click the **Start Clock** button on the toolbar to start the design clock.

   When the specified trigger is reached, the Message Console will display the following message:

   ```
   Event trigger "tr_overflow" breakpoint reached.
   ```

7. Click the **Disconnect** button on the toolbar to disconnect from the ProtoLink Interface Card.
8. In the *ProtoRun* window, select the **Tools -> New Waveform** command to examine the captured waveform.

   In the *nWave* window, the trigger point is marked at the 3rd positive edge of the *Fib_emu.uFibonacci.overflow* signal.

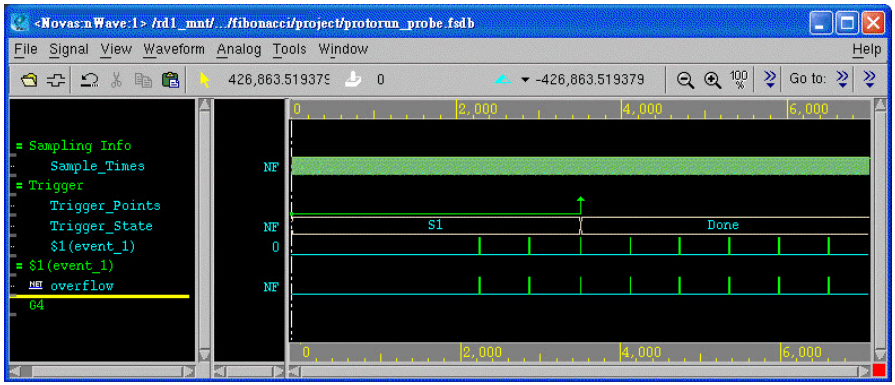

Figure: Trigger Point

9. In the *nWave* window, select the **Signal -> Get All Signals** command to view all probe signals.

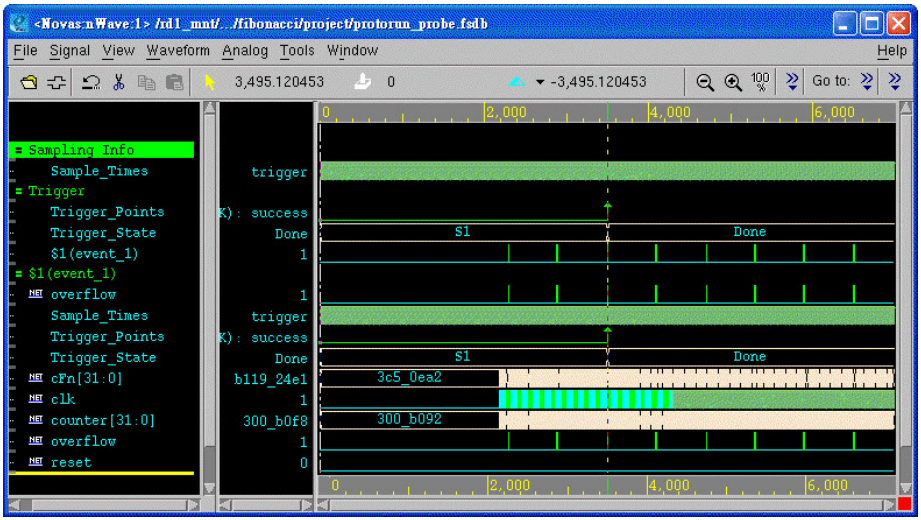

Figure: All Probe Signals

Add New Probes with SLA Probe ECO

1. Select the **SLA Probe ECO** page.
2. Add the following new probes to the **Modified Probe** tab:

```
Fib_emu.uFibonacci.vFn_1[0]
Fib_emu.uFibonacci.vFn_1[1]
Fib_emu.uFibonacci.vFn_1[2]
```

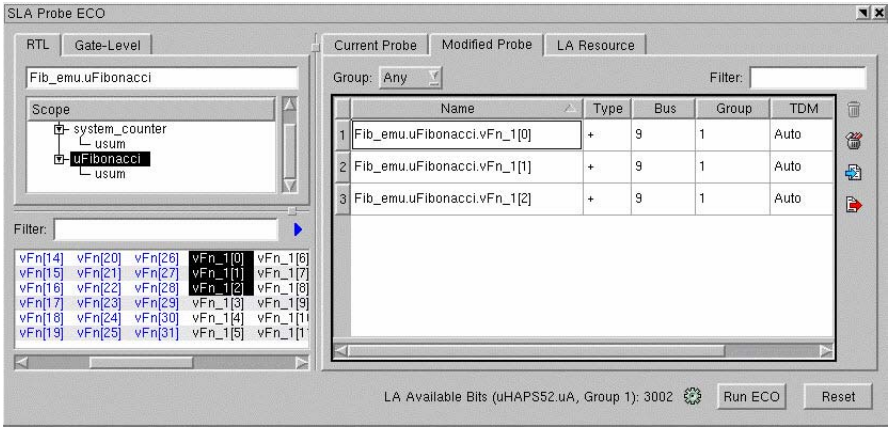

Figure: SLA Probe ECO Page - Modified Probe Tab

3. Click the **Run ECO** button to start the Probe ECO process. This may take a couple of minutes to complete.
4. Program the FPGA with the resulting FPGA image located at *fib_haps52.rdb/root/PNR/uHAPS52.uA/fpga_1.bit*. Refer to the *Download FPGA Image* section for details.
5. Click the **Connect** button on the toolbar to connect to the ProtoLink Interface Card.
6. Click the **Configure** button on the toolbar.
7. Click the **Arm** button on the toolbar to activate the System LA.
8. Click the **Start Clock** button on the toolbar to start the design clock.

   When the specified trigger is reached, the Message Console will display the following message:

   ```
   Event trigger "tr_overflow" breakpoint reached.
   ```

9. Click the **Disconnect** button on the toolbar to disconnect from the ProtoLink Interface Card.
10. In the *ProtoRun* window, select the **Tools -> Invoke Waveform** command to examine the captured waveform.
11. In the *nWave* window, select the **Signal -> Get All Signals** command to view all probe signals.

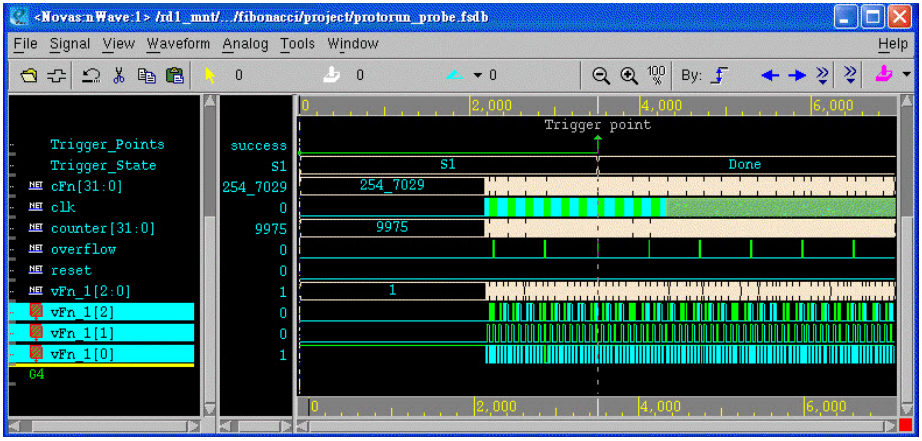

Figure: ECO Waveform