

---

# Demo Guide

Publication number 16600-97003  
March 1998

For Safety and Warranty information, see the pages behind the index.

© Copyright Hewlett-Packard Company 1998  
All Rights Reserved

---

# HP 16600A-Series Logic Analysis System

---

# The HP 16600A-Series Logic Analysis System Reduces Time to Insight

With the HP 16600A-series logic analysis system:

- You can view target system behavior at many different levels of the design hierarchy: from analog signals (using an oscilloscope module), to the timing relationships of signals, to microprocessor execution, to source code, to bus execution, to overall system performance. Being able to look at a problem from many perspectives helps you gain insight into problems faster.
- You can time-correlate different views of target system behavior (and use multiple views) to analyze the same target system event in different ways.
- You can use a pattern generator module to provide stimulus to parts of the target system when testing hypotheses or analyzing the target system's response to specific inputs.
- You can use an emulation module to control microprocessor execution (run, stop, step, breakpoints) and display and modify the contents of microprocessor registers and memory.
- You can explore data collected from your target system, perhaps data that captures a rarely occurring problem, in a more efficient way by taking deep-memory traces and by using post-processing filtering tools. You can also trigger on a specific sequence of events and store only the data of interest, and perhaps the context around that data, to capture execution over a longer time.
- You can export critical data from your target system, import it later, and compare it to new data.
- You can share the networked logic analysis system among project team members to facilitate communication and collaborative debugging.
- You can solve problems quickly and move on to the next problem, speeding up your overall development time.

These capabilities let you identify problems and track them to their root cause. They let you explain all the symptoms of a problem and give you confidence in your solution.

---

## In This Book

This demo guide shows many of the things you can do with the HP 16600A-series logic analysis system. It's part of a demo kit that includes a MPC860 microprocessor-based system.

You can probe the MPC860 demo board by connecting the logic analyzer modules and emulation modules. Later you will connect an oscilloscope module. Then, you can follow the instructions in this guide to capture and analyze MPC860 demo board execution. You will see some of the more powerful ways you can use the logic analysis system to debug and verify your own target systems.

This demo guide shows:

- You can quickly set up the logic analysis system to capture hundreds of waveforms. You can also use the Setup Assistant to quickly trace microprocessor execution, inverse assemble the microprocessor trace, perform run control functions, and correlate source code.
- You can quickly find hardware and software interaction problems by correlating views of the captured data with traces of microprocessor execution, and by analyzing system performance.
- You can quickly find the cause of difficult hardware problems using HP's deep-memory logic analyzers, high-speed logic analyzers, and pattern generators.

This demo guide also contains an appendix that describes the MPC860 demo board and its firmware in more detail and state and timing.

---

# Contents

## **The HP 16600A-Series Logic Analysis System Reduces Time to Insight**

### **In This Book**

#### **1 Getting Started**

- Connecting the demo board to analyzer 10
- Connecting the demo board to the emulation module 12

#### **2 Quickly Set Up the Analysis System**

- Tracing Hundreds of Your Target's Signals 14
- Connecting the analyzer to your target 15
- Using an HP logic analysis module 16
  
- Tracing Processor Code Execution with Source Code  
Correlation 22
- HP's Processor Solutions 22
- Using the Setup Assistant 27

#### **3 Quickly Find the Cause of Difficult HW/SW Interaction Problems**

- Looking at Correlated Hardware/Software Traces 42
- Correlating processor execution with external buses 43
- Tracking hardware problems to their software causes 52
- Tracking software problems to their hardware causes 61
  
- Looking at Firmware Driver Issues 70
- Controlling and modifying processor execution 71
- Downloading code to RAM or Flash ROM 80

---

# Contents

Looking at Software Issues	83
Analyzing system performance	84
Using context store	91
Tracking processor execution with caches turned on	99

## **4 Quickly Find the Cause of Difficult Hardware Problems**

Capturing Very Deep Traces	106
Using logic analyzers with deep memory	107

## **A About the MPC860 Demo Board**

Demo Board Hardware	114
Introduction	114
Configuring the Logic Analysis System for the Demo Board	114
Demo Board Connector Mapping	118
Demo Board Features	122

Demo Board Firmware	127
Introduction	127
Overview of main()	127
Overview of proc_specific	130
Variables	131
Using the PowerPC 860 Emulation Module	134

Recommended Demo Configuration	135
Why use Recommended Demo Configuration	135

## **B Concepts**

Timing Analysis vs. State Analysis in Logic Analyzers	138
-------------------------------------------------------	-----

---

# Contents

**Glossary**

**Index**

---

# Contents



---

## Getting Started

---

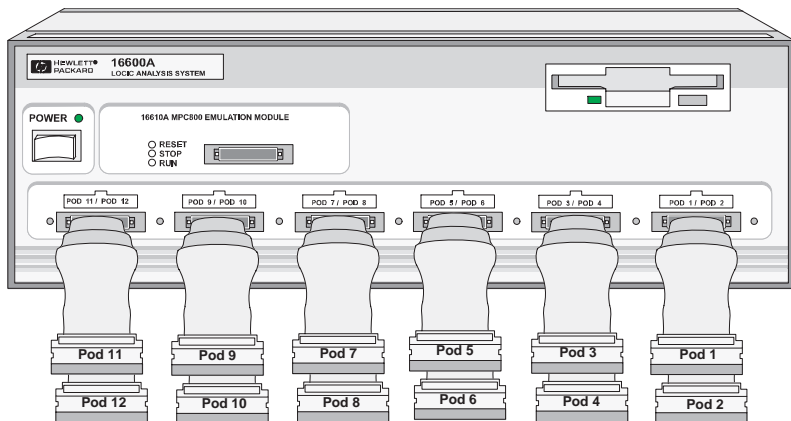
## Connecting the demo board to analyzer

The following provides instructions for connecting the demo board to the logic analyzer. The instructions, as well as the entire guide, assume that your analysis system has the recommended configuration of modules.

Go to “Recommended Demo Configuration” on page 135 for instructions on verifying your configuration.

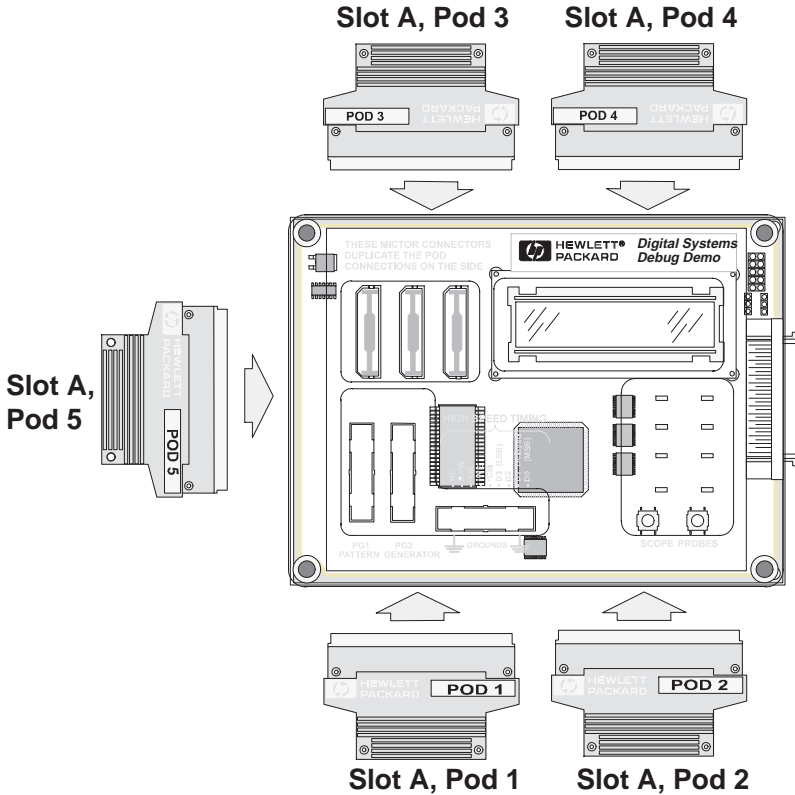
If the logic analysis system does not have the recommended configuration, you can still connect the demo board by following the instructions provided by the Setup Assistant (go to “Using the Setup Assistant” on page 27). However, some of the exercise may not work properly; this will depend on the particular configuration you use.

First connect the logic analysis modules. The recommended demo configuration has 12 pods of logic analysis. The first 5 pods are used to capture MPC860 demo board execution.



Connect the logic analyzer pods to the demo board connectors as described in the following table:

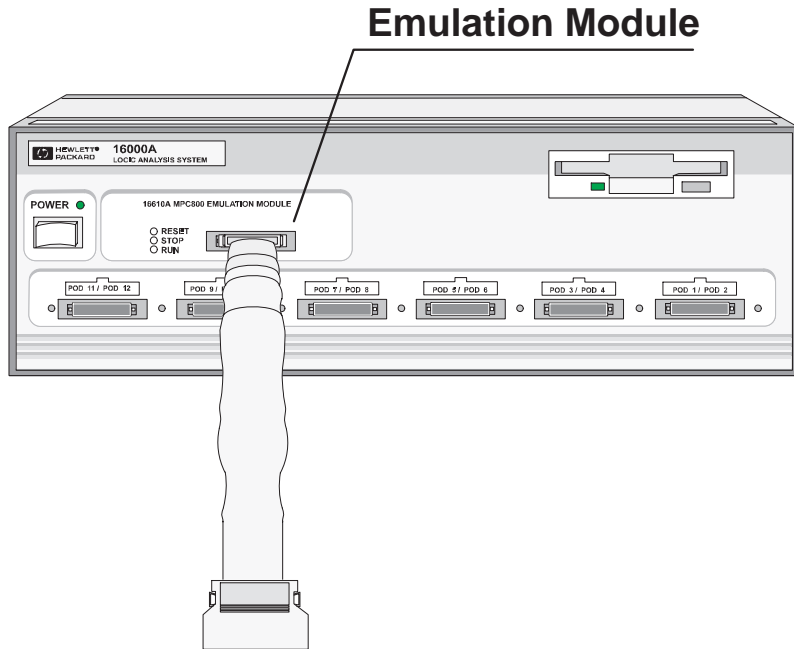
Demo board	Logic Analyzer
Pod 1	Slot A, Pod 1
Pod 2	Slot A, Pod 2
Pod 3	Slot A, Pod 3
Pod 4	Slot A, Pod 4
Pod 5	Slot A, Pod 5



---

## Connecting the demo board to the emulation module

Now, connect the emulation module.



If the emulation module cable is not connected to the emulation module, do so now. Connect the other end of the cable to the demo board. It goes on the edge with no logic analyzer cables connected to it.

---

## Quickly Set Up the Analysis System

## Tracing Hundreds of Your Target's Signals

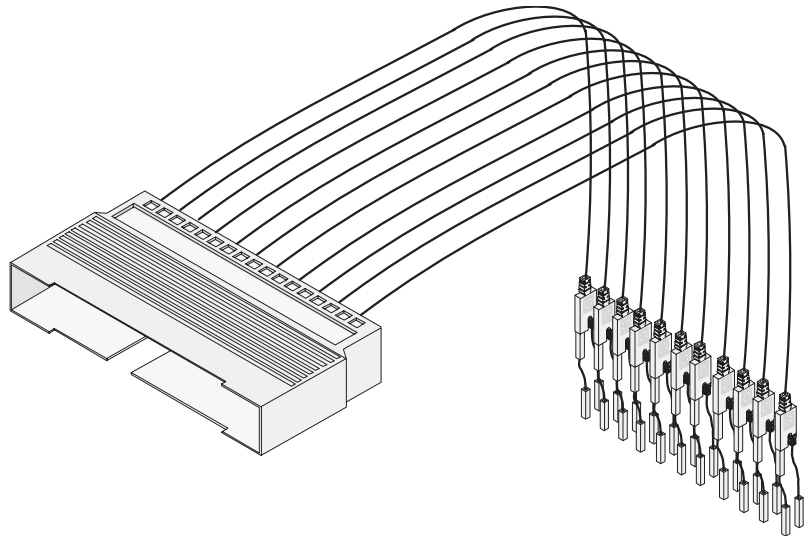
Being able to capture and display a large number of logic analyzer channels helps you look at more target system execution at one time.

- By using a logic analyzer's flying lead set, you can probe and look at digital signals in any part of a target system.
- By loading the HP 16600A-series logic analysis system with logic analyzer cards and using the logic analysis system's display capabilities, you can easily view and manage a large number of waveforms.

---

## Connecting the analyzer to your target

The most common way to probe a target system is with a flying lead set. The flying lead sets connect to logic analyzer pods to provide 16 individual data connections and one clock connection. HP provides a variety of clips and connections that attach to the flying leads and make it easier to attach to fine-pitch leads.



Two other methods for attaching the analyzer to your target system are discussed at the beginning of the next exercise.

## Using an HP logic analysis module

HP has a variety of logic analysis modules for tracing your target system's signals. They range in channel count, trace depth, and acquisition speed. This exercise uses the HP 16600A logic analysis module. However, it pretty much applies to all HP analysis modules. For an exercise specific to deep trace, go to "Capturing Very Deep Traces" on page 106.

In this exercise, you will see how the large windows and features of the HP 16600A-series logic analysis system make it easy to manage a large number of waveforms. The HP 16600A-series logic analysis systems can be configured with as many as 1,020 channels of analysis.

### 1 Probe the MPC860 demo board.

Follow the instructions in the "Getting Started" chapter on page 9 for connecting the logic analysis module to the demo board.

---

**NOTE:**

The emulation module connection should NOT be made; it could put the processor into reset, preventing it from running, which is necessary for this exercise.

---

### 2 Start with the default configuration.

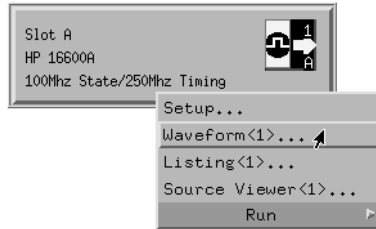
To get a default configuration, go to the "16600A Logic Analysis System" window, select the "Exit" button in the lower right-hand corner, and click "OK" in the dialog that comes up.

When the session has ended, go to the "Session Manager" window, and select "Start Session on This Display".

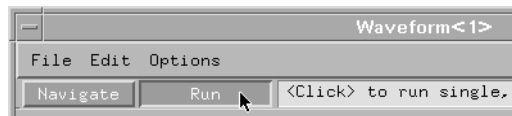


### 3 Set up trigger, and run measurement

When the “16600A Logic Analysis System” window has come up, select the “HP16600A” button on the left-hand side and select “Waveform<1>...” from the popup menu.



Click the green “Run” button to take a trace of the lower 16 bits of the address bus.

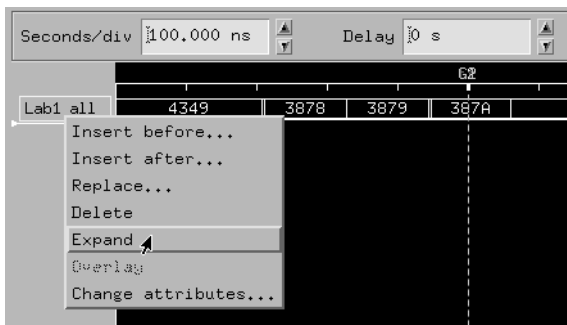


### 4 Display the captured data.

Maximize the “Waveform<1>” window.

You are looking at an overlaid trace of 16 bits of the address bus; in other words, each of the individual traces of the address bus bits are displayed in this one trace. You can see the hexadecimal value of the address bus where there is space to display it.

To see each individual address line, right-click on “Lab1 all” and select “Expand”. Notice that each “Lab1” is now individually numbered.

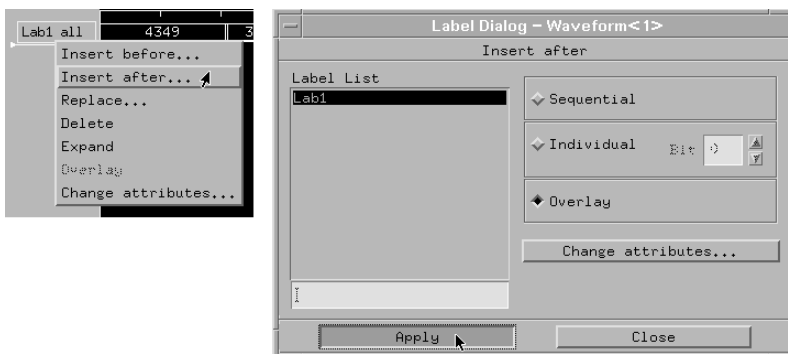


To overlay the lines again, right-click on one of the “Lab1” signals, and select “Overlay”.

**5** Replicate the one occurrence of “Lab1 all” several times.

To get to the point of this exercise, managing a large number of waveforms, the Lab1 trace will be replicated multiple times.

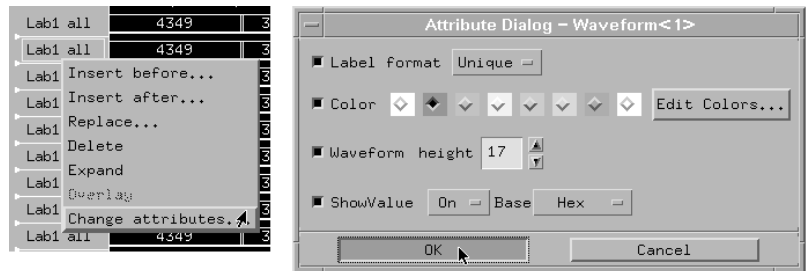
Right-click on “Lab1 all”, select “Insert after...”, and a label dialog comes up. This dialog lets you add more labels to the display, either one bit at a time or as overlaid signal sets. The default is overlaid. Add 9 more of the overlaid “Lab1 all” to the Waveform<1> window by clicking “Apply” 9 times; then, close the dialog.



## 6 Add color to waveforms.

We now have a total of 160 waveforms displayed. To help identify them more easily, you can add some color.

Pick the second “Lab1 all” from the top, right-click on it, and select “Change attributes...”. Select the red radio button to the right of “Color” and click “OK”.



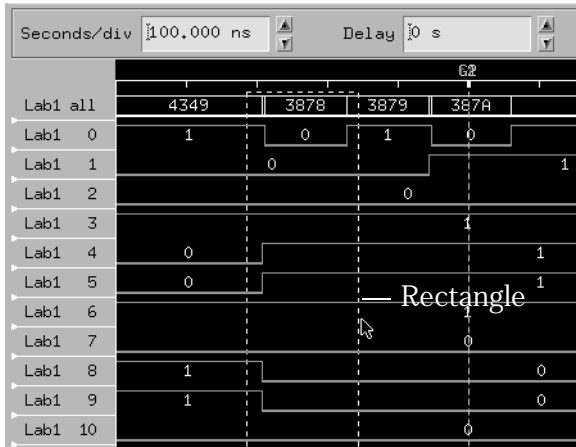
Try changing some of the other “Lab1 all”.

Now, expand one of the colored “Lab1 all” and see how the coloring helps you to follow the timing waveforms.

## 7 Zoom in on the information of interest.

Pick an area in the waveform trace that you want to look at more closely. Left-click inside the black waveform display area to the left of the area of interest to you. Drag the mouse to the right side of the area you are interested in. This will display a rectangle encompassing the area that the display will be expanded to.

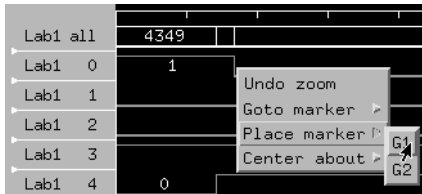
## Chapter 2: Quickly Set Up the Analysis System Tracing Hundreds of Your Target's Signals



When you let go the mouse button, the display will expand horizontally so you can better see the area you are interested in.

- 8 Use timing markers to establish the timing relationship between edges in the waveforms displayed.

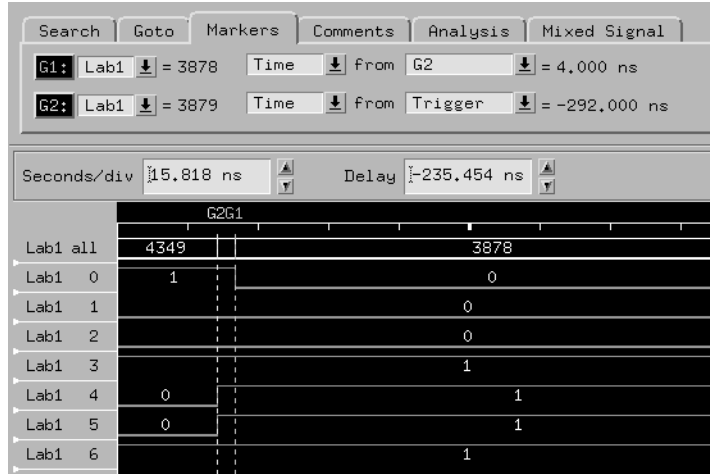
Right-click on an edge that you are interested in, select “Place Marker >”, and select “G1”.



Right-click on another edge that you would like timing information about relative to the G1 marker, and select “G2”.

Select the “Markers” tab and select the list arrow next to the text window for the G1 marker that says “Trigger”.

Select G2 from the list. What you see displayed to the right of the G1 marker line is the time between the G1 and G2 markers.



If you would like to learn more about the analysis system's search capabilities, go to "Capturing Very Deep Traces" on page 106.

### Summary

By using the logic analysis system's large display capabilities, you can easily view and manage a large number of waveforms.

## Tracing Processor Code Execution with Source Code Correlation

### HP's Processor Solutions

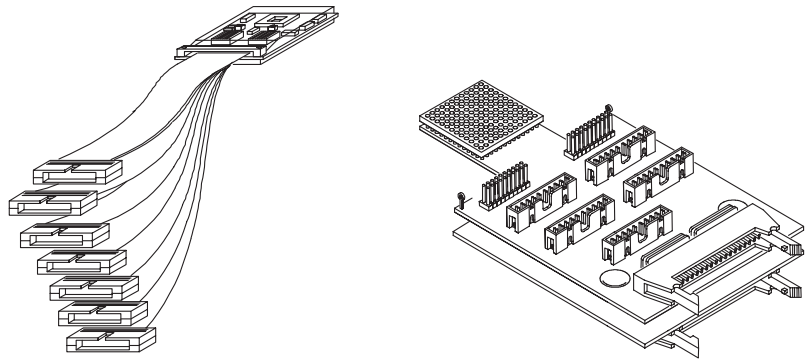
Probing target system circuits can be difficult. The narrow spacing of surface mount package pins and the dozens of connections that a microprocessor requires might make the task seem almost impossible.

However, HP and its channel partners provide products that make probing surface mount packages and microprocessors easier. HP also provides products like emulation modules and the source correlation tool set that make debugging microprocessor execution easier.

### Analysis Probes

HP and its channel partners provide *analysis probes* for probing microprocessors and standard buses. Analysis probes are available for over 200 microprocessors and standard buses.

Analysis probes provide the mechanical connection, electrical connection, active circuitry (when necessary), and the software required to trace and inverse assemble microprocessor execution. The following demo, "Using an HP logic analysis module" on page 16, will give you a good feel for the benefits provided by analysis probe software.



Analysis probes plug into Pin Grid Array (PGA) sockets, bus connectors, and even clamp over Thin Quad Flat Pack (TQFP) packages and connect to Ball Grid Array (BGA) footprints. They bring dozens of connections out to logic analyzer pods. Analysis probes are typically low profile compact boards with minimum capacitive loading.

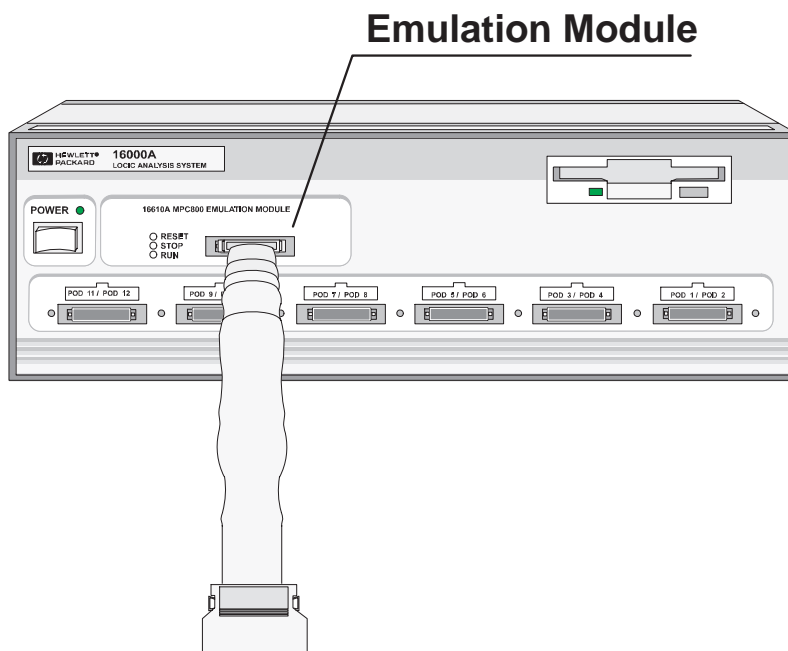
**Designing Connections into Your Target System.** When probing microprocessor cores embedded in ASICs or when analysis probes cannot be used for some other reason, you can design logic analyzer connections into your target system and purchase an inverse assembler for your processor separately.

HP provides information on designing several types of logic analyzer connections into target systems (which vary in cost and connection density).

Connectors can range from the 0.1 inch 2x20 connectors (like the five around the edge of the MPC860 demo board) to the high-density Mictor38 connectors (like the three on top of the demo board). The Mictor38 connectors provide connections for two logic analyzer pods each, while the 2x20 connectors provide connections for one logic analyzer pod. The Mictor38 connectors require that you use the HP E5346A high-density termination adapters.

## Emulation Modules

The HP 16600A-series logic analysis system can contain *emulation modules* that use a processor's Background Debug Mode (BDM) or JTAG port to control the processor. You can run a microprocessor, stop it, set breakpoints, modify the contents of microprocessor registers and memory locations, and download code to RAM and Flash ROM.



HP's emulation modules require a Target Interface Module (TIM) to connect to the processor's BDM or JTAG port (see picture). A different TIM is used for each processor or processor family to adapt the emulation module's connections to the processor's. The 860 demo board that you will be using in the following demo exercises does not need a TIM because we have built it into the board.

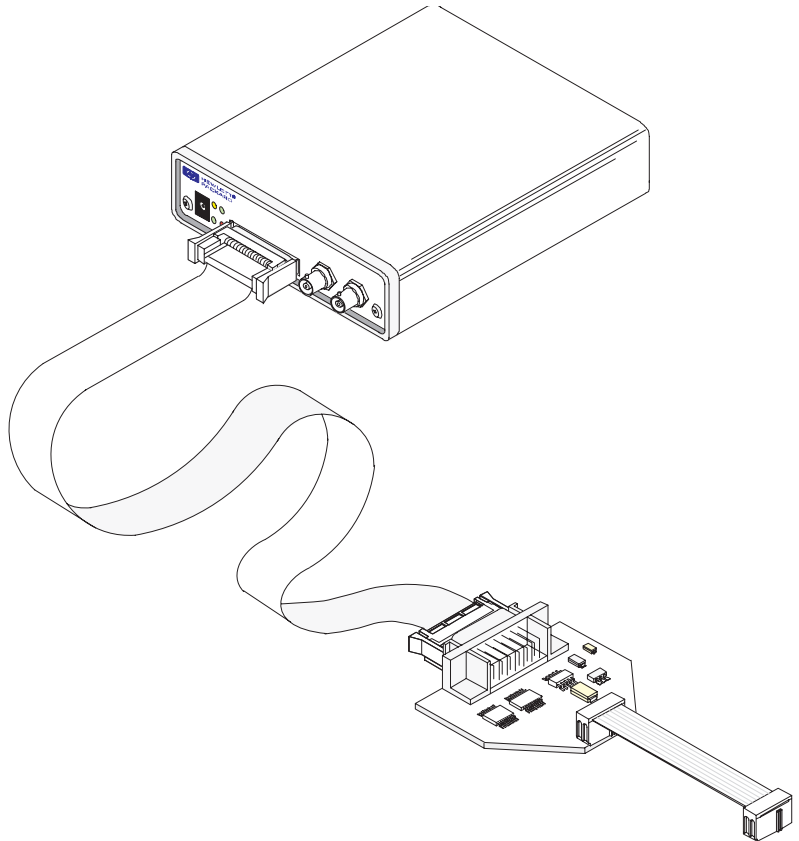


## Emulation Probes

Emulation probes are just stand-alone emulation modules. They, combined with a commercial debugger, provide an economical run control solution.

Emulation probes can also be accessed from the HP 16600A-series logic analysis system to provide run control for more than two processors.

Below is a drawing of an emulation probe with a TIM attached.



## **Source Correlation Tool Set**

The *source correlation tool set* add-on for the HP 16600A-series logic analysis system lets you view the source code that corresponds to data captured on the microprocessor bus.

The source correlation tool set requires that symbol information be loaded into the logic analyzer from the target system program's object file.

## **Processor Solution Packages**

You can order HP *processor solution packages* that combine an analysis probe, an emulation probe, and the source correlation tool set for a particular microprocessor.

## **Processor Solution Information on the Web**

You can find up-to-date processor solution information on the world-wide web at:

<http://www.hp.com/go/uPsolutions>

Or, contact your HP sales representative.

## Using the Setup Assistant

The HP 16600A-series logic analysis system includes a *setup assistant* to help you configure the logic analyzer for a particular analysis probe. It also configures the emulation module for the selected processor and helps you read in your symbol file for the inverse assembly and Source Viewer.

The setup assistant analyzes the configuration of your logic analyzer and the type of microprocessor you want to trace. Then, it asks what options you want implemented. The setup assistant tells you how to connect the analyzer probes, and it creates the necessary configurations.

You only need to run the setup assistant when you start working with a new processor or when you change the configuration of the logic analyzer. Once the setup assistant has created a configuration, it can be saved and reloaded.

The following steps show you how to use the setup assistant to trace microprocessor execution on the MPC860 demo board and view the source code associated with captured data.

### 1 Start with the default configuration.

To get a default configuration, go to the “16600A Logic Analysis System” window, select the “Exit” button in the lower right-hand corner, and click “OK” in the dialog that comes up.

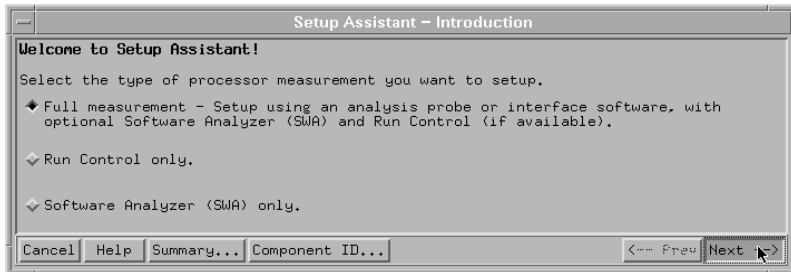
When the session has ended, go to the “Session Manager” window, and select “Start Session on This Display”.

### 2 Start the setup assistant.

Go to the “16600 Logic Analysis System” dialog and select “Setup Assistant” from the bottom buttons.



The “Setup Assistant – Introduction” dialog will be launched. Select “Full measurement - ...” and then select “Next -->” to go to the next dialog.



### 3 Identify your microprocessor

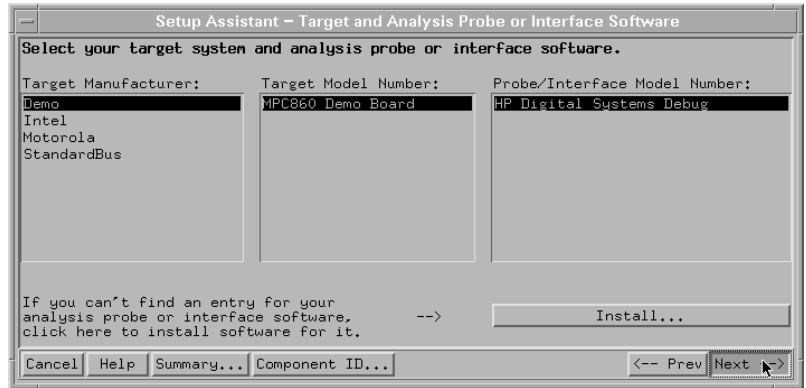
The “Setup Assistant – Target and Analysis Probe or Interface Software” dialog now comes up. This is where you tell the setup assistant what processor you are using and whether you are using an analysis probe or connecting directly to your target.

#### **Microprocessor Support**

Processors are added on a regular basis, so if you do not see the one you need, check with your HP sales representative to see if it has become available.

Tell the setup assistant you are using the MPC860 demo board (which has built-in connections for the logic analyzer).

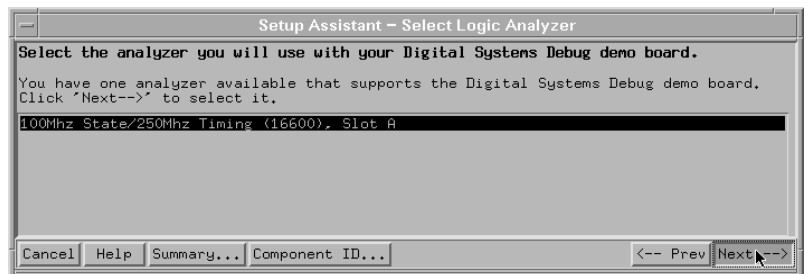
Select “Demo” from the list which expands to “MPC860 Demo Board” and “HP Digital Systems Debug”, then go to the next dialog.



#### 4 Select the logic analyzer.

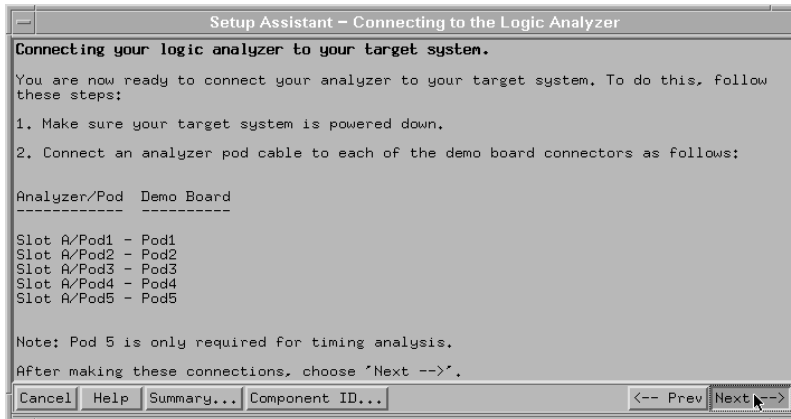
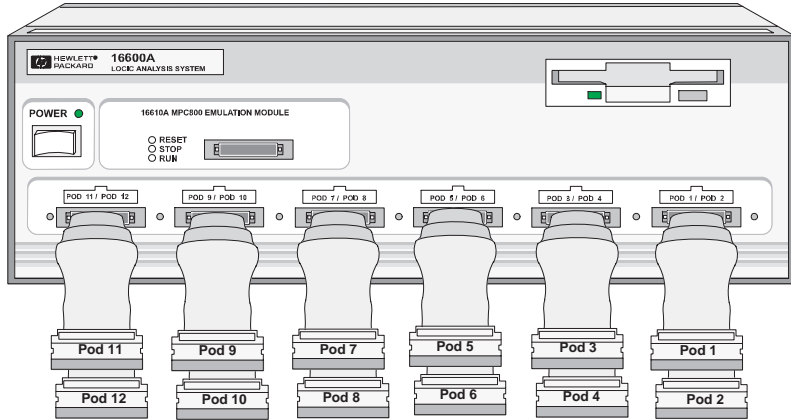
The “Setup Assistant – Select Logic Analyzer” dialog is now up. This dialog presents you with a list of analyzer modules installed in your system that are suitable for the analysis probe you are using.

If you have the standard demo set of modules in your system, only one analyzer is listed. Select the logic analyzer, and go to the next dialog.



## 5 Connect the demo board to the logic analyzer.

This brings up the “Setup Assistant – Connecting to the Logic Analyzer” dialog. This dialog tells you how to connect the analyzer module to the demo board.



When you are done connecting the logic analyzer to the demo board, select “Next -->”.

The setup assistant now loads the proper configuration for the demo board, and the proper inverse assembler, into the HP 16600A analysis module. Click “OK” to clear the

informational dialogs that pop up during this part of the exercise.

---

**NOTE:** Don't connect the pattern generator just yet. The default pattern generator signal level outputs will turn off the MPC860 demo board's LCD display.

---

**6** Set the inverse assembler preferences and filter for the demo board.

---

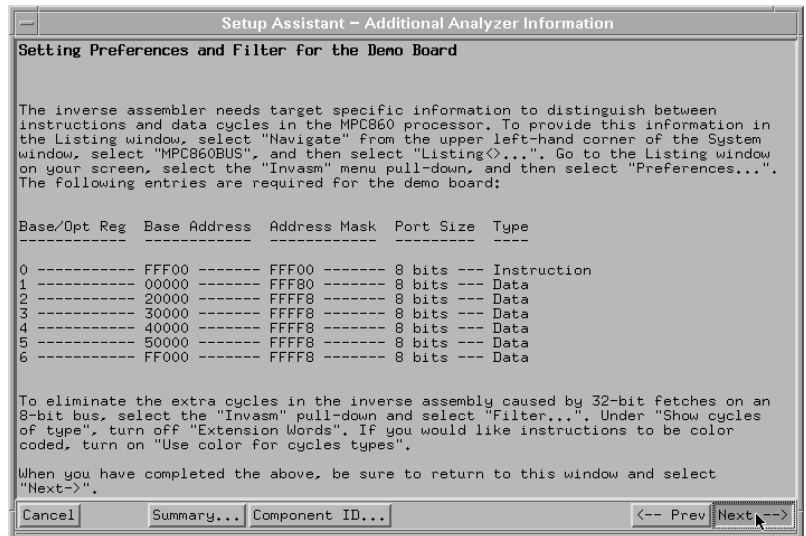
**NOTE:** It is important that you follow the instructions in the "Setup Assistant - Additional Analyzer Information" dialog. This is not optional.

---

Follow the instructions in the "Setup Assistant - Additional Analyzer Information" dialog for entering the demo board memory map and for setting up the filter.

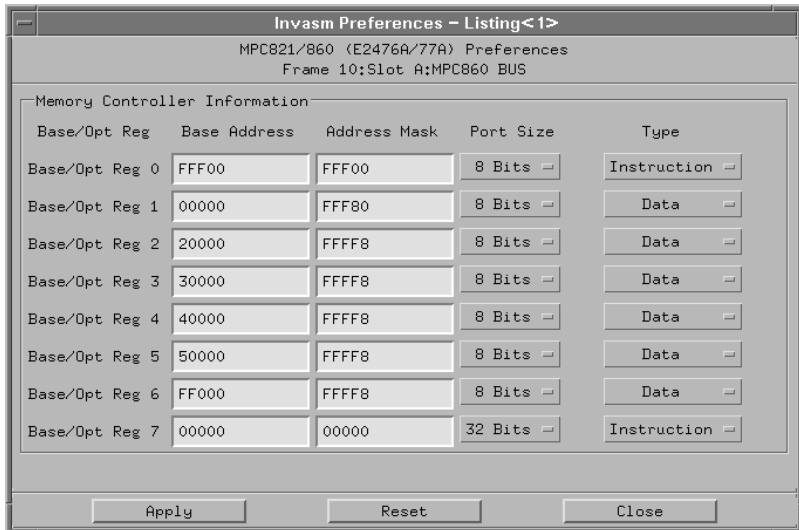
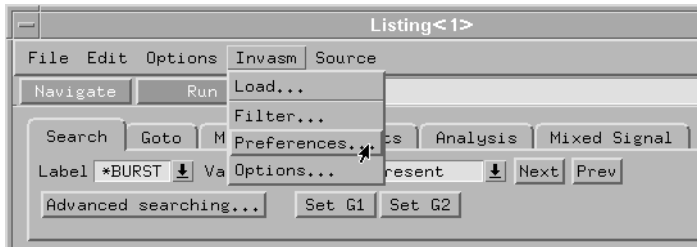
Note that the MPC860 processor does not provide status information that distinguishes between data and instruction fetches. Therefore, you must provide that information.

After you have set the inverse assembler preferences and filter for the demo board, go to the next dialog.

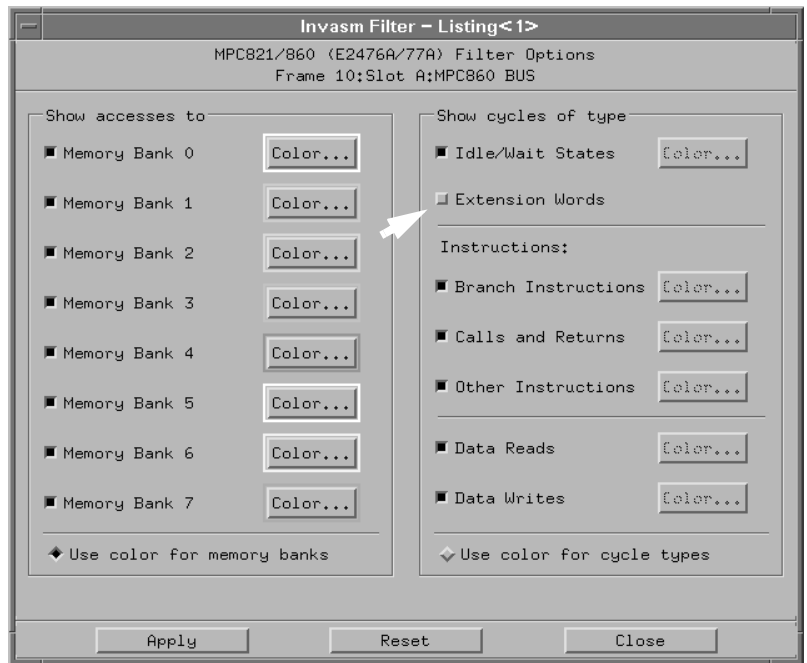


## Chapter 2: Quickly Set Up the Analysis System

### Tracing Processor Code Execution with Source Code Correlation





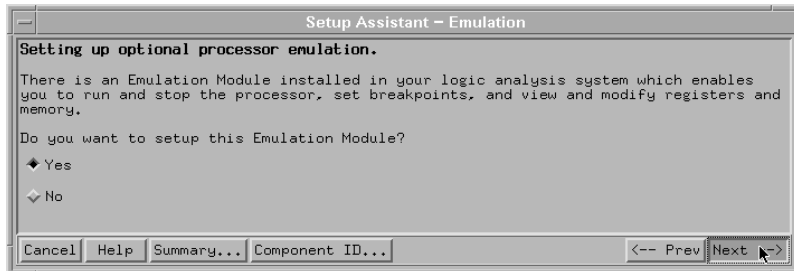


## 7 Connect the demo board to the emulation module.

The “Setup Assistant – Emulation” dialog comes up at this point.

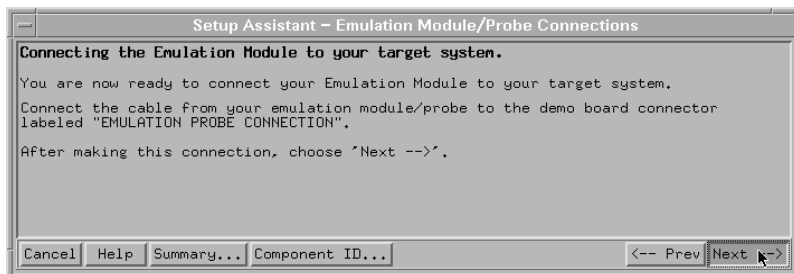
If you have the standard demo set of modules, the setup assistant will detect an emulation module in your system.

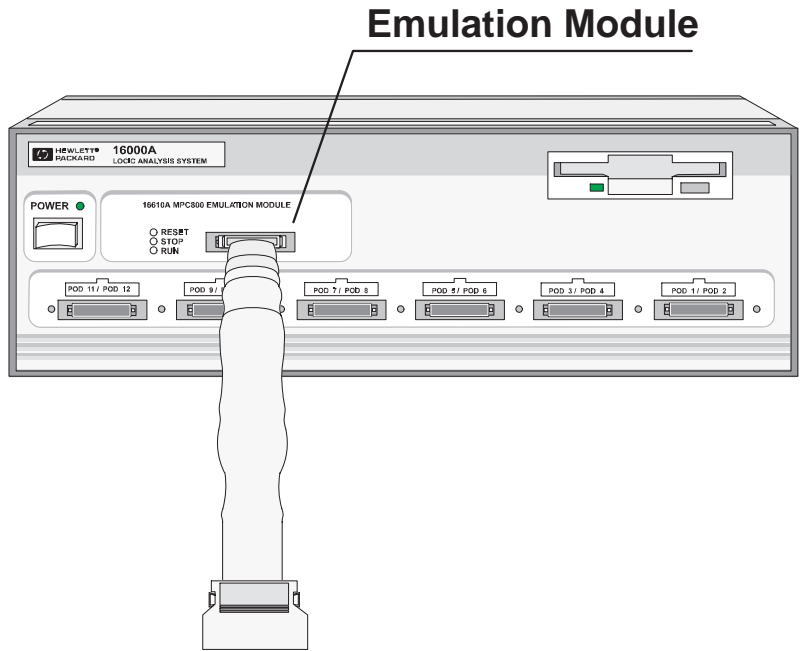
Select “Yes” to have the MPC860 personality loaded into the module, then go the next dialog.



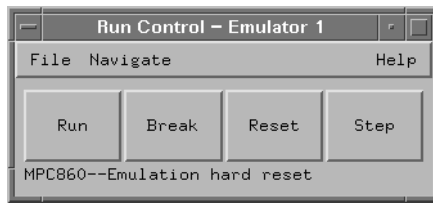
The “Setup Assistant – Emulation Module/Probe Connections” dialog gives you instructions on how to connect the emulation module to the demo board.

When you finish connecting the emulation module cable, proceed to the next dialog.





The setup assistant will now install the MPC860 personality and launch the “Run Control – Emulator 1” dialog.

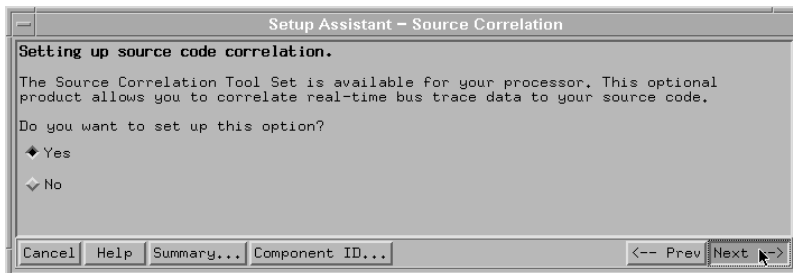


## 8 Turn on the Source Viewer.

The “Setup Assistant – Source Correlation” dialog should now be up. If it isn’t, it could be that the source correlation tool set license has not been turned on for the system you are using. Contact your HP representative to get this corrected.

Select “Yes” to have the setup assistant install the source correlation tool set. The source correlation tool set lets you

view the high-level source code associated with data captured by the logic analyzer.

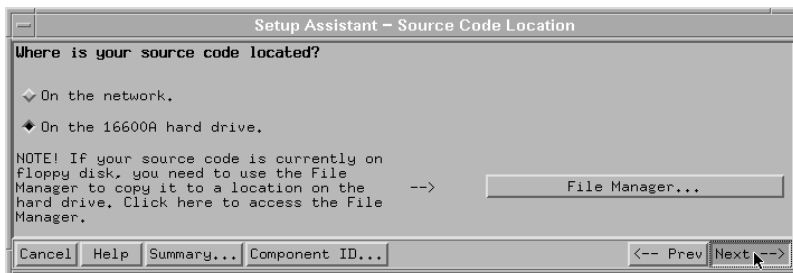


Go to the next dialog.

## 9 Identify the location of the demo board's source code.

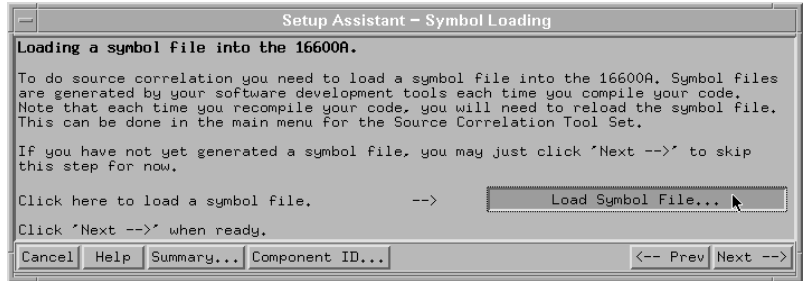
The “Setup Assistant – Source Code Location” dialog helps you load your source code. Source code can be located on a remote computer in your network (if the logic analyzer has been connected to the network), or you could have transferred the source code to the logic analyzer's hard disk. In this example, the source code for the demo board is located on the logic analyzer's hard disk.

Select “on the 16600A hard disk” and go to the next dialog, “Setup Assistant – Symbol Loading”.



**10** Load the symbol file.

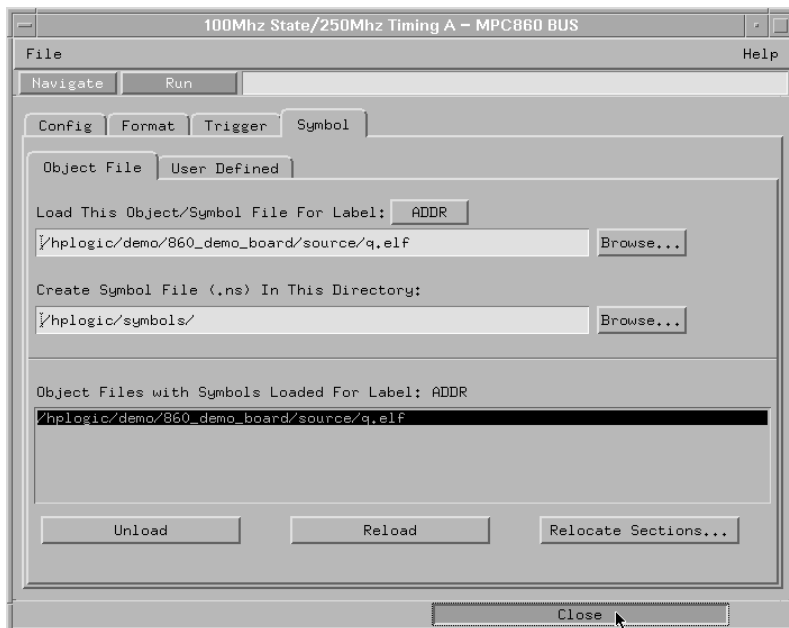
Select “Load Symbol File..”.



The dialog launched is the analysis module’s Setup dialog. The “Symbol” tab has been pre-selected for you. Select the “Browse..” button next to the “Load This Object/Symbol File for Label: ADDR” text box.

Open the demo folder by clicking on the plus symbol in front of it. Open the 860\_demo\_board folder, select the Source folder, and then scroll down to the file called q.elf.

Select the “q.elf” file in the “demo/860\_demo\_board/Source” directory, and click the “Load” button. The symbol file for the demo board has now been loaded.



Minimize the “100MHz State/250MHz Timing A- MPC860 BUS” dialog.

Go to the next Setup Assistant dialog. You will find that this opens the Source Viewer display and the Listing display. These are the dialogs that you will use to trace the MPC860 microprocessor.

## 11 Save the configuration.

The “Setup Assistant – Saving to Disk” dialog is now up. This dialog prompts you to save your configuration to disk. If you save the configuration, you can load it the next time you turn

on the analyzer; this lets you configure the logic analysis system without having re-run the setup assistant.



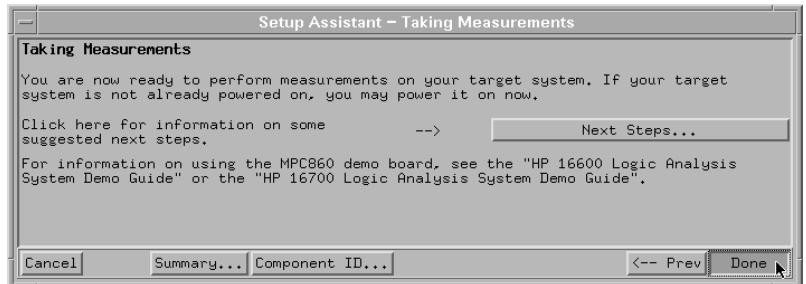
## 12 Exit the Setup Assistant.

The "Setup Assistant - Taking Measurements" dialog is your last dialog in the Setup Assistant.

### What to Do Next

You can get some suggestions about what to do next by selecting "Next Steps". This will take you into the on-line help system.

Since you will be getting help from this demo guide as to what to do next, just select "Done" for now.



**13** Start MPC860 demo board execution.

Now you are ready to take a trace. First, you must get the MPC860 microprocessor running. Run Control starts up in reset by default. Click the “Run” button in the “Run Control – Emulator 1” dialog, you should see “MPC860—Running user program” at the bottom of the dialog.

**14** Capture MPC860 demo board execution.

Now, click the green Run button in any window and the analyzer will take a trace for you.

**15** Display the captured data.

Open the “Listing” display. Scroll through the listing and watch how the source viewer follows along.

Go to the Source Viewer display, select the “Step Source” tab, and click “Previous” and “Next” to step through the source code. Watch how the “Listing” trace follows the source code.

**Summary**

You can quickly set up the logic analysis system to capture a specific microprocessor’s execution using HP’s probing, run control, source code viewing, and setup assistant tools.



---

**Quickly Find the Cause of  
Difficult HW/SW Interaction  
Problems**

## Looking at Correlated Hardware/Software Traces

Being able to look at a problem from different perspectives helps you gain insight into problems faster.

- By looking at different areas in a target system and by time correlating the captured data, you can view the flow of data from one part of the target system to the next.
- By looking at the analog values of a signal as well as the digital values and by time-correlating the captured data to a trace of the processor, you can see how hardware symptoms might be caused by a software problem or how software symptoms might be caused by a hardware problem.

## Correlating processor execution with external buses

You can follow the flow of data in a target system by capturing and correlating data from different parts of the target system. For example, you can see how data flows from a microprocessor bus to an external bus or port.

### **Analysis Probes for Standard Buses**

Analysis Probes are available for over a dozen buses. This includes PCI, ISA, PCMCIA, SCSI 1, 2, and 3, USB, VME, and more. Analysis probes make it easy to connect the logic analyzer to a bus. They provide circuitry to decode the bus's various states, and they configure the analyzer to correctly present the bus's states in symbolic form.

Visit the HP web site at [www.hp.com/go/uPTools](http://www.hp.com/go/uPTools), or contact your HP sales representative, for a current list of buses supported by analysis probes.

On the MPC860 demo board, you can capture code executed on the microprocessor bus and correlate it to the waveform trace captured on the asynchronous, serial Controller Area Network (CAN) bus.

Also, you can use the logic analysis system's serial analysis tool set to interpret the waveform trace of the CAN bus.

As is the case in this demo, you can configure a logic analysis module into two "virtual" logic analyzers:

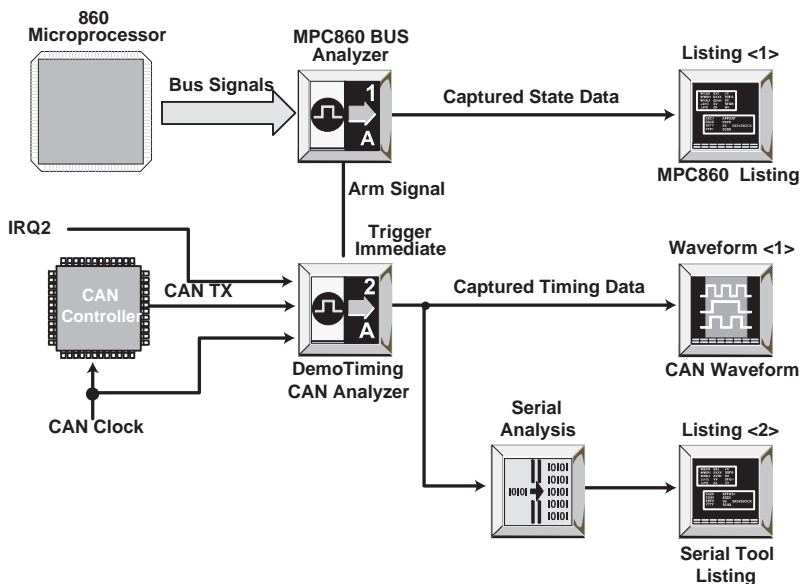
- one logic analyzer that captures microprocessor execution synchronously with the microprocessor's clock (a *state analyzer*).
- one logic analyzer that captures data on the CAN bus asynchronously at sample rate determined by an internal clock (a *timing analyzer*).

(For more explanation of the difference between timing and state analyzers, see “Timing Analysis vs. State Analysis in Logic Analyzers” on page 138.)

The two virtual analyzers can run independently, or, as in this demo, the trigger of the MPC860 BUS analyzer can arm the DemoTiming analyzer.

For this exercise, Analyzer1, called “MPC860 BUS”, is configured as a state analyzer to trace the demo board’s processor. Pods 1 through 4 of the slot A analysis module have been assigned to this analyzer. These pods are connected to the processor.

Analyzer2, called “DemoTiming”, is configured as a timing analyzer. Pods 5 and 6 of the slot A analysis module have been assigned to this analyzer. Pod 5 is connected to the CAN bus TX line, IRQ2, and CAN clock.



## 1 Probe the MPC860 demo board.

Follow the instructions in the “Getting Started” chapter on page 9 for connecting the logic analysis module to the demo board.

---

**NOTE:**

Make sure the pattern generator data pods are not connected to the demo board; otherwise, the changes in demo program execution will affect the results of the measurements that follow.

---

## 2 Load the configuration files for this demo.

In the main logic analysis system window, click the File Manager button.



In the File Manager dialog, select the CAN\_bus.\_\_\_\_ configuration file from the /hpllogic/demo/860\_demo\_board/configs/hp1660x directory, and click the Load... button.

In the Load Configuration dialog, click Load.

If a confirmation dialog appears, click Yes.

(You can also choose the File, Load Configuration... command from the menu bar of most windows.)

## 3 Make sure the MPC860 demo board processor is running in a known state.

Click the “Reset”, “Break”, and “Run” buttons in order.



Resetting the processor starts it from a known state. Breaking puts the processor into the background monitor and allows the emulation module to read the microprocessor's configuration registers (whose contents enable breakpoints among other things).

#### **4** Set up to trigger on a source line.

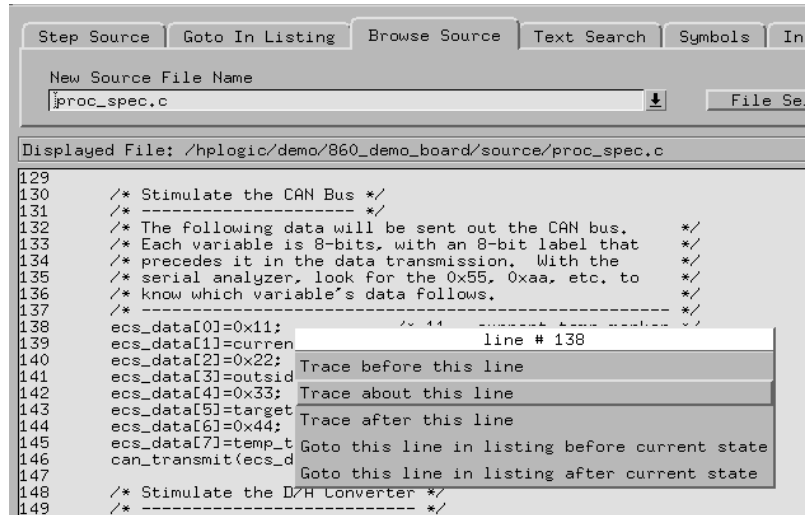
To follow the flow of data from when it is fetched from memory to when it flows on the CAN bus, first trigger the analyzer that captures microprocessor execution when data is fetched from memory.

To set this trigger up, go to the "Browse Source" tab of the "Source Viewer<1>" window, click "File Selection...", select the `/hplogic/demo/860_demo_board/source/proc_spec.c` file, and click OK. This is the source code for one of the main functions executed by the MPC860 demo board.

Scroll down to line 130, which is the beginning of the CAN bus stimulus routine. You can see that this is where the data string that is going to be put onto the CAN bus gets built up. This data string includes marker bytes 11, 22, 33, 44, and temperature bytes.

If you would like to better understand the purpose of the temperature values, see the overview of the code that is executing in the demo board located in the Appendix , "About the MPC860 Demo Board," on page 113.

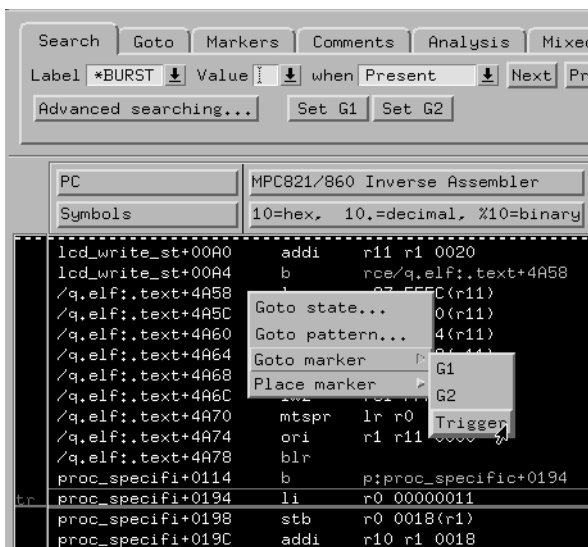
To set up the logic analyzer trigger, click on line 138 and select the “Trace about this line” option. Make sure the heading of the option list says “line # 138”.



- 5 Click the green Run button to perform the measurement.
- 6 Display the captured microprocessor data.

Go to the microprocessor trace listing, right click in the listing area, select “Goto Marker >”, and select “Trigger”. This places the center of the listing on the trigger point. You see that the highlighted line of the source viewer is line 138.

## Chapter 3: Quickly Find the Cause of Difficult HW/SW Interaction Problems Looking at Correlated Hardware/Software Traces



Scroll down through the listing and you see the value of the four temperatures being fetched from memory. The data reads are in yellow. Notice the correct symbols for the temperatures are used in the PC column.

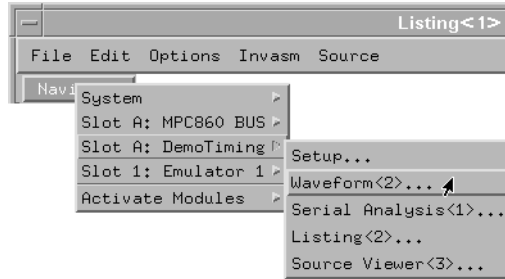
Continue to scroll down until you see the turquoise data writes to the CAN bus controller. You see that the temperature markers and then the temperatures were written to the controller. You then see that a 0x43 and 0xE6 was written to the controller. This told the controller to transmit the data.

### 7 Display the captured CAN bus data using the serial analysis tool set.

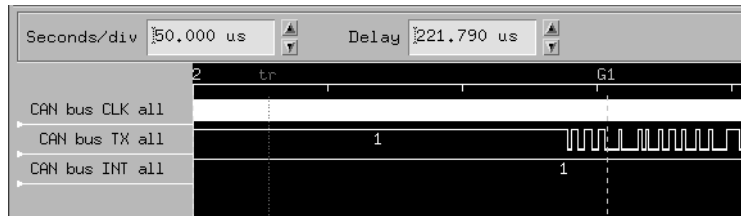
Now, verify that the correct data was transmitted on the CAN bus.



Click one of the “Navigate” buttons. Select “DemoTiming >”, and select Waveform<2>.



This brings up the waveform trace of the CAN bus that was taken when you last ran the analyzer. It is a trace of the CAN clock, CAN transmit line, and CAN interrupt line. The trigger point on the trace is the same trigger point that is on the processor listing.



Now, look at the output of the serial tool to verify that the correct data was placed on the CAN bus. Click one of the “Navigate” buttons”, select “DemoTiming >”, and select “Listing<2>...”.

This listing is the output of the serial analysis tool. The listing shows the start block of the transmission (0CCC), the data transmitted, and the end block (7F). You see that the data fetched from memory was correctly transmitted on the CAN bus

### 8 Use global markers to correlate the captured data.

You can use the global markers to help you look at what the serial analysis tool is doing.

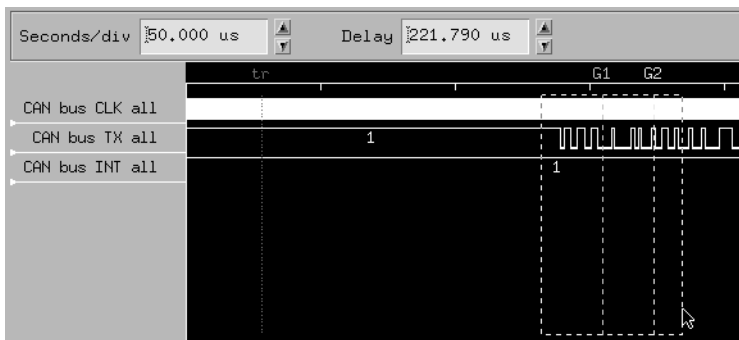
Right click in the listing area of the “Listing<2>” dialog. Select “Place Marker >” and “G1”.



Drag the G1 marker to just under the first 0CCC.

Place the G2 marker in the listing and drag it to just under the 11 in the data stream

Go to the “Waveform<2>” dialog. Place the pointer just to the left of the beginning of the CAN bus transmission. Click and hold the left mouse button and drag the left edge of the rectangle that was formed to a point just to the right of the G2 marker.



You now have an expanded view of the trace. You can see the form of the data transmitted on the CAN bus. (Keep in mind the CAN bus protocol specifies that a 0 gets added, or *stuffed*, after five 1s and a 1 gets stuffed after five 0s.)

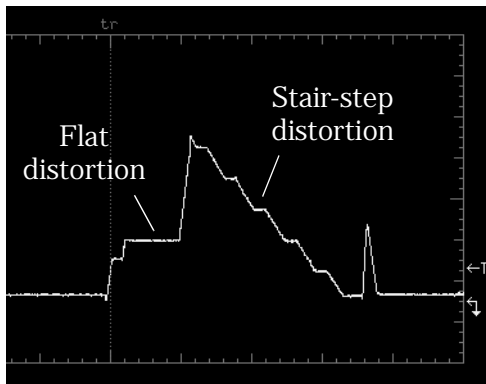
**Summary**

By looking at different areas in a target system and by correlating the captured data, you can view the flow of data from one part of the target system to the next.

## Tracking hardware problems to their software causes

You can identify a problem with an analog signal using an oscilloscope module, trigger a logic analyzer module tracing software execution, and correlate the captured data to identify the cause of the problem all the way to the source code.

For example, the demo board's MPC860 processor has been programmed to generate a triangular waveform from a D/A converter, but the oscilloscope module shows problems with the waveform.

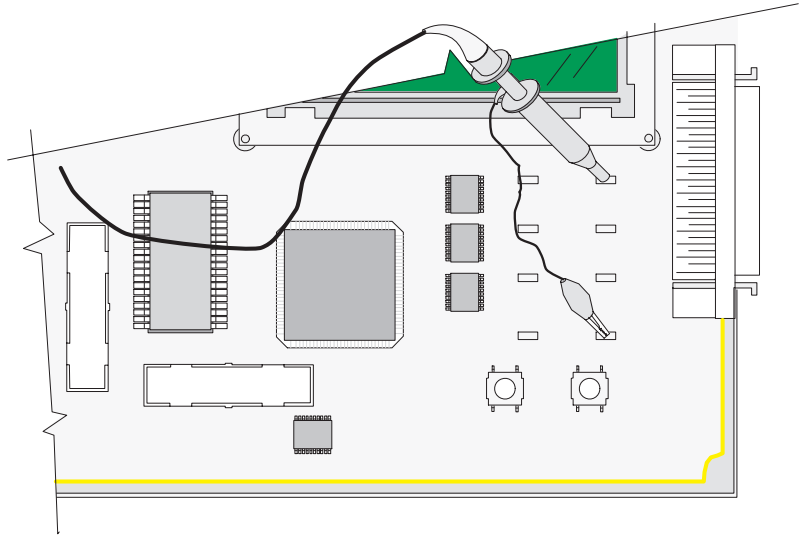


There are two problems with the triangular waveform: when the voltage is flat for approximately 350 microseconds, and when the voltage is flat for shorter times in varying numbers and duration, leading to a stair step effect.

- 1 Connect the logic analyzer, emulation module, and oscilloscope to the demo board.

Follow the instructions in the “Getting Started” chapter on page 9 for connecting the logic analysis module to the demo board.

Also, connect the channel 1 oscilloscope probe to the D/A test point on the demo board. And, connect the ground clip to the GND test point.



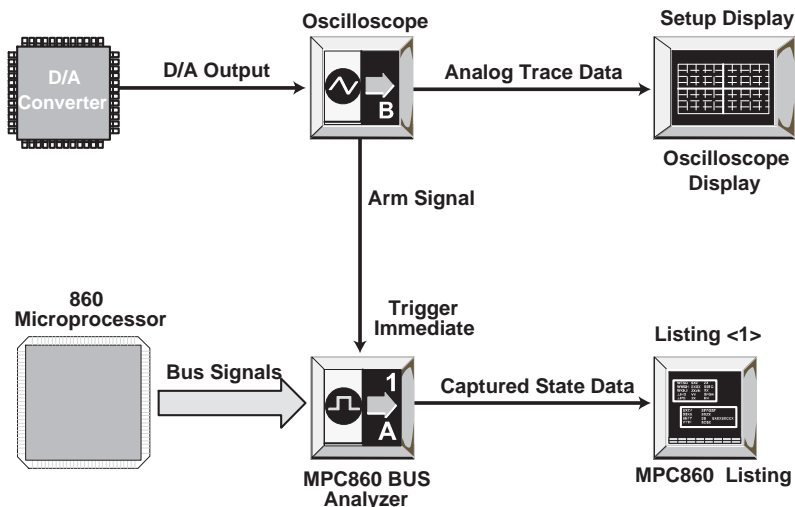
---

**NOTE:**

Make sure the pattern generator data pods are not connected to the demo board; otherwise, the changes in demo program execution will affect the results of the measurements that follow.

---

**2** Load the configuration files for this demo.



You will be loading the above configuration that will allow you to make a time-correlated measurement involving the oscilloscope and logic analysis modules. The oscilloscope module is configured to arm the logic analysis module when it triggers. The logic analysis module, in turn, is configured to trigger immediately when it is armed. This results in the two traces occurring at the same time and allows them to be time-correlated.

In the main logic analysis system window, click the File Manager button.



In the File Manager dialog, select the D\_to\_A.\_\_\_\_ configuration file from the /hpllogic/demo/860\_demo\_board/configs/hp1660x directory, and click the Load... button.

In the Load Configuration dialog, click Load.

If a confirmation dialog appears, click Yes.

(You can also choose the File, Load Configuration... command from the menu bar of most windows.)

- 3** Make sure the MPC860 demo board processor is running in a known state.

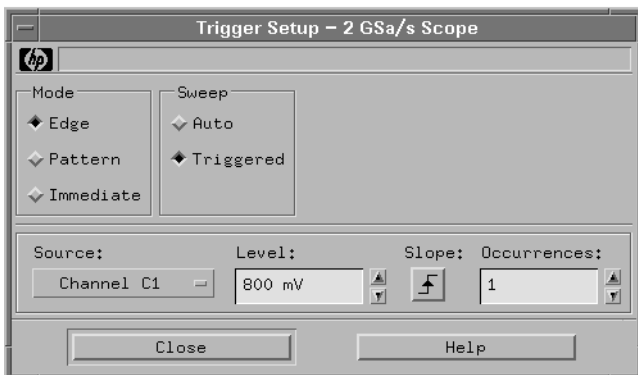
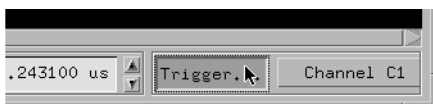
Click the “Reset”, “Break”, and “Run” buttons in order.



Resetting the processor starts it from a known state. Breaking puts the processor into the background monitor and allows the emulation module to read the microprocessor's configuration registers (whose contents enable breakpoints among other things).

#### 4 Start the oscilloscope measurement.

Select “Trigger..” on the bottom of the oscilloscope display. Notice that the oscilloscope has been set up to trigger on an edge.



Click the “Group Run” button to take a trace and view the triangular waveforms. You should see the “flat distortion” defect, but you may not see the “stair step” distortion because it is very intermittent.

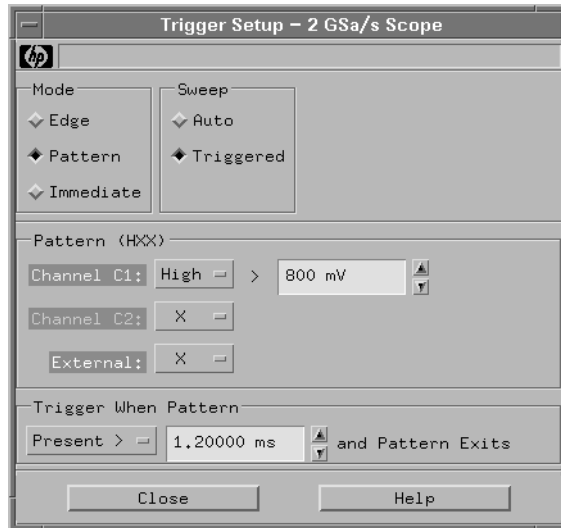
Click “Group Run” to view the captured waveforms a few more times. You see that the triangular waveform goes through various forms, from mildly distorted to heavily distorted. Notice that the heavily distorted triangular waveforms are infrequent.

#### 5 Modify the oscilloscope trigger.

Now, set up the oscilloscope to trigger on a heavily distorted waveform. Go to the “Trigger Setup - 2 Gsa/s Scope” dialog by selecting “Trigger..”; then, select the “Pattern” radio button.



The pattern trigger has already been setup for a waveform that stays above 800 mV for more than 1.2 msec.



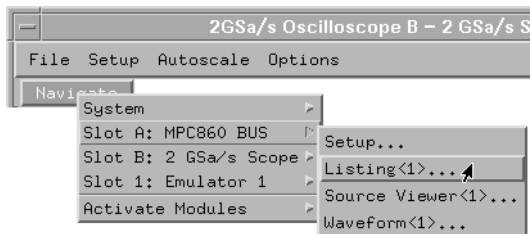
Click “Group Run” to capture one of the heavily distorted waveforms. (It may take several seconds because the waveforms are so infrequent.)

If the oscilloscope does not trigger, it may be because of variations in the demo boards. Shorten the period of the pattern trigger to capture a less distorted version of the waveform.

Close the trigger dialog to get it out of the way

## 6 Open the Listing and Source Viewer displays.

Click the “Navigate” button, select the “MPC860 BUS >” option, and select “Listing<1>...”.



Click the “Navigate” button again, select the “MPC860 BUS >” option, and select “Source Viewer<1>...”.

Move the Listing<1> window to the lower right hand corner of the screen and the Source Viewer window to the upper left hand corner of the screen to get them out of the way.

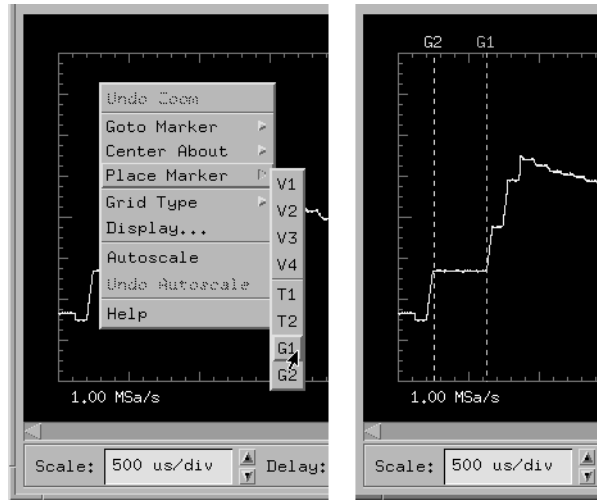
## 7 Use global markers to correlate the captured data.

### Global Markers

The logic analysis system provides timing markers that span any modules that have been correlated. Placing a marker in the display of one module positions an identical marker at the correct time position in all correlated module displays.

Now, mark off the 350 microsecond (approximately) flat distortion with the global markers. Right click on the oscilloscope trace and select “Place marker >” and select “G1”. Don’t worry about getting the marker in exactly the right place at first, because they can be dragged into place after they appear. Place the G1 marker just past the end of the 350 microsecond (approximately) stretch in the “triangular” waveform. Place the G2 marker just before the beginning of the stretch.

Note that the inverse assembled listing has now moved to the location of the code that was executed at the G2 marker.



PC	MPC821/860 Inverse Assembler	
Symbols	10=hex, 10.=decimal, %10=binary	
D/A control (CS5)	write	64
proc_specifi+0238	addi	r6 r6 000A
proc_specifi+023C	andi	r6 r6 00FF
/q.elf:reset+0400	subi	r1 r1 0080
/q.elf:reset+0404	stmw	r0 0000(r1)
/q.elf:reset+0408	mf spr	r26 srr0
/q.elf:reset+040C	mf spr	r27 srr1
/q.elf:reset+0410	mfc r	r28
/q.elf:reset+0414	mf spr	r29 lr
/q.elf:reset+0418	mf spr	r30 ctr
/q.elf:reset+041C	mf spr	r31 xer
/q.elf:reset+0420	subi	r1 r1 0018
/q.elf:reset+0424	stmw	r26 0000(r1)
G2. /q.elf:reset+0428	mf spr	r26
/q.elf:reset+042C	ori	r26 r26 0002
/q.elf:reset+0430	mtmsr	r26
/q.elf:reset+0434	subi	r1 r1 0008
/q.elf:reset+0438	bl	q. :isr:ext_exception
isr:ext_exception	mf spr	r0 lr
ext_exceptio+0004	stw	r0 0004(r1)
ext_exceptio+0008	stwu	r1 FFF8(r1)
ext_exceptio+000C	lis	r12 0000
ext_exceptio+0010	lwz	r11 4004(r12)

Bring the Source Viewer to the front and then bring the processor trace listing to the front. As you scroll through the processor trace listing, watch the Source Viewer. You see the source code move from running D/A code at the G2 marker to

servicing CAN interrupts. It will pass through the CAN service routine twice. You may have to back up a bit to get into the section of code that writes to the D/A controller. This is because it is difficult to place the marker precisely at the beginning of the flat stretch.

When you go past the G1 marker, you see the code go from the CAN service routine back to the main loop, which updates the D/A. You may also see the IRQ3 service routine execute while you are scrolling.

**Summary**

You can see that the CAN interrupt postpones the updating of the D/A voltage for over 350 microseconds. If the D/A update was considered more important than the CAN controller, you could modify the code to give priority to the D/A controller.

You can go through the same procedure to identify the cause of the smaller stair step distortions in the D/A output signal. By placing the G1 marker at the end of one of the stair steps, placing the G2 marker at the beginning, and scrolling through the listing, you can see that IRQ3 servicing is responsible for stair step effect.

**See Also**

“Controlling and modifying processor execution” on page 71. There you will turn off interrupts and observe the effect on the D/A waveform.

## Tracking software problems to their hardware causes

When software execution behaves in a way that doesn't seem logically possible, you may have to look beyond the logical (digital) behavior of the system to the physical behavior, for example, by looking at the analog parameters of signals. If you can capture a software execution problem with the logic analyzer, you can also capture oscilloscope data from the target system at the same time, and you can correlate the captured data using global markers.

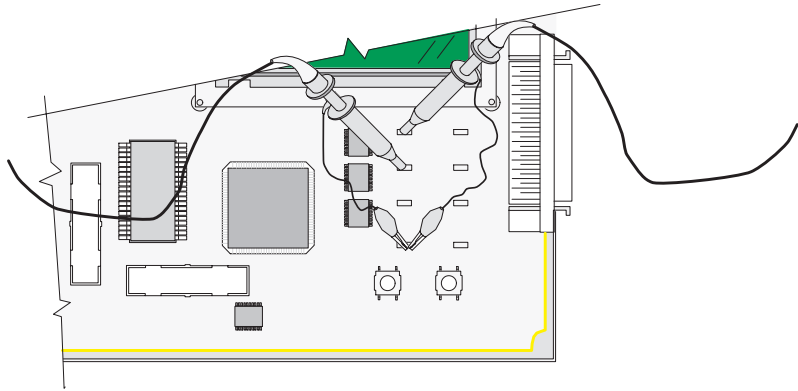
For example, in the MPC860 demo board's program code there are instructions that should never execute. However, if you set up the logic analyzer to trigger on them, you'll find that they do execute. You can set up a correlated measurement to capture data with the oscilloscope at the same time, and you can use global markers to correlate the captured data and discover the hardware cause of the problem.

- 1** Connect the logic analyzer, emulation module, and oscilloscope to the demo board.

Follow the instructions in the "Getting Started" chapter on page 9 for connecting the logic analysis module to the demo board.

Connect the channel 1 oscilloscope probe to the "D0 BOUNCE" test point on the demo board, and connect its ground clip to the GND test point.

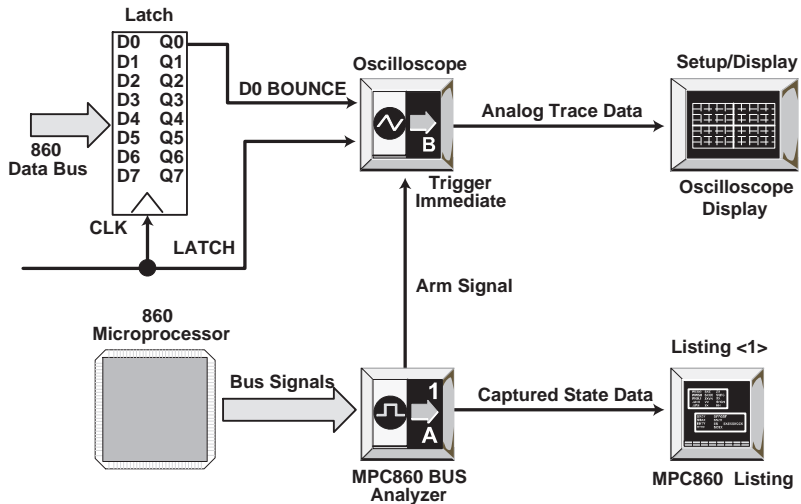
Connect the channel 2 oscilloscope probe to the "LATCH" test point on the demo board, and connect its ground clip to the GND test point.



**NOTE:**

Make sure the pattern generator data pods are not connected to the demo board; otherwise, the changes in demo program execution will affect the results of the measurements that follow.

**2** Load the configuration files for this demo.



You are loading the above configuration. The logic analyzer arms the oscilloscope which then triggers immediately. This

allows the oscilloscope to capture what is happening on the D0 and LATCH lines at the time of the event the logic analyzer triggers on.

In the main logic analysis system window, click the File Manager button.



In the File Manager dialog, select the Register.\_\_\_\_ configuration file from the /hplogic/demo/860\_demo\_board/configs/hp1660x directory, and click the Load... button.

In the Load Configuration dialog, click Load.

If a confirmation dialog appears, click Yes.

(You can also choose the File, Load Configuration... command from the menu bar of most windows.)

**3** Make sure the MPC860 demo board processor is running in a known state.

Click the “Reset”, “Break”, and “Run” buttons in order.



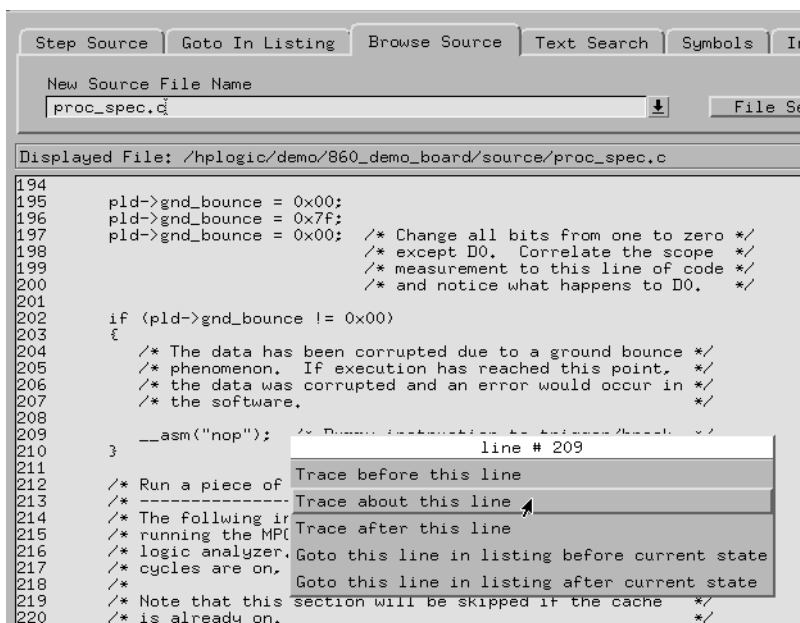
Resetting the processor starts it from a known state. Breaking puts the processor into the background monitor and allows the emulation module to read the microprocessor’s configuration registers (whose contents enable breakpoints among other things).

#### 4 Set up to trigger on the software execution problem.

First take a look at the code where the problem is happening. Go the Source Viewer, select the “Browse Source” tab, select “File Selection...”, and select the file /hpllogic/demo/860\_demo\_board/source/proc\_spec.c.

Scroll down to line 195. What you see is three lines that write 0x00, 0x7f, and 0x00 into a register. When the register is read after these writes, it should contain 0x00. The next line of code checks to see if 0x00 is in the register and goes to a NOP on line 209 if it is not. The code should never get to the NOP if the value in the register is 0x00 like it should be.

We can check to see if the microprocessor executes the NOP by setting the logic analyzer to trigger on it. Select line 209 and click on it. Select “Trace about this line” from the options.

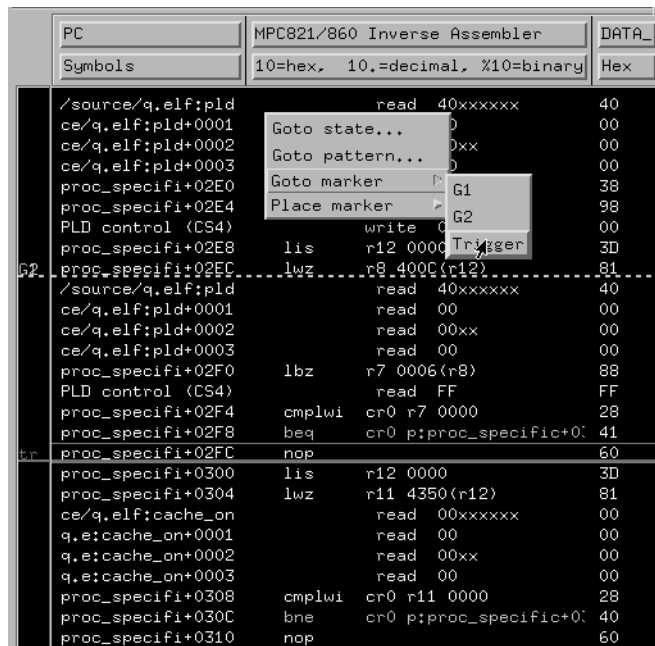




- 5 Click one of the “Group Run” buttons to start the measurement.
- 6 Display the captured data.

After the trace has been taken, go to the microprocessor bus listing. Try scrolling the listing up and down. The source code highlight tracks the listing.

If the trigger is not in the center of the listing, right click on the listing, select “Goto Marker >”, and select “Trigger”. This places the trigger point in the center of the listing.



Notice the logic analyzer triggered on the NOP; this means the register did not contain 0x00. The question now is why.

- 7 Open the oscilloscope display.

Click one of the “Navigate” buttons. Select the “2 Gsa/s Scope” option and select “Setup/Display...”. The yellow channel 1 trace

is the register's D0 line and the green channel 2 trace is the register's clock line.



The oscilloscope has been set to trigger immediately when the logic analyzer triggers, so the trace you see is correlated to the microprocessor bus listing. The oscilloscope is set up to check the waveform of the D0 line when it is latched.

You can see, with the help of the timing marker and voltage marker, that the D0 line is at a logic 1 voltage level when data is latched into the register by the rising edge of the clock.

## 8 Use global markers to correlate data.

You can establish what code was executing when this happened by using the global timing markers.

Right click on the oscilloscope display, select “Place Marker >”, and select “G1”. Drag the G1 marker to the rising edge of the clock signal.

Now, go to the Listing<1> window and right-click in the black listing box. Select Goto Marker and G1. This will center the listing on the G1 marker.

Notice that this corresponds to the last of the three writes to the register. This means that a 0x00 did not get written into the register as it should have.

PC	MPC821/860 Inverse Assembler	
Symbols	10=hex, 10.=decimal, %10=binary	
/source/q.elf:pld	read	40xxxxxx
ce/q.elf:pld+0001	read	00
ce/q.elf:pld+0002	read	00xx
ce/q.elf:pld+0003	read	00
proc_specifi+02E0	li	r0 00000000
proc_specifi+02E4	stb	r0 0006(r9)
G1 PLD_control (CS4)	write	00
proc_specifi+02E8	lis	r12 0000
G2 proc_specifi+02EC	lwz	r8 400C(r12)
/source/q.elf:pld	read	40xxxxxx
ce/q.elf:pld+0001	read	00
ce/q.elf:pld+0002	read	00xx
ce/q.elf:pld+0003	read	00
proc_specifi+02F0	lbz	r7 0006(r8)
PLD_control (CS4)	read	FF
proc_specifi+02F4	cmplwi	cr0 r7 0000
proc_specifi+02F8	beq	cr0 p:proc_specific+0
proc_specifi+02FC	nop	
proc_specifi+0300	lis	r12 0000
proc_specifi+0304	lwz	r11 4350(r12)
ce/q.elf:cache_on	read	00xxxxxx

It is also worth noting that the three writes consisted of 0x00, 0x7F, and 0x00. This means that the MSB, which is D0 in this system, never should have transitioned from a logic 0 during any of the writes. Yet the oscilloscope trace shows a logic 1 right at the rising edge of the latch.

You can also explore this phenomenon further by going up to line 182 in the source code. Here you will find three writes that

consist of 0x00, 0x01, and 0x00. If you set the trigger to the NOP after these writes and run the analyzer, you will find that it will not trigger.

So, what does this mean? When we only transition a single bit, as in the one set of writes, the correct value is latched by the register. When we transition most of the bits, as in the other set of writes, the wrong value gets latched by the register. A likely cause is ground bounce that gets severe enough to cause problems when most of the bits transition. Let's investigate further.

**9** Probe the register's ground, and re-run the measurement.

You can verify that ground bounce is the problem by connecting the channel 1 oscilloscope probe to the GND BOUNCE test point. This test point connects directly to the ground connection of the register.

Click "Group Run" to take a trace of the ground line and observe the ground bounce.

Use the oscilloscope's voltage markers to assess the severity of the ground bounce.



### Summary

By looking at the analog values of a signal as well as the digital values and by correlating the captured data, you can see how software symptoms might be caused by a hardware problem.

## Looking at Firmware Driver Issues

Being able to start and stop program execution, set up logic analyzer triggers at locations in source code, and modify processor register and memory contents helps you develop and debug firmware drivers faster.

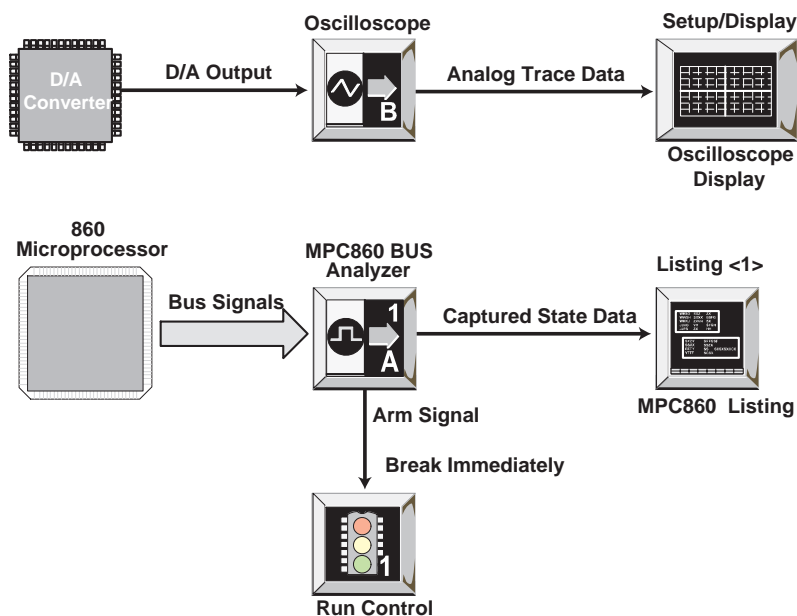
- By controlling the target system microprocessor's execution (run, stop, step, breakpoints), you can stop program execution at certain points and examine the state of your system.
- By using an HP emulation module, you can download code to target system RAM and execute it.

## Controlling and modifying processor execution

If your HP 16600A-series logic analysis system has an emulation module connected to a target system microprocessor's debug port, you can use the emulation module to start and stop the processor, and set breakpoints.

Normally you would use a Target Interface Module (TIM) to connect to the processor's debug port. In the case of the MPC860 demo board, however, the TIM is built in. Therefore, it has an emulation module connector right on the board, and you're able to control the MPC860 microprocessor execution using the emulation module.

The MPC860 demo board's D/A converter has a triangular waveform output signal with distortion caused by interrupts. In this demo, you will stop microprocessor execution, modify a microprocessor register to disable interrupts, continue microprocessor execution, and observe the effects on the triangular waveform.



The configuration you will load for this exercise arms the emulation (run control) module from the logic analyzer. This means that whatever the logic analyzer triggers on will cause a break in the processor's execution very close to that point. It is also important to note that the oscilloscope runs completely independently of any of the other instruments.

- 1 Connect the logic analyzer, emulation module, and oscilloscope to the demo board.

Follow the instructions in the “Getting Started” chapter on page 9 for connecting the logic analysis module to the demo board.

Connect the channel 1 oscilloscope probe to the “D/A” test point on the demo board, and connect its ground clip to the GND test point.



---

**NOTE:**

---

Make sure the pattern generator data pods are not connected to the demo board; otherwise, the changes in demo program execution will affect the results of the measurements that follow.

**2** Load the configuration files for this demo.

In the main logic analysis system window, click the File Manager button.



In the File Manager dialog, select the Turn\_off\_IRQ.\_\_\_\_ configuration file from the /hpllogic/demo/860\_demo\_board/configs/hp1660x directory, and click the Load... button.

In the Load Configuration dialog, click Load.

If a confirmation dialog appears, click Yes.

(You can also choose the File, Load Configuration... command from the menu bar of most windows.)

The configuration file sets up the emulation module to stop microprocessor execution when the logic analyzer triggers.

**3** Make sure the MPC860 demo board processor is running in a known state.

Click the “Reset”, “Break”, and “Run” buttons in order.



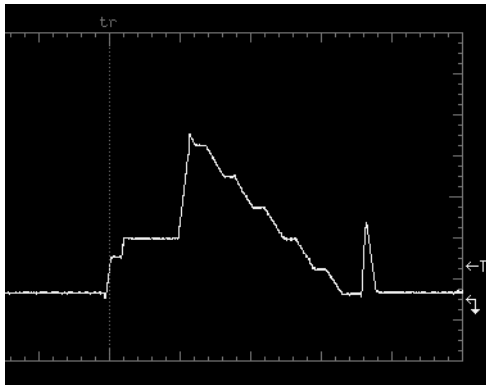
Resetting the processor starts it from a known state. Breaking puts the processor into the background monitor and allows the

emulation module to read the microprocessor's configuration registers (whose contents enable breakpoints among other things).

#### 4 Display the triangular waveform's distortion.

In the first step of this measurement, you look at the analog triangular waveform generated by the D/A and processor.

To see that the form of the distortion varies over time, the oscilloscope is set up to run and trigger repetitively. In the "2G Sa/s Oscilloscope - Display" dialog, click the "Run(r)" button.



Let the oscilloscope trace the D/A repetitively for a while. You will see that you cannot get a true triangular waveform trace. Stop the oscilloscope trace, right-click the "Run(r)" button, and select "Single".

#### 5 Look at the source code generating the triangular waveform.

Open the Source Viewer window by selecting Navigate, MPC860 BUS, and SourceViewer<1>.

In the Source Viewer dialog select the "Browse Source" tab and select "File Selection...". Select the file /hplogic/demo/860\_demo\_board/source/proc\_spec.c file. Go to line 88, this is the line where IRQ3 gets enabled on each pass through the

proc\_specific loop. IRQ3 is disabled towards the end of the proc\_specific loop.

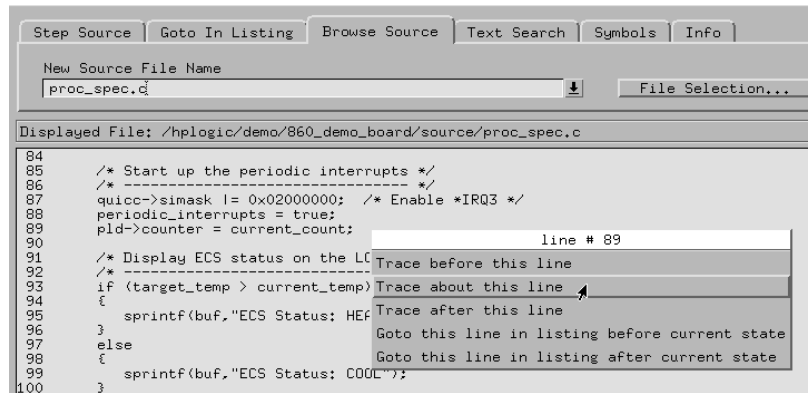
It is worth noting that IRQ2, the CAN interrupt and cause of the flat stretch, is only turned on during the demo board initialization. If it gets turned off, it stays off.

Now, go to lines 154 through 166 where you see the triangular waveform generation code.

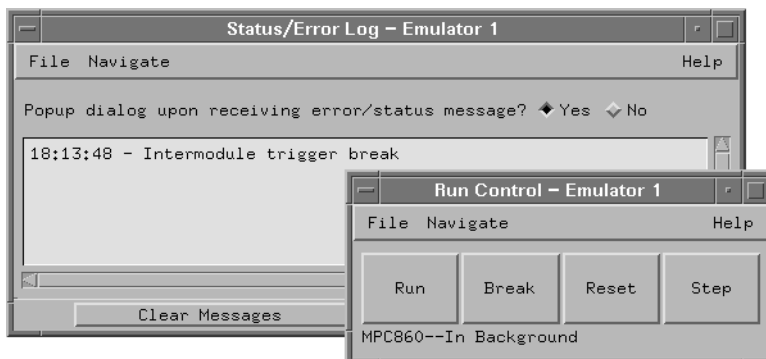
If you would like more detailed information about what the Proc\_specific loop does, go to “Demo Board Firmware” on page 127.

**6** Stop processor execution just after IRQ3 is enabled (and before the triangular generation code is executed).

In the “Source Viewer” window, click on line 89 and select “Trace about this line”. Click one of the “Group Run” buttons to start the analyzer trace.



You get a status message that says there has been an “Intermodule trigger break”. This message, and the status line at the bottom of the “Run Control” dialog, inform you that the processor has stopped executing demo board firmware and is in the background monitor.



Click “Stop” to stop the trace measurement. (When processor execution stops, there are no states for the analyzer to capture and fill trace memory with, so the measurement does not complete).

You can verify where you are by looking at the processor trace in the “Listing<1>” dialog and at the correlated source in the Source Viewer window. Remember that the trigger must be centered in the listing display.

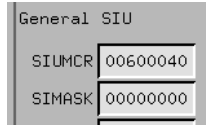
```
Displayed File: /hplogic/demo/860_demo_board/source/proc_spec.c
84
85 /* Start up the periodic interrupts */
86 /* ----- */
87 quicc->simask |= 0x02000000; /* Enable *IRQ3 */
88 periodic_interrupts = true;
89 pld->counter = current_count;
90
91 /* Display ECS status on the LCD */
92 /* ----- */
93 if (target_temp > current_temp)
94 {
```

## 7 Modify the SIMASK register to turn off interrupts.

Click a “Navigate” button, select “Emulator<1>”, and select “Registers...”. This dialog shows you selected registers of the processor and their values. The Registers dialog has been preconfigured to show only the SIU group.

Turn off interrupts by going to the SIMASK register, entering all 0’s, and then moving the cursor out of the SIMASK text window.

(Moving the cursor out of the text window causes the register change to take effect.)



### 8 Start the oscilloscope measurement.

Make sure the oscilloscope is not set to run repetitively.

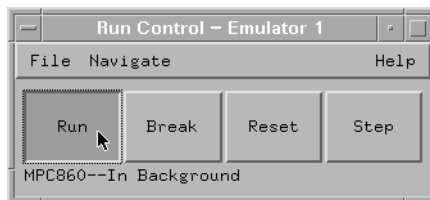
Click the green “Run” button on the oscilloscope.

The oscilloscope will not trigger until the processor is running because the D/A waveform will not be generated until then.

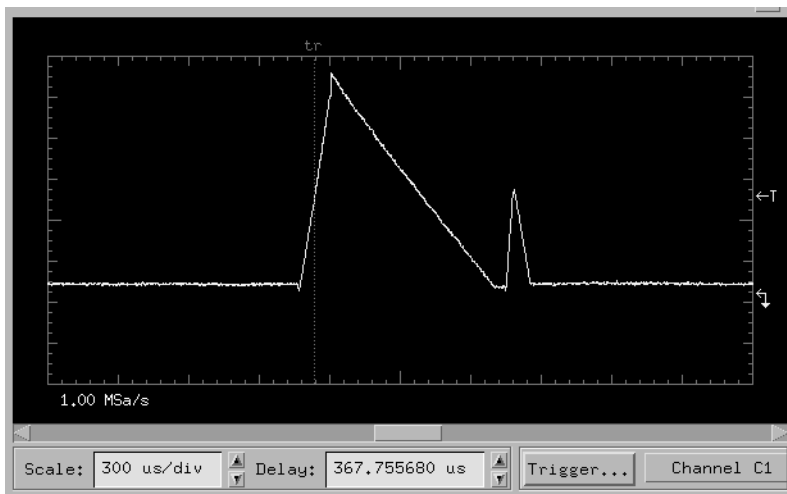
Also, note that the oscilloscope is not part of the group run associated with the analyzer. It is an independent instrument.

### 9 Continue processor execution.

In the Run Control dialog, click the Run button.



The oscilloscope should trigger almost immediately. What you get on the display is a “perfect” triangular waveform.



From this exercise, you can see that the defects in the triangular waveform were caused by interrupt service routines executing during the generation of the waveform.

**10** Modify the SIMASK register to turn interrupts back on.

As the processor continues to run, IRQ3 will be turned back on again by code in the Proc\_specific loop; however, the CAN bus interrupt will not. You can verify this by running the oscilloscope trace repetitively. You will not see the CAN bus flat distortion, but you will see the IRQ3 stair step effect.

You can turn IRQ2 on again, along with other interrupts. Stop the processor in the proc\_specific function by selecting one of the group runs. Now, go to the “Registers – Emulator 1” dialog, select the SIMASK text window, enter a 2a000000, and then move the cursor out of the SIMASK text window. Run the processor by clicking “Run” in the Run Control dialog.

Click the green Run button in the oscilloscope again to trace the triangular waveform repetitively and view the effect of the register change on the waveform.

**Summary**

Controlling the target system microprocessor's execution (run, stop, step, breakpoints) lets you stop program execution at certain points and examine or modify the state of your system.

## Downloading code to RAM or Flash ROM

If your HP 16600A-series logic analysis system has an emulation module connected to a target system microprocessor's debug port, you can use the emulation module to download code to RAM or Flash ROM. Once code has been downloaded, you can use the emulation module to start the processor executing that code.

It is easy to quickly try out new or modified pieces of code on the target using the following procedure. Boot the processor and let the code in ROM initialize the system. Using the emulation module, break the processor, download the code, and change the PC to the start address of the code in RAM. Run the processor to execute the new code

For example, there is code located on the analysis system that you can download into RAM and start executing. The code will modify the message displayed on the LCD so that you will know it is running.

- 1** Connect the logic analyzer and emulation module to the demo board.

Follow the instructions in the “Getting Started” chapter on page 9 for connecting the logic analysis module to the demo board.

- 2** Load the configuration files for this demo.

In the main logic analysis system window, click the File Manager button.





In the File Manager dialog, select the Download.\_\_\_\_ configuration file from the /hplogic/demo/860\_demo\_board/configs/hp1660x directory, and click the Load... button.

In the Load Configuration dialog, click Load.

If a confirmation dialog appears, click Yes.

(You can also choose the File, Load Configuration... command from the menu bar of most windows.)

### **3** Set up the microprocessor for downloading code.

Reset, break, and run the processor, and wait until the LED stops flashing green. This ensures that the microprocessor has properly initialized.

Then, break the processor so it is not running when the download happens.

### **4** Load the executable file into RAM.

Select Navigate, then Emulator<1>, then “Load Executable...”.

Note that, in addition to downloading code into RAM, you can download code into Flash ROM or erase Flash ROM. Go ahead and explore these options if you would like. Note that the four main Flash algorithms are supported. Very few Flash ROMs do not support one of these algorithms. Before you continue be sure you return the “Operation” selection at the top of the dialog to “Load Executable”.

The file you are about to load is in Motorola S-record form, so it will automatically be loaded into the correct memory location. Note also that “Set PC after load” is selected. This means that when you hit run after the code is downloaded, execution will start at the beginning of the downloaded program. Select the “Browse...” button at the bottom of the dialog and select the file /hplogic/demo/860\_demo\_board/download/demo.srec. Select OK in the file browser dialog; then, select Apply and the file will be loaded. Click OK in the Load Completed dialog, and close the Load Executable window.

Now, look at the LCD display and you will see something like:

```
ECS Status: COOL  
Current Temp ##
```

**5** Run the downloaded code.

Select Run on the Emulation dialog and note that the LCD display now says:

```
Running \
```

This comes from the code you just downloaded and verifies that it is executing.

**6** Trace the code executing out of RAM.

You can now trace the code executing out of RAM. Click the green Run button in the Listing dialog. You will get a trace of the code executing in the demo board's RAM as well as a correlation to its source code.

**Summary**

By using an HP emulation module, you can download code to target system RAM and execute it.

## Looking at Software Issues

Being able to make system performance measurements helps you analyze and validate application software.

- By using the HP B4600B system performance analysis (SPA) tool set, you can identify and characterize your target system's performance.
- By using the context store feature, you can easily find the code that is responsible for corrupting a variable.
- By using the cache-on execution tracking inverse assembler, you can trace MPC860 processor execution when caches are turned on.

---

## Analyzing system performance

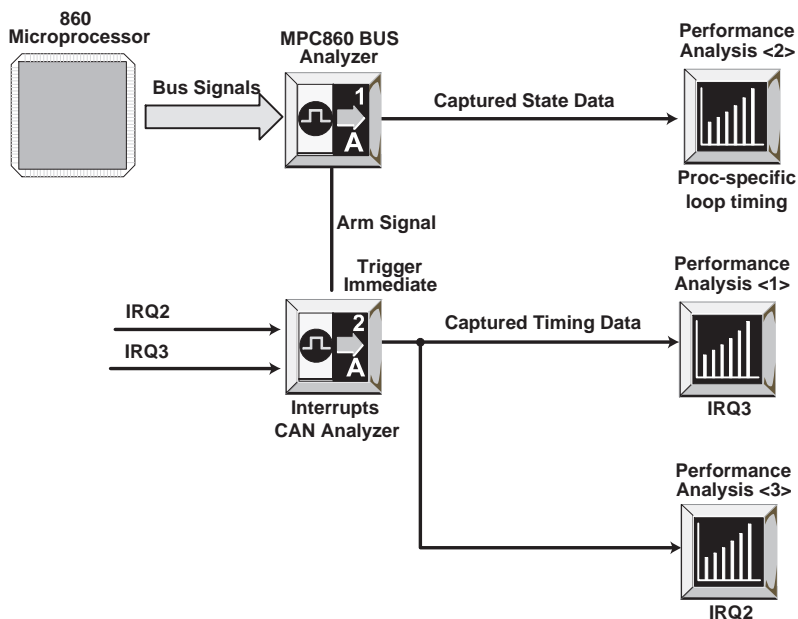
You can quickly find the cause of performance problems in a section of code by using the HP B4600B system performance analysis tool set (also known as the SPA tool set).

This demo uses the SPA tool set to identify an occasional performance problem with a section of code executing on the MPC860 demo board. The SPA tool set is also used to identify the cause of the problem.

### 1 Probe the MPC860 demo board.

Follow the instructions in the “Getting Started” chapter on page 9 for connecting the logic analysis module to the demo board.

### 2 Load the configuration files for this demo.



You are loading the above configuration. The logic analyzer has been split into two virtual analyzers: one state to trace the MPC860 bus, and one timing to trace the IRQs. The outputs of both analyzers go to performance analysis tools.

In the main logic analysis system window, click the File Manager button.



In the File Manager dialog, select the SPA.\_\_\_ configuration file from the /hplogic/demo/860\_demo\_board/configs/hp1660x directory, and click the Load... button.

In the Load Configuration dialog, click Load.

If a confirmation dialog appears, click Yes.

(You can also choose the File, Load Configuration... command from the menu bar of most windows.)

**3** Make sure the MPC860 demo board processor is running in a known state.

Click the “Reset”, “Break”, and “Run” buttons in order.



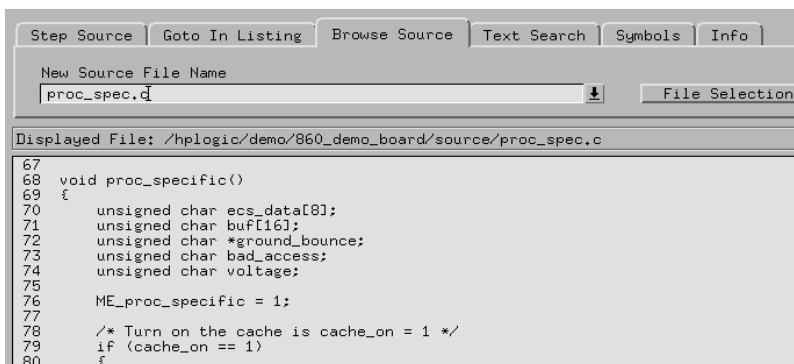
Resetting the processor starts it from a known state. Breaking puts the processor into the background monitor and allows the emulation module to read the microprocessor's configuration registers (whose contents enable breakpoints among other things).

#### 4 Review the source code.

Normally, for a measurement like this, you would be looking at code that you wrote. Since this is not the case, first load the code into the Source Viewer to review it.

Select the “Browse Source” tab, then select “File Selection...”, highlight the file `/hpllogic/demo/860_demo_board/source/proc_spec.c`, and select “OK”.

Go to line number 68 where `proc_specific` begins. On line 76 you see a write to the variable `ME_proc_specific`. This write is used to mark the beginning of `proc_specific`.



```
67
68 void proc_specific()
69 {
70     unsigned char ecs_data[8];
71     unsigned char buf[16];
72     unsigned char *ground_bounce;
73     unsigned char bad_access;
74     unsigned char voltage;
75
76     ME_proc_specific = 1;
77
78     /* Turn on the cache is cache_on = 1 */
79     if (cache_on == 1)
80     {
```

Go to line 275 and you see our end of loop marker, `MX_proc_specific`. You could have also found this line by using the “Text Search” tab.

---

**NOTE:**

Close the Source Viewer now as its search for source lines may interfere with the following measurements.

By using the logic analyzer to capture and store only writes to these two variables, you can observe the duration of the `proc_specific` loop over a large number of executions.

#### 5 Set up the `proc_specific` performance measurement.

The processor trace has been set up to acquire writes to the range of marker variables.

We have set up the system performance analysis tool to calculate the time duration between the pair of writes and to put them in buckets that are 1 millisecond in duration. This will tell us how much variation in the execution time of the `proc_specific` loop there is.

Open the first performance analyzer by selecting `Navigate`, `MPC860 BUS`, and `Performance Analysis<2>`. Move the dialog to the lower right-hand corner of the screen. Close the `Listing` window (if visible) to get it out of the way.

You can see the time buckets down the far left side of the chart in the black area of the dialog. The tool shows what percentage of the marker pairs that were acquired has a duration that falls into each bucket.

## 6 Run the measurement.

Take a trace to see where the execution times fall. Click the green “Run” button. It will take about 50 seconds to get data in the chart. Most of this time is spent collecting the writes to the marker variables.

---

**NOTE:**

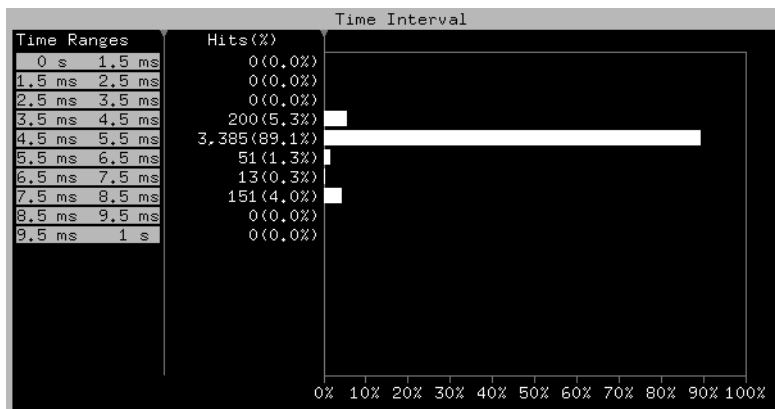
---

DO NOT CLICK THE “STOP” BUTTON. If you already have, then take the trace again by clicking the green “Run” button.

## 7 Display the captured microprocessor bus data.

Look at the chart in the “Performance Analysis” dialog. You will see that most of the time intervals, about 90%, fall in the range

of 4.5 ms to 5.5 ms. About 4% fall into the 7.5 ms to 8.5 ms range.



It is this last group of time duration that we will focus on. They represent a performance degradation of the `proc_specific` loop of 50%. We would like to know what is causing the degradation.

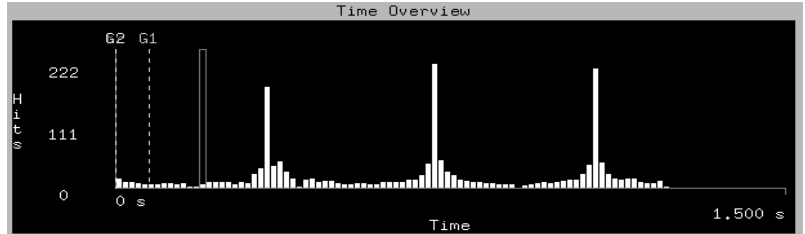
The `proc_specific` loop does not have a lot of different branches that it can execute. It basically executes the same code the same way for every loop. So we suspect that interrupts may be causing the degradation. For a complex system with a lot of interrupts, it might not be easy to determine which interrupt or interrupts were causing the problem. The demo board only has two interrupts active, not counting the interrupt initiated by the button next to the reset button. We will simulate locating this problem with these two interrupts.

## 8 Display the captured interrupt data.

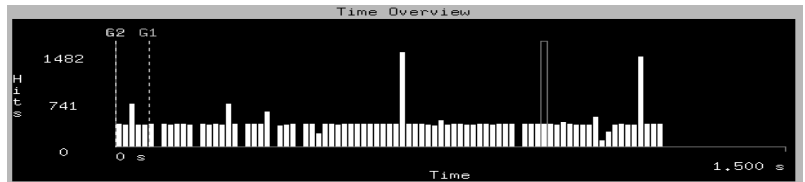
Click the “Navigate” button, select Interrupts >, and select “Performance Analysis<1>...”. This dialog contains a performance analysis of the number of occurrences of IRQ3 over a period of about 1.5 seconds. The X axis of the chart is time, which spans from about 0 sec to about 1.5 sec. This time span has been divided up into 104 buckets and the number of IRQ3s that occurred during that time period (bucket) have



been counted. As you can see IRQ3s sometimes occur very frequently and sometimes very infrequently.

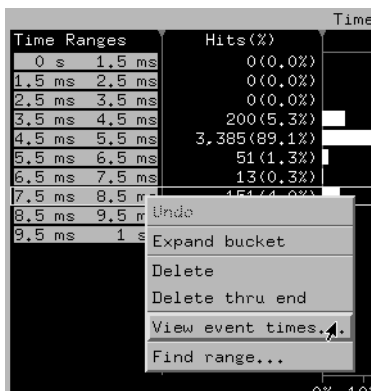


Go to navigate and get Performance Analysis<3>, which contains IRQ2 hits. You can see that IRQ2 is fairly consistent in the number of hits per time bucket.



**9** Use global markers to correlate the captured data.

To determine which IRQ is causing the problem, go to the Performance Analysis<2> dialog, `proc_specific` data, and right-click on the 7.5 ms to 8.5 ms bar and select "View event times...". The dialog that pops up contains all the event times that were collected in that bucket.

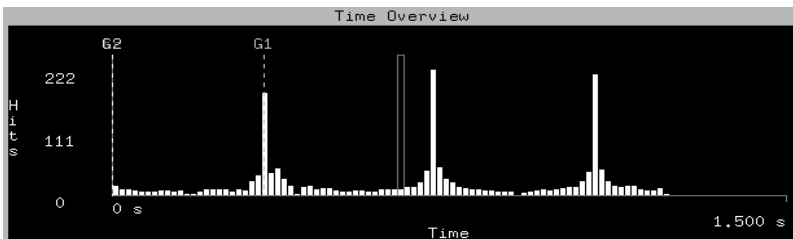


Time Ranges	Hits(%)
0 s 1.5 ms	0(0.0%)
1.5 ms 2.5 ms	0(0.0%)
2.5 ms 3.5 ms	0(0.0%)
3.5 ms 4.5 ms	200(5.3%)
4.5 ms 5.5 ms	3,385(89.1%)
5.5 ms 6.5 ms	51(1.3%)
6.5 ms 7.5 ms	13(0.3%)
7.5 ms 8.5 ms	151(4.0%)
8.5 ms 9.5 ms	
9.5 ms 1 s	

Context menu options: Undo, Expand bucket, Delete, Delete thru end, View event times..., Find range...

Right-click on the first event times, select “Send start time to marker>”, and select “G1”. Look in the dialog that contains the IRQ3 data, Performance Analysis<1>.

You will see that the G1 global marker is right at one, probably the first, of the hit spikes in the IRQ3 data.



If you do the same with the rest of the event times that happen before 1.5 sec, you will see that the G1 marker lands right at one of the spikes in the number of hits. This tells us that it is likely that the performance degradation of proc\_specific is due to the occasional large frequency of IRQ3s.

## Summary

You have just seen how the performance analysis tool can be used to both identify a problem and find the cause.

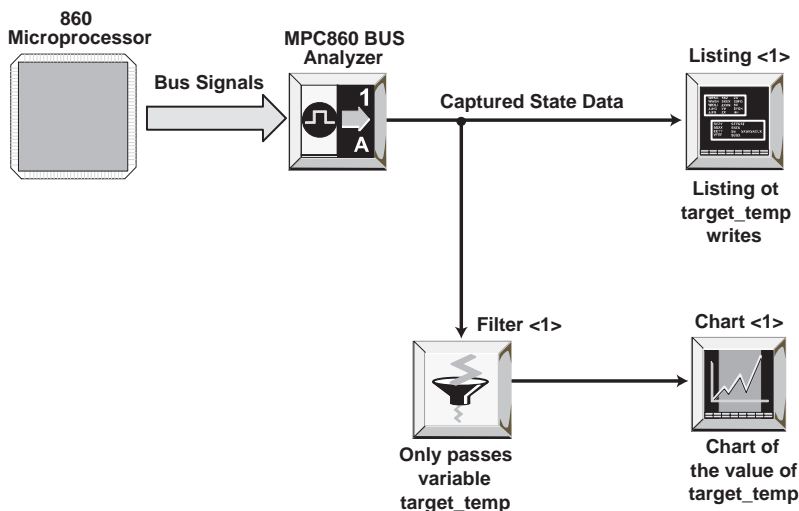
## Using context store

HP state analyzers let you store only certain captured states in trace memory. You use *storage qualifiers* to do this. For example, if you're only interested in the values of a particular variable, you can specify that only writes to that variable are stored.

When tracing a variable's values, you may notice a point where the variable becomes corrupted, and you want to know what code is responsible for the corruption. In this case, you want to view the context in which the bad variable write occurs. In other words, you want to view the execution before and after the bad variable write so you can determine the code responsible for it. HP 1660xA state analyzers have a *context store* feature that lets you do just that.

This exercise begins with storing writes to a variable called `target_temp`. It becomes necessary to get source correlation to see what code is executing when a problem happens. It is not possible to get source correlation without some inverse assembly in the trace and there is none in this trace. By turning on context store, enough bus cycles are captured to get inverse assembly and source correlation.

This measurement is setup so that a listing of the trace is generated and a chart, or graphical representation of the trace, is generated. The trace data is filtered before it is charted.



**1** Probe the MPC860 demo board.

Follow the instructions in the “Getting Started” chapter on page 9 for connecting the logic analysis module to the demo board.

**2** Load the configuration files for this demo.

In the main logic analysis system window, click the File Manager button.



In the File Manager dialog, select the Context,\_\_\_ configuration file from the /hplogic/demo/860\_demo\_board/configs/hp1660x directory, and click the Load... button.

In the Load Configuration dialog, click Load.

If a confirmation dialog appears, click Yes.

(You can also choose the File, Load Configuration... command from the menu bar of most windows.)

**3** Make sure the MPC860 demo board processor is running in a known state.

Click the “Reset”, “Break”, and “Run” buttons in order.



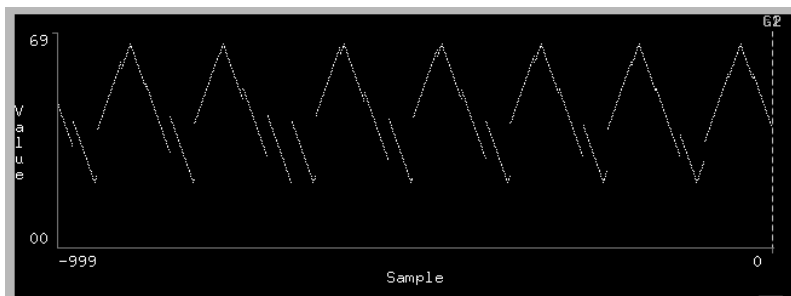
Resetting the processor starts it from a known state. Breaking puts the processor into the background monitor and allows the emulation module to read the microprocessor’s configuration registers (whose contents enable breakpoints among other things).

**4** Run the measurement.

Click the green Run button in the Listing<1> dialog. The trace will take about 15 seconds because we are only capturing writes to the target\_temp variable. We are capturing 1000 of them and they only occur on average every 14 milliseconds.

## 5 Display the captured data.

Target\_temp should ramp up and down consistently over and over again, but it does not. This can be seen with the chart tool. Select Navigate, MPC860 BUS, and Chart<1>.

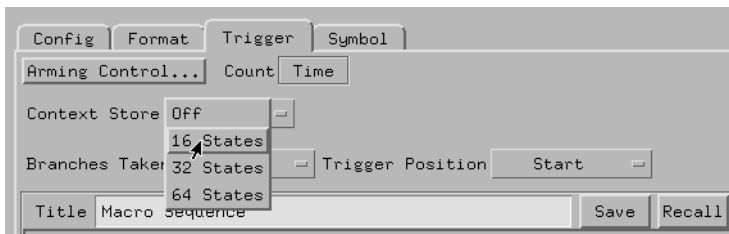


As you can see, the value of target\_temp does not gracefully rise and fall as it was programmed to do.

## 6 Turn on context store.

It would be useful to see what code was executing when one of the target\_temp discontinuities occurs. To do this, more cycles of the bus trace must be captured around the writes to target\_temp. Turning on context store will accomplish this.

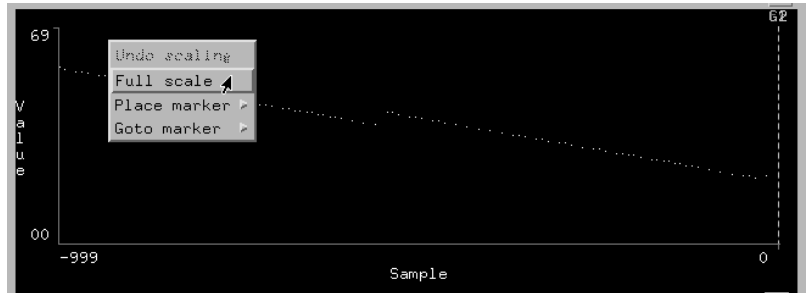
Select Navigate, MPC860 BUS, Setup, and the trigger tab of setup. Select the Off next to “Context Store,” and select 16 states from the pull-down.



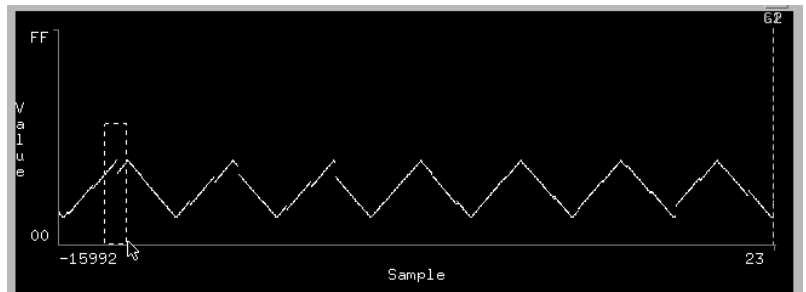
## 7 Run the measurement again, and view the captured data.

Lower the setup dialog and click the green Run button. Again the trace will take several seconds to happen.

It will be necessary to rescale the chart dialog. Right-click in the middle of the black part of the chart dialog and select Full scale.



Now, zoom in on one of the discontinuities. Left-click just to the left of the discontinuity and drag to the right of it. A rectangle will form around the area that will be zoomed to. If you do not get the zoom you want, you can left-click and undo the scaling.



Zoom in again so that you can see the individual points representing individual target\_temp writes.

**8** Use a global marker to locate a bad variable value.

Now place a global timing marker on one of the discontinuities. Left-click, select Place marker >, and select G1. Drag the marker so that it is directly on the discontinuity.



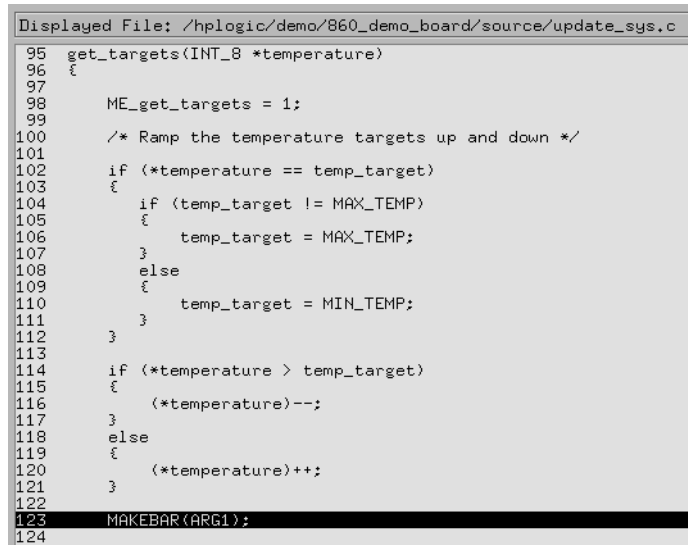
In the Listing window, scroll up from this point and look at the yellow writes, which are writes to `target_temp`; they will be incrementing or decrementing. Go back to the marker and you will see that the write is not in sequence with the writes you just looked at.

To understand what is happening, you need to look at the source code. But first scroll back to one of the writes that is in sequence.



## 9 Open the Source Viewer window.

Select Navigate, MPC860 BUS, and Source Viewer <1>. You will be close to line 120 of the update\_sys.c source. This is where the writes to target\_temp were coded to happen.



```
Displayed File: /hplogic/demo/860_demo_board/source/update_sys.c
95  get_targets(INT_8 *temperature)
96  {
97
98      ME_get_targets = 1;
99
100     /* Ramp the temperature targets up and down */
101
102     if (*temperature == temp_target)
103     {
104         if (temp_target != MAX_TEMP)
105         {
106             temp_target = MAX_TEMP;
107         }
108         else
109         {
110             temp_target = MIN_TEMP;
111         }
112     }
113
114     if (*temperature > temp_target)
115     {
116         (*temperature)--;
117     }
118     else
119     {
120         (*temperature)++;
121     }
122
123     MAKEBAR( ARG1 );
124 }
```

What you will see is an incrementing or decrementing of \*temperature. \*temperature is a parameter to the function get\_targets. The argument use for this parameter when get\_targets was called was target\_temp. So when \*temperature is incremented or decremented, it is target\_temp where the new value is stored.

Notice also that the writes are happening exclusively in the function get\_targets. You can establish this by looking at the address symbols in the PC column.

Now, in the Listing window, scroll down to the discontinuity. The Source Viewer jumps to line 300 of the code. You are now

in the function `save_pointers`, which should never write to `target_temp`.

```
Displayed File: /hplogic/demo/860_demo_board/source/update_sys.c
290 * References: None.
291 *
292 * Returns: Nothing.
293 *****
294 void
295 save_pointers()
296 {
297
298     ME_save_points = 1;
299
300     old_data[curr_loc].c_temp = current_temp;
301     old_data[curr_loc].t_temp = target_temp;
302     old_data[curr_loc].f_needed = func_needed;
303     curr_loc++;
304
305     if (curr_loc > NUM_OF_OLD)
306     {
307         curr_loc = 0;
308     }
309
310     /* On the last write before setting "curr_loc" to 0, "curr_loc"
311        is equal to the array size "NUM_OF_OLD", thus causing a write
312        past the end of the array into "target_temp" */
313
314     MAKEBAR(ARG3);
315
316     MX_save_points = 1;
317 }
318 }
319
```

What is happening here is the pointer `curr_loc` has written past the end of the array that was set up to store the old temperatures. This caused it to write to the variable `target_temp`, something that should only have happened in `get_targets`.

```
Displayed File: /hplogic/demo/860_demo_board/source/ecs2.c
35
36 /*****
37 * Global Variable Declarations
38 *****/
39
40 old_el   old_data[NUM_OF_OLD]; /* History of temp data. */
41 INT_8   target_temp; /* Target temperature. */
42 INT_8   d_temp; /* place holders */
43 char    d_func;
44 INT_8   temp_target;
45 char    ascii_old_data[LISTLEN][16];
46 /* ASCII values of temp data. */
47 char    history_buffer[HIST_LENGTH]; /* History of control outputs */
48
```

## Summary

The context store feature lets you easily find the code that is responsible for corrupting a variable.

## Tracking processor execution with caches turned on

The MPC860 analysis probe provides a special execution trace inverse assembler that identifies branch trace messages generated by the MPC860 processor. This allows the Source Viewer to track source code even when the processor is executing out of cache memory.

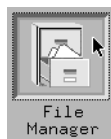
For example, some of the MPC860 demo board's code is set up to execute out of cache memory. You can capture this execution, display it using the two kinds of inverse assemblers, and compare the Listing and Source Viewer displays.

### 1 Probe the MPC860 demo board.

Follow the instructions in the “Getting Started” chapter on page 9 for connecting the logic analysis module to the demo board.

### 2 Load the configuration files for this demo.

In the main logic analysis system window, click the File Manager button.



In the File Manager dialog, select the Cache\_off.\_\_\_\_ configuration file from the /hplogic/demo/860\_demo\_board/configs/hp1660x directory, and click the Load... button.

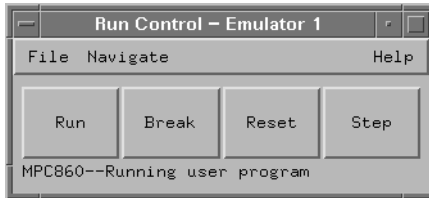
In the Load Configuration dialog, click Load.

If a confirmation dialog appears, click Yes.

(You can also choose the File, Load Configuration... command from the menu bar of most windows.)

- 3 Make sure the MPC860 demo board processor is running in a known state.

Click the “Reset”, “Break”, and “Run” buttons in order.

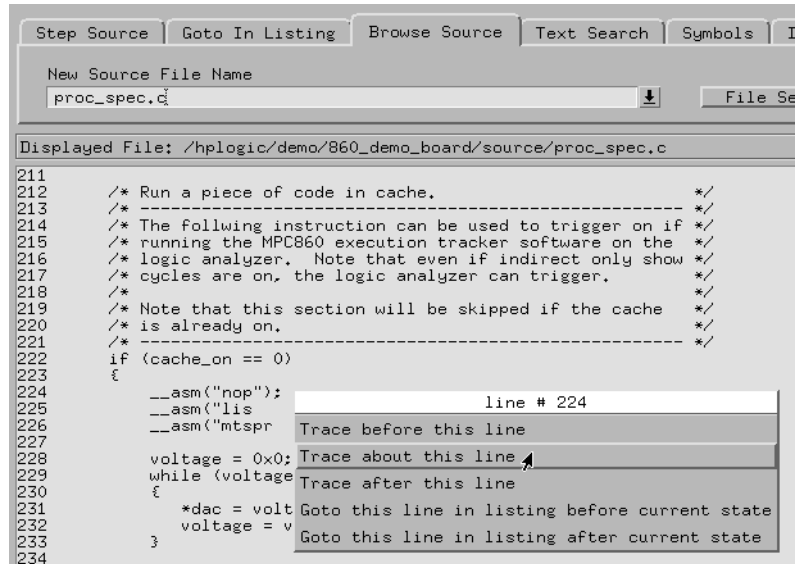


Resetting the processor starts it from a known state. Breaking puts the processor into the background monitor and allows the emulation module to read the microprocessor’s configuration registers (whose contents enable breakpoints among other things).

- 4 With the standard inverse assembler, look at executed code with caches turned on.

To set the analyzer trigger, go to the source viewer, select the “Browse Source” tab, select “File Selection...”, and select the file `/hplogic/demo/860_demo_board/source/proc_spec.c`. Scroll down to line 212. This is the beginning of the code that executes with cache on.

Set the analyzer to trigger about line 224, the NOP, and run the analyzer.



## 5 Display the captured data

When the trace has been taken, go to the “Listing<1>” dialog, and then go to the trigger point in the listing. You will see the assembly code being executed that turns off the cache. After that, things get strange. You will see the assembly code again because it was fetched into cache, and very shortly after that you will only see reads and writes to the D/A converter. Once the cache is turned on, following the code on the bus does not work very well.

Note also that the Source Viewer is unable to track the source code through this part of the inverse assembly.

State Number	PC	MPC821/860 Inverse Assembler
Decimal	Symbols	10=hex, 10.=decimal, %10=binary
0	proc_specifi+0310	nop
4	proc_specifi+0314	lis r20 0200
8	proc_specifi+0318	mtspr ic_cst r20
12	proc_specifi+031C	li r6 00000000
16	proc_specifi+0320	b p:proc_specific+0338
20	proc_specifi+0314	lis r20 0200
24	proc_specifi+0318	mtspr ic_cst r20
28	proc_specifi+0338	cmpwi cr0 r6 0080
32	proc_specifi+033C	blt cr0 p:proc_specific+00
36	proc_specifi+0340	li r6 00000080
40	proc_specifi+0334	andi r6 r6 00FF
44	proc_specifi+0324	lis r12 0000
48	proc_specifi+0328	lwz r10 4014(r12)
52	proc_specifi+032C	stb r6 0000(r10)
56	proc_specifi+0330	addi r6 r6 000A
60	/source/q,elf:dac	read 50xxxxxx
61	ce/q,elf:dac+0001	read 00
62	ce/q,elf:dac+0002	read 00xx
63	ce/q,elf:dac+0003	read 00
64	D/A control (CS5)	write 00
65	/source/q,elf:dac	read 50xxxxxx
66	ce/q,elf:dac+0001	read 00
67	ce/q,elf:dac+0002	read 00xx
68	ce/q,elf:dac+0003	read 00
69	D/A control (CS5)	write 0A

**6** Load the configuration files that use the execution trace inverse assembler.

Now load the configuration file “/hplogic/demo/860\_demo\_board/configs/Cache\_on.\_\_\_\_”.

This configuration is set up to use the execution trace inverse assembler.

The execution trace inverse assembler uses the MPC860’s branch trace messaging feature to follow processor execution when the caches are turned on.

**7** With the execution trace inverse assembler, look at executed code with caching turned on.

Go to the source viewer, select the “Browse Source” tab, select “File Selection...”, and load the source file /hplogic/demo/860\_demo\_board/source/proc\_spec.c.

Go to line 224 and set the trigger again.

Click the green Run button to perform the measurement; then, go to the “Listing<1>” dialog.

State Number	SW_ADDR	MPC821/860	
Decimal	Symbols	Cache-On Execution Tracker	
0	proc_specifi+0310	Instruction	Execution
16	proc_specifi+0314	Instruction	Execution
32	proc_specifi+0318	Instruction	Execution
53	proc_specifi+031C	Instruction	Execution
101	proc_specifi+0320	Instruction	Execution
102	proc_specifi+0338	Instruction	Execution
150	proc_specifi+033C	Instruction	Execution
151	proc_specifi+0324	Instruction	Execution
199	proc_specifi+0328	Instruction	Execution
202		4 byte Data	Read = 504123D7H Address = ard/source/q.elf:dac
205		1 byte Data	Read = 004123D7H Address = ource/q.elf:dac+0001
208		2 byte Data	Read = 004123D7H Address = ource/q.elf:dac+0002
211		1 byte Data	Read = 004123D7H Address = ource/q.elf:dac+0003
216	proc_specifi+032C	Instruction	Execution
235		1 byte Data	Write = 00H Address = D/A control (CS5)
240	proc_specifi+0330	Instruction	Execution
244	proc_specifi+0334	Instruction	Execution
248	proc_specifi+0338	Instruction	Execution
252	proc_specifi+033C	Instruction	Execution
254	proc_specifi+0324	Direct Branch	Taken
258	proc_specifi+0328	Instruction	Execution
263		4 byte Data	Read = 504123D7H

You will now see the branch trace messages.

In the Source Viewer, you will be able to follow what source code was executed during this trace.

In this example, the section of code that executes out of cache is very short, but it demonstrates how source code could be tracked through a long section of code executing out of cache.

### Summary

The execution trace inverse assembler makes it possible for the logic analysis system to provide source code correlation even when the processor is executing out of cache memory.





---

**Quickly Find the Cause of  
Difficult Hardware Problems**

## Capturing Very Deep Traces

Being able to capture very deep traces gives you the ability to capture widely separated causes and effects. You can also capture a large number of repeating events to help track down problems that might occur infrequently or to establish statistics of the event.

- By using logic analyzers that give you deep-memory traces, and by using the different types of filtering and display tools, you can capture a large amount of data (perhaps of a rarely occurring problem) and view the data in different ways without having to recapture it.

---

## Using logic analyzers with deep memory

With HP's deep-memory logic analyzer modules, you can easily capture a trace of long duration and search it for a widely separated problem and its root cause.

For example, with the HP 16600A state/timing logic analyzer module and 64K states of trace memory, you can capture 10 milliseconds of MPC860 demo board (25 Mhz) execution.

By contrast, you can capture 98 microseconds of MPC860 demo board execution with the HP 16550A state/timing logic analyzer module and 4K states of trace memory.

### 1 Probe the MPC860 demo board.

Follow the instructions in the "Getting Started" chapter on page 9 for connecting the logic analysis module to the demo board.

---

#### NOTE:

Make sure the pattern generator data pods are not connected to the demo board; otherwise, the changes in demo program execution will affect the results of the measurements that follow.

---

### 2 Load the configuration files for this demo.

In the main logic analysis system window, click the File Manager button.



In the File Manager dialog, select the Deep\_trace\_ configuration file from the /hplogic/demo/860\_demo\_board/configs/hp1660x directory, and click the Load... button.

In the Load Configuration dialog, click Load.

If a confirmation dialog appears, click Yes.

(You can also choose the File, Load Configuration... command from the menu bar of most windows.)

The Waveform<1> dialog that is brought up is set up to produce a timing trace of the microprocessor on the demo board. As you can see, the address and data buses are traced, as well as various status lines on the processor.

The analyzer has been set up to trace 64K acquisitions of the 51 processor signals. As you will see, this is 1.5 msec of execution (at a sample rate of 8 ns).

For large traces such as this, 408 Kbytes, HP uses a demand-driven approach, which significantly speeds up the display of that data. Other logic analyzers will get bogged down displaying less data than is captured in this exercise.

### **3** Run the measurement.

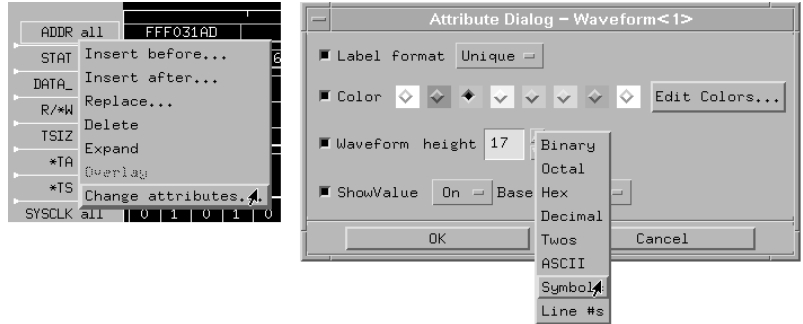
Click the green Run button to take a trace.

After the acquisition has completed, try moving around in the trace. Click on the scroll bar (because of the large trace, it is more of a dot than a bar), and drag it to the right or left. Notice how responsive the display is. This is a trace of bus cycles occurring during the execution of code on the demo board.

### **4** Display symbols in the captured data.

Symbols for the 860 processor were loaded into the analyzer with the configuration file you loaded. These symbols can be used in the trace as well. Right-click on “ADDR all” and select

“Change attributes...”. On the ShowValue line select “Hex”, pick “Symbols” from the pull down, and select “OK”.



The symbols for address that have been defined are displayed in the trace. To see more of the symbols, you can expand the display by decreasing the seconds per division.

**5** Use the Goto tab to quickly go to certain locations.

You also have navigation aids. Select the “Goto” tab. Here you can quickly go to the beginning, trigger, or end of the trace.

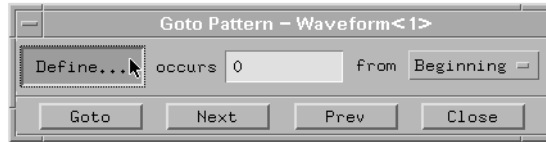


You can also place markers in the trace and go to them quickly.

**6** Use the search capability to find particular data.

To place a marker at a specific address, the search feature can be used. We wish to find the first occurrence of an access to the

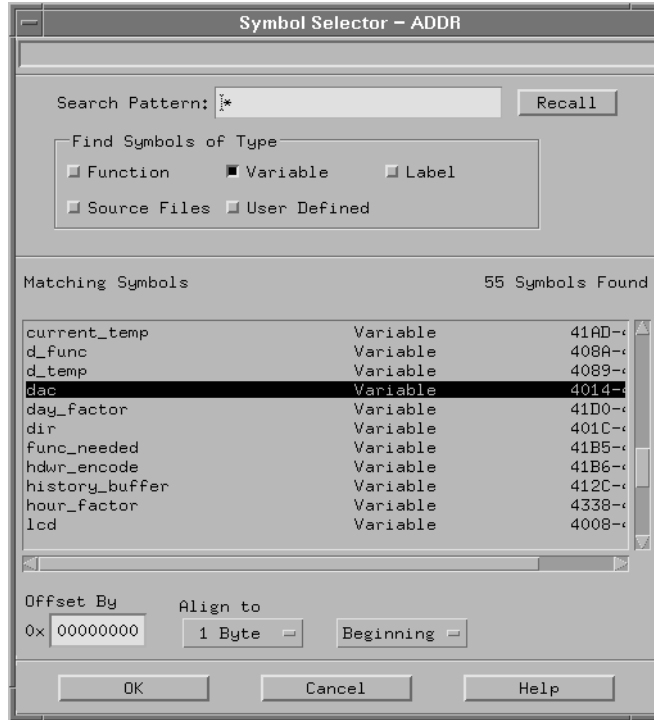
DAC. Select the “Search” tab, select “Advanced searching...”, and in the dialog that comes up select “Define...”.



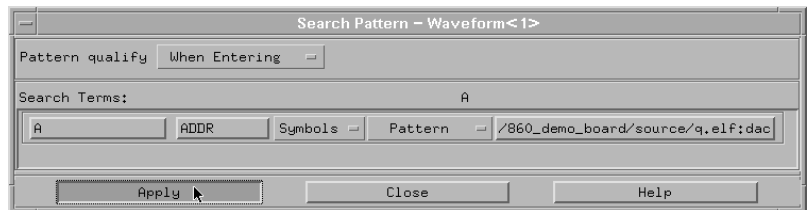
We want to search for a symbol, so in the Search Pattern dialog select “Hex” and change it to “symbols”. Expand the dialog and select “Absolute XXXXXXXX”. The dialog that comes up shows the list of symbols read into the analyzer.

To narrow down the list deselect all of the types in the “Find Symbols of Type” box except for the “Variable” type. Click in

the list box and scroll down until you find the “dac” symbol. Select it and click OK.

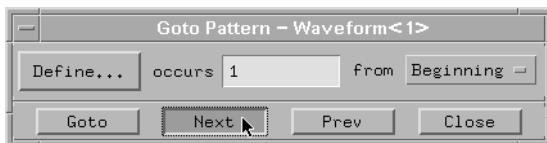


In the “Search Pattern” dialog, select “When Present” and change it to “When Entering”, then select “Apply”.



The waveform display has now gone to the 0th occurrence of the DAC address from the beginning, which is the beginning of the trace.

In the “Goto Pattern” dialog, select “Next”.



The trace will center on the first occurrence of an access to the DAC.

## 7 Use markers for easy return to points in the data.

Should you want to return to this position at a later time without using the search, you can place a marker here. Right-click within the ADDR trace where you want the marker placed. Select “Place marker >” and select “G1”. If the marker does not go exactly where you want, you can drag it into place.



One point is worth noting. If you compress the display, select the up arrow next to “Second/div”, more and more of the data acquired is being displayed. The more data displayed, the slower the trace will scroll.

## Summary

By using logic analyzers that give you deep-memory traces, and by using the different types of filtering and display tools, you can capture a large amount of data (perhaps of a rarely occurring problem) and view the data in different ways without having to recapture it.



---

**About the MPC860 Demo Board**

## Demo Board Hardware

### Introduction

The MPC860 demo board is a Motorola MPC860 PowerPC embedded system. It is designed for easy interface to an HP 16600A/16700A-series logic analysis system. The features built into the demo board were chosen to demonstrate a variety of problems that can be solved by the logic analysis system. For ease of use, the demo board draws power from a single, logic analysis module pod or from an emulation module or probe. The demo board does not need to be configured, all features and problems are active whenever the processor is running.

In this section the following is discussed:

- how to configure the logic analysis system for use with the demo board
- what signals are mapped to the connectors
- what features are found on the demo board
- the firmware running on the demo board

### Configuring the Logic Analysis System for the Demo Board

The easiest way to configure the analysis system for the demo board is to run the Setup Assistant. This is described in “Tracing Processor Code Execution with Source Code Correlation” on page 22.

For completeness, the required configuration is described here as well.

## Labels and Format

The following labels and format are needed for inverse assembly. It should be noted that the demo board uses an 8 bit data bus to keep the number of analyzer pods required down to four. To use the existing 6-pod inverse assembler, some overlap is necessary between the DATA and STAT labels to make the 4-pod solution work.

Label	CLK	Pod 4	Pod 3	Pod 2	Pod 1
ADDR	----	-----	-----	*****	*****
DATA	----	*****	*****	-----	-----
STAT	****	-----*	*****	-----	-----

## Clocking

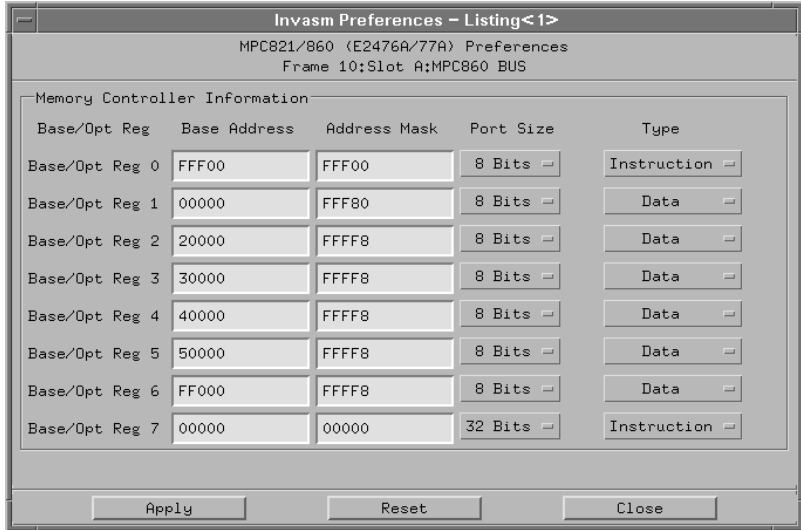
**Standard inverse assembler.** When using the standard inverse assembler, the state analysis clock qualifier should be set to clock on the rising edge of the J (SYSCLK) and K equal to zero (TA asserted), that is  $(J^{\wedge}) * (K=0)$ .

**Execution trace inverse assembler.** When using the cache-on execution trace inverse assembler, the state analysis clock qualifier should be set to clock on the rising edge of the J (SYSCLK), that is  $(J^{\wedge})$ .

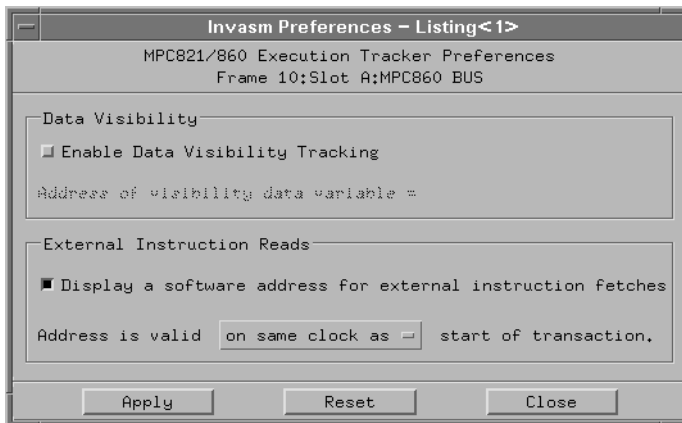
## Inverse Assembler Preferences

The MPC860 processor does not provide status information as to whether it is fetching data or instructions. This information is necessary for the proper functioning of the inverse assembler. The user tells the inverse assembler where data and instructions are in the Preferences display. This display can be reached by going to the menu bar of the listing display for the MPC860 BUS analyzer. Select Invasm and then Preferences.

**Standard inverse assembler.** The Preference display should look like this:



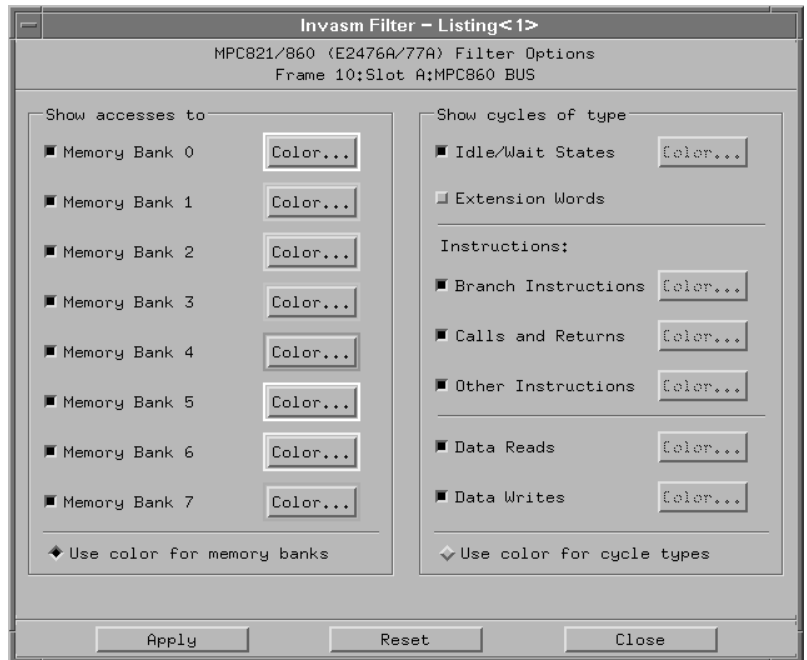
**Execution trace inverse assembler.** The Preference display should look like this:



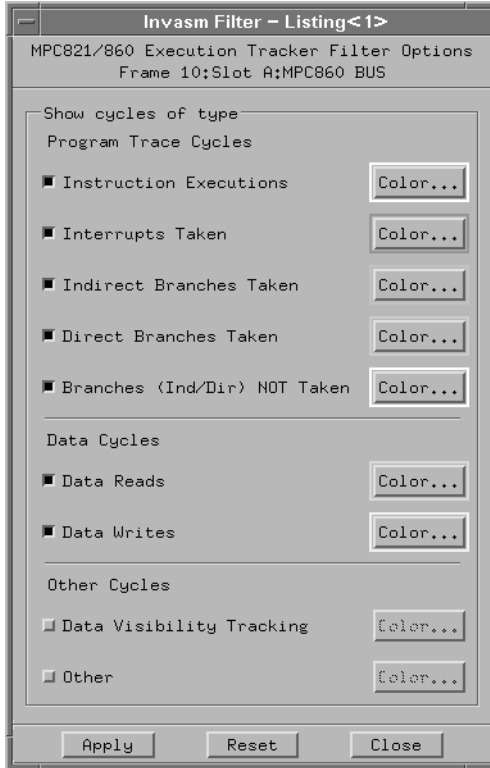
## Inverse Assembler Filter

Filtering help to make the listing easier to read by suppressing some cycles and displaying others in color.

**Standard inverse assembler.** The demo board only has an 8 bit data bus which results in three extra bus cycles each time a data or instruction word is fetched. The filter can be set to suppress these cycles. In addition, the filter can be set to display different cycles or different areas of memory in different colors. Set the filter display to suit your tastes, the following is only an example.



**Execution trace inverse assembler.** The following is just an example, set the Filter display to suit your tastes:



## Demo Board Connector Mapping

There are five sets of built-in connections on the demo board for connecting various system modules.

### Emulation Module Connector

This connector provides a connection to either an emulation module or probe for run control of the processor. Normally this would not be a direct connection. A Target Interface Module (TIM) would be used to connect to the target's Background Debugger Mode connection or JTAG connection and then to

the emulation module. However, to eliminate the need for a TIM during demos, it has been built in.

### **Logic Analyzer Connectors**

There are five low-density pod connections around the bottom of the demo board and six pod connections in the three Mictor38 connectors on the top of the demo board. The four listed below allow the analysis module to trace the bus cycles of the MPC860 processor.

<b>Data</b>	<b>Pod 1: ADDR</b>	<b>Pod 2: ADDR</b>	<b>Pod 3: STAT</b>	<b>Pod 4: DATA</b>
15	A16	A0	VFLS0	D0 - Data Bus - MSB
14	A17	A1	VFLS1	D1
13	A18	A2	AT2	D2
12	A19	A3	VF2	D3
11	A20	A4	VF0	D4
10	A21	A5	VF1	D5
9	A22	A6	$\overline{CS5}$	D6
8	A23	A7	$\overline{CS4}$	D7 - LSB (Only 8 bits used)
7	A24	A8	$\overline{CS3}$	MEM_OE
6	A25	A9	$\overline{CS2}$	MEM_WE
5	A26	A10	$\overline{CS1}$	unused
4	A27	A11	$\overline{CS0}$	unused
3	A28	A12	$\overline{BURST}$	unused
2	A29	A13	unused	unused
1	A30	A14	R/ $\overline{W}$	unused
0	A31 - LSB	A15	$\overline{TS}$	MODCK1
CLK	SYSCLK - 25MHz	$\overline{TA}$	TSIZ0	TSIZ1

Appendix A: About the MPC860 Demo Board  
**Demo Board Hardware**

The following two pods are for timing traces of features on the demo board.

<b>Data</b>	<b>Pod 5: Timing Probe</b>	<b>Pod 6: Pattern Generator Probe *</b>
15	PORESET	PG15
14	HRESET	PG14
13	SRESET	PG13
12	IRQ5	PG12
11	IRQ4	PG11
10	IRQ3	PG10
9	IRQ2	PG9
8	IRQ1	PG8
7	CAN_RXD	PG7
6	CAN_TXD	PG6
5	CAN_CLK	PG5
4	DO_BOUNCE	PG4
3	unused	PG3
2	STEP_UNTERM	PG2
1	unused	PG1
0	unused	PG0
CLK	BOUNCE_LATCH	unused

\* Pod 6 is only available on the Mictor38 Connector.



## HP 16517A High-Speed Timing Connector

Pin	P7 - HP 16517A Connector
10 Top R	DO - Data Bus 0
9	DO_DEL - 10 ns Delayed DO
8	D1
7	D2
6	D3
5	$\overline{\text{CS}}\text{T}$ - RAM Chip Select
4	$\overline{\text{MEM\_WE}}$
3	$\overline{\text{MEM\_OE}}$
2	R/ $\overline{\text{W}}$
1 Top L	

## Pattern Generator Connectors

Pin	Pattern Generator - PG1	Pattern Generator - PG2
7	LCD_DATA0 - MSB	$\overline{\text{LCD\_SELECT}}$
6	LCD_DATA1	LCD_RS
5	LCD_DATA2	LCD_R/W
4	LCD_DATA3	unused
3	LCD_DATA4	unused
2	LCD_DATA5	unused
1	LCD_DATA6	unused
0	LCD_DATA7 - LSB	PG_EN

1 - PG Disabled, 0 - PG Enabled

## Oscilloscope Connections

There are six scope connections on the demo board. They are used for probing various analog signals on the board. The

connections are as follows:

Label	Signal
DO BOUNCE	DO line on latch
Latch	Clock input of latch
GND BOUNCE	Ground of latch
D/A	Output of D/A converter
STEP	1 MHz signal with reflections
CAN BUS	Serial output of CAN controller

In addition, there are two ground connections for the scope probe ground clips.

## Demo Board Features

### Main Components

#### CPU.

- Motorola MPC860 Embedded PowerPC™ Microprocessor
- 25 MHz operation
- Programmed in 8-bit mode for minimal power consumption

#### Memory Map.

Chip Select	Device	Base Address	End Address	Size
$\overline{CS0}$	Flash ROM	0xFFF00000	0xFFFFFFFF	1 Mbytes
$\overline{CS1}$	SRAM	0x00000000	0x0007FFFF	512 Kbytes
$\overline{CS2}$	LCD	0x20000000	0x20000001	2 Bytes
$\overline{CS3}$	CAN Controller	0x30000000	0x300000FF	256 Bytes
$\overline{CS4}$	PLD	0x40000000	0x4000000F	16 Bytes
$\overline{CS5}$	D/A Converter	0x50000000	0x50000000	1 Byte
CPU	CPU Space	0xFF000000	0xFF003FFF	16K Bytes

### **Flash ROM.**

- AMD M29F800B 1MBx8 Flash ROM
- 16 erasable sectors
- Embedded Erase/Program Algorithms
- 5 Volt Operation
- 1MB Flash ROM partitioned into protected and user blocks.

### **SRAM.**

- Sony HM6285 512K x 8 70ns SRAM
- 512K SRAM for downloading demo code from 3rd party debuggers.

### **LCD.**

- 2 line, 16 character display
- Optrex DMC16249

### **PLD.**

- Altera EPM7128SQC100-15
- In-Circuit Programmable
- Contains:
  - Memory Interface
  - Interrupt Controller
  - Pattern generator interface controller
  - Hardware control registers

## **Features and Problems**

The firmware constantly stimulates the features that need it so they are always ready for use.

**Variable frequency interrupt subsystem.** IRQ3 is generated by this system. It produces interrupts in varying

frequencies, with interrupts coming in a flurry, subsiding to occurring very infrequently, and then coming in a flurry again. This is intended to simulate the kind of interrupts that might occur with a LAN controller. This feature deliberately interferes with the waveform produced by the D/A converter in an asynchronous way.

**LCD display with pattern generator connections.** The LCD display provides feedback to the user and is driven by the processor. In addition, it is connected to the pattern generator connections through a multiplexer. When the pattern generator is connected, it can override the processor's control of the display. Vectors can be created in the pattern generator that will write to the LCD display.

---

**NOTE:**

The pattern generator by default takes control of the LCD display. When this happens, the processor will make 2,000 attempts at writing to the display before going to the following code. Needless to say, this significantly slows down the execution of the code.

**Latch with a ground bounce problem.** The processor writes to a latch with its ground isolated by a coil. When only a few bits change value, all is well. However, if most of the bits change value, the resulting ground bounce prevents the correct value from being written to the latch.

**D/A Converter.** The D/A converter is a TI TLC7528, which has 100ns settling time. The firmware stimulates the D/A converter twice each time it loops. The first time produces a triangular waveform, the second time caching is turned on and a smaller triangular waveform is produced. The triangular waveforms are distorted by IRQ2 interrupts (CAN controller) and occasionally by IRQ3 interrupts. Once in a great while the IRQ3 effect dominates the triangular waveform.

**CAN Controller.** The demo board includes an Intel AS82527 CAN Bus Communications Controller. The Controller Area Network (CAN) bus is primarily used by the automotive industry. This controller supports CAN 2.0 protocol. The controller provides a good example of how well the analysis

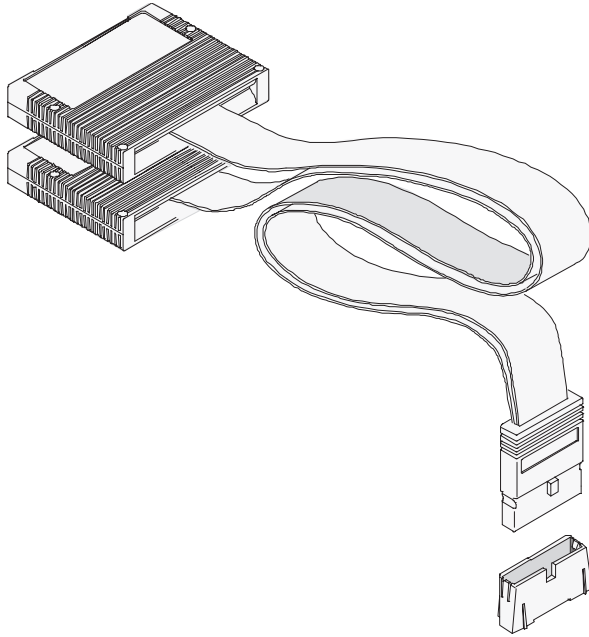
system works with external system buses. In addition, it provides an opportunity to demonstrate the serial analysis tool.

The firmware programs the controller to send out information from the Environmental Control System function each time it loops. However, CAN controllers require an acknowledge from another controller to complete a transmission. This gets simulated by the PLD, it counts the number of bytes sent by the CAN controller and then shuts it down. However, the controller is able to retransmit one more time before stopping. The IRQ2 from the CAN controller get serviced by a routine that simulates getting data from a controller that received the message.

**Pushbutton.** The pushbutton on the front panel labeled “Interrupt” generates an IRQ1. It can be used to generate an asynchronous event that can be detected by the analysis system.

**Mictor38 probing technology.** In addition to the low-density pod connections around the bottom of the demo board, there are three Mictor38 connectors on the top of the demo board. HP E5346A adapters are available to provide the connections to the analyzer pods. These connectors duplicate the low-density connections and are there primarily to demonstrate HP’s high-density connections. The footprint of these connections is reduced in two ways, the connector is smaller and no termination circuitry is required. The termination circuitry for the low-density connectors can be seen as a collection of surface mount resistors and capacitors next to the connectors.

Appendix A: About the MPC860 Demo Board  
Demo Board Hardware



## Demo Board Firmware

### Introduction

The firmware on the 860 demo board boots the processor and initializes the board. It then prepares to execute a simulation of an Environmental Control System (ECS) and to execute code that stimulates the features and bugs built into the demo board. Then it continuously loops through the ECS and stimulus code.

### Overview of main()

The function main() is located in eps2.c. It boots the processor, performs some initialization and then continuously loops through updating the display and two main functions, update\_system (ECS code) and proc\_specific (stimulus code). The variable num\_checks is a pass counter used by update\_system and update\_display.

### Functions and operations executed by main()

Initialization	<code>boot_q();</code> <code>init_system();</code> <code>proc_spec_init();</code>
Continuous Loop	<code>update_system(num_checks);</code> <code>num_checks++</code> <code>update_display(num_checks);</code> <code>proc_specific();</code>

## What the Functions do

<code>boot_q()</code>	<ul style="list-style-type: none"><li>• Initializes/sets up demo board (processor, PLD, etc.)</li><li>• Calls <code>can_init()</code>, <code>lcd_init()</code></li><li>• Flashes LED and puts "HP Demo Board c1997" on LCD</li><li>• Calls <code>pv()</code> ;Only leaves <code>pv()</code> if all is well.</li><li>• Stops LED flash and puts "ECS II Ver 1. Initializing" on LCD</li></ul>
<code>init_system()</code>	Initializes the environmental control system. The variables are initialized within this procedure so that the system can reboot without being reloaded.
<code>proc_spec_init()</code>	Clear out the problem variable
<code>problem = 0;</code>	used to cause run-time problems CHECK ON THIS, MAY NOT BE USED
<code>main_interrupts = 0;</code>	0 - No interrupts in <code>main()</code> Set 1 for interrupts to occur in <code>main</code>
<code>periodic_interrupts = true;</code>	
<code>cache_on = 0;</code>	The cache is not on in <code>proc_specific</code>
<code>main_cache = 0;</code>	No effect since cache is disabled



update_system()	<p>ECS Update System Code - Modified for Demo Board. It is the service routine that alters the state of the entire environmental control system. It calls several functions, each of which have particular parts of the system that they alter or update. The following action is taken when this routine is called.</p> <ol style="list-style-type: none"><li>1. New temperature target is read in.</li><li>2. New environment conditions are read.</li><li>3. The func_needed is modified based on the actual state of the environment versus the desired state to indicate what needs to happen in the current environment.</li><li>4. func_needed is used to derive hdwr_encode (the 16-bit quantity that indicates what the hardware needs to do to achieve the correct change in the environment).</li><li>5. The environment conditions are saved for posterity.</li><li>6. THERE IS A BUG IN THIS ROUTINE (ON PURPOSE!!!)</li></ol>
update_display()	<p>Clear out the history buffer and update the current ascii display of operating data (ascii_old_data).</p>
proc_specific()	<p>Proc_specific exercises all of the features and bugs built into the demo board. There is a detailed description of what proc_specific does below.</p>

## Where the functions are located

Function	Location
main()	ecs2.c
boot_q	boot.c
can_init	can.c
init_system	init_sys.c
lcd_init	lcd.c
proc_spec_init	proc_spec.c
proc_specific	proc_spec.c
pv	pv.c
update_display	ecs2.c
update_system	update_sys.c

## Overview of proc\_specific

Turn on the cache if cache\_on = 1

Start up the periodic interrupts

Display ECS status on the LCD

## Stimulate the Controller Area Network (CAN) Bus

The following data will be sent out the CAN bus. Each variable is 8-bits, with an 8-bit label that precedes it in the data transmission. With the serial analyzer, look for the 0x11, 0xaa, 0x22, 0xbb, etc. to know which variable's data follows.

## Stimulate the D/A Converter

The D/A converters output voltage is 5 volts times D/256.  
Create a triangle wave by ramping up the voltage one step at a time, then immediately ramping it down.

## Stimulate the ground bounce register

The demo board has register with a ground bounce problem.  
Connect the scope to the 'D0 Bounce' and 'Latch' Signals.

Trigger the scope on the rising edge of 'latch' with a threshold of 1.5 volts and a sample rate of at most 5ns. Note the change in 'D0 Bounce' when data line changes.

### **Run a piece of code in cache**

The following instruction can be used to trigger on if running the MPC860 execution tracker software on the logic analyzer. Note that even if indirect only show cycles are on, the logic analyzer can trigger.

Note that this section will be skipped if the cache is already on.

### **Return to normal ECS code**

Before returning, check if any of the execution flags have been set - if `main_interrupts = 0`, Turn off interrupt activity before leaving function. If `main_cache = 0`, Turn off the cache for main.

## **Variables**

### **Variables of interest**

The following are some of the key variables in the demo board firmware.

<b>Variable</b>	<b>Remarks</b>	<b>See Also</b>
<code>ascii_old_data</code>	ASCII values of temperature data. It is used to show the value of a debugger to monitor complex C variables. Historically, 'printf' was used to write various values to a port on a regular basis. This is inherently slow and may require the use of a dedicated port on the target. A better alternative is to use 'sprintf' to write to RAM and display that memory using a debugger. While 'sprintf' incurs the same overhead as 'printf' for data conversion, there is no interaction with the outside world required.	<code>update_display</code>
<code>cache_on</code>	0 - (default) The cache is off 1 - The cache is on in <code>proc_spec()</code>	<code>main_cache</code>
<code>curr_loc</code>	Location to write old temp, incremented past end of <code>old_data</code>	<code>old_data</code>

Appendix A: About the MPC860 Demo Board  
**Demo Board Firmware**

<b>Variable</b>	<b>Remarks</b>	<b>See Also</b>
current_temp	Current temperature, used by ECS program.	
func_needed	It is the byte that indicates what the ECS code needs to do to the environment. The following specifies the values it may have: bit                                    bit #1    bit #0 function needed            cool            heat The only valid values for this char are: (hex) (binary) (1)    0001 heat (2)    0010 cool	
hdwr_encode	It is the encoded 16-bit quantity output by the system which is interpreted by external devices. It tells the external devices what to do, for example, turning the air conditioner on (indicated by hdwr_encode = 0010). There are four sets of four bits within the 16 bit quantity hdwr_encode. These sets of bits are encoded as follows: bits 7 - 4                    bits 3 - 0 Air Conditioner            Heater 0 = off                            0 = off 1 = on                              1 = on	
main_cache	0 - (default) The cache is on in proc_spec() (only checked if cache_on = 1) 1 - The cache is on in all modules	cache_on
main_interrupts	0 - (default) No interrupts in main() 1 - Interrupts in main()	
MaKeBaR	MaKeBaR Set to zero to turn off all MAKEBAR's in ECS code, which speeds up the code.	
num_checks	Counts the number of times update_system() is called	
old_data	History of temp data, used by ECS code. Code writes past the end of old_data into target_temp, creating a bug.	curr_loc
outside_temp	Outside temperature, used by ECS code	
target_temp	Target temperature, used by ECS code	
temp_target	Used by ECS code to ramp target_temp; specifies limits	

**Markers, special variables**

The following variables are used to instrument the code. They come in ME\_ and MX\_ pairs. The ME\_ variable is used to mark

when a function is entered by writing a one to it as the first instruction in the function. The end of the function is marked in the same way except a one is written to the MX\_ variable when exiting the function. The analyzer can be set up to trace just the addresses at which a write to the variable pairs occurs, thereby tracing the beginning and end of the functions instrumented. The analyzer can keep track of the time between the writes and there can come up with a trace of how long the software spends in each function over a long period of time.

The ME\_first\_marker and MX\_last\_marker are used to make it easy to set a range across all the code without having to know the exact addresses.

```
ME_first_marker
ME_update_system
MX_update_system
ME_update_display
MX_update_display
ME_clear_hist_buff
MX_clear_hist_buff
ME_proc_specific
MX_proc_specific
ME_do_sort
MX_do_sort
ME_get_targets
MX_get_targets
ME_read_conditions
MX_read_conditions
ME_set_outputs
MX_set_outputs
ME_write_hdwr
MX_write_hdwr
ME_save_points
MX_save_points
ME_add_to_history
MX_add_to_history
MX_last_marker
```

## **Using the PowerPC 860 Emulation Module**

Do not run from reset. Break from reset; then, run. Otherwise, the default DER register setup will not allow setting hardware breakpoints. The copy from the emulation module's CF\_DER register doesn't occur.

## Recommended Demo Configuration

### Why use Recommended Demo Configuration

All of the exercises in this guide, except for the ones in the “Quickly Set Up the Analysis System” chapter on page 13, rely on loading configuration files that were developed on an analysis system with a particular configuration. If the files are loaded into a system that has a different configuration of modules, the exercises may or may not work. This is because different analysis modules have different trace depths and different internal configurations. The analysis system will attempt to load the configuration files in the best way possible, which will allow some of the configuration to work.

### What is the Recommended Demo Configuration

The recommended demo configuration is as follows:

---

<b>Analysis System Slot</b>	<b>Module</b>	<b>Required for:</b>
A	HP 16600A	most exercises
B	HP 16534A	most exercises

---

Most exercises also require an HP 16610A emulation module in slot 1.

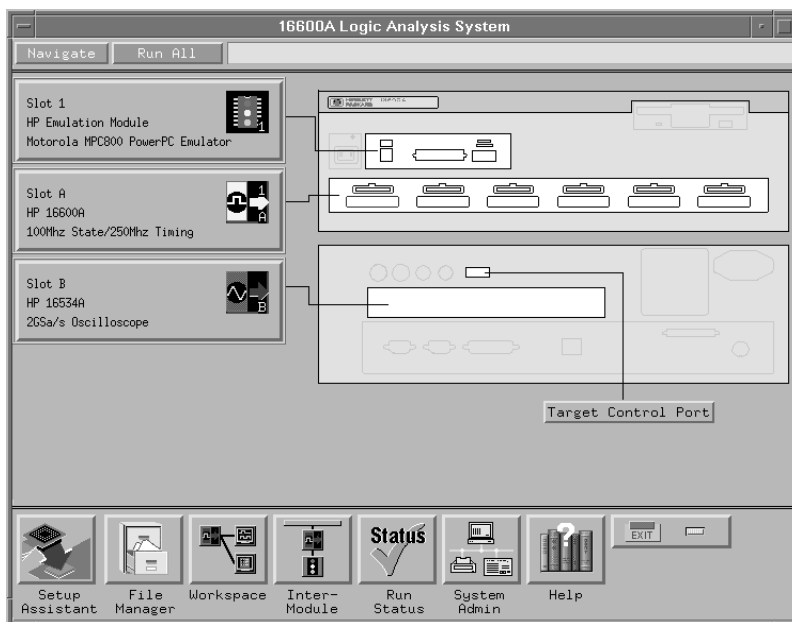
#### Module Descriptions.

HP16600A - 100 MHz State/250 MHz Timing

HP16534A - 2 GSa/s, 2 Ch. Oscilloscope

## Verifying Your Configuration

Start an analysis session and look at the HP 16600 system window. Look at the analysis module buttons down the left side of the window. They should be as follows:





---

**B**

**Concepts**

## Timing Analysis vs. State Analysis in Logic Analyzers

Timing analysis with a logic analyzer is much like tracing a signal with a digital oscilloscope. A scope samples the signal at a rate established by an internal clock and displays it on a screen for viewing. A logic analyzer differs from a digital scope in that one-bit comparators rather than eight-bit are used. This establishes two signal levels (logic 1 or 0) rather than the continuous range of signal levels that you get with a scope. As a result, logic analyzer channels are much cheaper than a scope's, allowing many more to be put in a single instrument.

With a large number of channels, the logic analyzer can show the timing relationship of logic between various signals in a large variety of systems. For example you could look at the timing relationship between the logic levels of the control lines, address lines, and data lines of a processor. However, you may be more interested in a sequence of states on the processor's bus than the timing relationship of the bus elements. This acquisition would be best if each state were acquired only when they were valid. In other words, synchronous with the bus clock.

When you want to capture signals that are synchronous with clocks or other signals in the target system, the logic analyzer can use external clock signals for sampling data. Of course, the clock edge you give to the logic analyzer must sample target system data when it is valid. When an external clock is being used for sampling, the logic analyzer is configured as a *state analyzer*.

---

# Glossary

## A

**analysis probe** A special type of probe for connecting the logic analyzer to microprocessors or standard busses. Analysis probes typically include files for configuring the logic analyzer and labeling channels, as well as inverse assemblers for decoding captured bus cycles into assembly language instructions.

**arm** A condition that enables an instrument and allows it to trigger. You can set up one instrument to arm another using the Intermodule window. In some analyzer instruments, you can set up one analyzer machine to arm the other analyzer machine in the Setup window's Trigger tab.

## C

**channel** A single probe for capturing data on one signal.

**clock inputs (labeled J, K, L, M, etc.)** The clock inputs on the logic analyzer pods are labeled J, K, L, M, etc. for pods 1, 2, 3, 4, etc., respectively. We have preserved this labeling to maintain continuity with previous logic analyzers.

## E

**emulation adapter** Attaches to an emulation module and has a special connector for interfacing with a microprocessor's debug port connector.

**emulation module** A module that can be installed in the HP 16600A-series logic analysis system to provide the ability to run, step, or stop microprocessor execution, set breakpoints, and modify the contents of microprocessor registers and memory locations.

**emulation probe** A stand-alone emulation module and adapter combination that can exist on the Local Area Network (LAN) and be used with 3rd-party debuggers or the HP 16600A-series logic analysis system.

## L

**logic analyzer** An instrument that captures digital signals.

**labels** Names given to logic analyzer channels. Typically, these names correspond to the signals or busses they are probing.

---

# Glossary

## M

**machine** A collection of logic analyzer channels that can be configured independently as a timing analyzer or as a state analyzer.

**module** An instrument that uses a single time base in its operation. Modules can have from one to five cards functioning as a single instrument. When a module has more than one card, they main system window will show the instrument icon in the slot of the master card.

## O

**oscilloscope** An instrument that captures analog signals and the analog parameters of digital signals.

## P

**pod** A group of logic analyzer channels. These are the 2x20 connectors on the end of the cables coming out of the logic analyzer. A pod is assigned to a virtual analyzer by dragging it to the area below the analyzer's name.

**pod pair** All current HP logic analyzer modules group their pods into pod pairs.

**processor solution** A bundled HP product that includes an analysis probe, and emulation module, and the source correlation tool set.

## S

**sequencer** A state machine in the logic analyzer's data recognition and filtering circuitry that is used to identify when data should be captured (in other words, when the logic analyzer should trigger).

**state analysis** When data is sampled at a rate determined by clocks external to the logic analyzer.

## T

**target system** The system under development whose digital signals you are measuring.

**timing analysis** When data is sampled at a rate determined by the logic analyzer's internal clock.

---

# Glossary

**trigger** The reference point in a logic analyzer measurement.

**tool set** An add-on software product for the HP 16600A-series logic analysis system.

## V

**virtual analyzer** Virtual analyzers provide a way of grouping the different signals you collect. For the most part, you won't have to worry about having two virtual analyzers, just group all your active pods under one analyzer and proceed to the format tab of the setup dialog. However, there are times when you must use two analyzers, such as when you want to collect both state and timing traces.



---

# Index

---

## A

analog parameters of signals, 61  
analysis probe, 27, 139  
analysis probes, 22, 43  
arm, 139  
ascii\_old\_data, 131

## B

Background Debug Mode (BDM), 24  
Background Debugger Mode, 118  
BGA footprints, 23  
boot\_q(), 128  
branch trace messages, 99  
branch trace messaging, 102

## C

cache\_on, 131  
cache-on execution trace inverse assembler, 115  
cache-on execution tracker, 99  
CAN controller, 124  
CAN interrupt, 75  
CF\_DER register, 134  
channel, 139  
clock inputs, 139  
clocking, 115  
code in cache, 131  
collaborative debugging, 2  
color, waveform, 19  
compare, 2  
concepts, 137  
configuring the logic analysis system, 114  
connecting the analyzer, 15  
connecting the demo board, 10  
connector mapping, 118  
context store, 2, 91  
Controller Area Network (CAN) Bus, 130  
Controller Area Network (CAN) bus, 43, 124  
curr\_loc, 131

current\_temp, 132

## D

D/A converter, 52, 71, 124, 130  
DATA and STAT labels, 115  
debug port, 71  
deep traces, 106  
deep-memory logic analyzer modules, 107  
deep-memory traces, 2, 106  
default configuration, 16, 27  
demo board connector mapping, 118  
demo board features, 122  
demo board firmware, 127  
demo board hardware, 114  
demo guide, 4  
demo kit, 4  
DER register, 134  
download code, 80  
downloading code into RAM, 81

## E

embedded microprocessor cores, 23  
emulation adapter, 139  
emulation module, 2, 71, 134, 139  
emulation module connector, 118  
emulation module, connecting demo board, 12  
emulation modules, 24  
emulation probe, 139  
emulation probes, 25  
Environmental Control System (ECS), 125, 127  
execution trace inverse assembler, 99, 102, 115  
external clock for sampling, 138

## F

filter, 116  
firmware drivers, 70  
Flash ROM, 80, 123  
flat distortion, 56

---

# Index

---

flying lead set, 15  
format, 115  
func\_needed, 132  
function locations, 130

## G

G1 marker, 20, 50  
G2 marker, 21, 50  
Getting Started, 9  
global markers, 49, 58  
glossary, 139  
ground bounce, 68  
ground bounce register, 130

## H

hard disk, 36  
hardware breakpoints, 134  
hdwr\_encode, 132  
high-density Mictor38 connector, 23  
high-density termination adapters,  
23  
high-speed timing connector, 121  
HP 16555D logic analysis modules, 16  
HP E5346A high-density termination  
adapters, 23

## I

init\_system(), 128  
insight, 2, 42  
inverse assemble, 22  
inverse assembler, 23  
inverse assembler filter, 31, 116  
inverse assembler preferences, 31,  
115  
inverse assembly, 115  
ISA busses, 43

## J

JTAG connection, 118  
JTAG port, 24

## L

labels, 139  
labels and format, 115  
latch with a ground bounce problem,  
124  
LCD, 123  
LCD display with pattern generator  
connections, 124  
logic analyzer, 139  
logic analyzer connectors, 119  
logical (digital) behavior, 61  
low-density pod connections, 119

## M

machine, 140  
main(), 127  
main\_cache, 132  
main\_interrupts, 132  
MaKeBaR, 132  
markers, 20  
markers, special variables, 132  
memory map, 122  
microprocessor cores, embedded, 23  
microprocessor support, 28  
Mictor38 connector, 23  
Mictor38 connectors, 119  
Mictor38 probing technology, 125  
module, 140  
Motorola S-record, 81  
MPC860 demo board, 113

## N

networked logic analysis system, 2  
num\_checks, 132

## O

object file, 26  
old\_data, 132  
oscilloscope, 140  
oscilloscope connections, 121  
oscilloscope module, 52



---

# Index

---

outside\_temp, 132  
overlay, 18

## P

pattern generator connectors, 121  
pattern generator module, 2  
PCI busses, 43  
PCMCIA busses, 43  
PGA sockets, 23  
physical behavior, 61  
PLD, 123  
pod, 140  
pod pair, 140  
post-processing filtering tools, 2  
preferences, 115  
proc\_spec\_init(), 128  
proc\_specific(), 129, 130  
processor solution, 140  
processor solution information, 26  
processor solution packages, 26  
processor solutions, 22  
pushbutton, 125

## R

recommended configuration, 10  
recommended demo configurations,  
135  
root cause, 3

## S

scope probe ground clips, 122  
SCSI 1, 2, and 3 busses, 43  
search capability, 109  
sequencer, 140  
Setup Assistant, 27  
SIMASK register, 76  
SIU group, 76  
software issues, 83  
source code location, 36  
source correlation tool set, 26, 35  
source line, trigger on, 46  
Source Viewer, 35

SPA tool set, 84  
special variables, 132  
SRAM, 123  
stair step distortion, 56  
standard bus, 22  
standard busses, 43  
standard inverse assembler, 115  
STAT label, 115  
state analysis, 138, 140  
state analyzer, 43  
stimulus, 2  
storage qualifiers, 91  
stuffed bits, 51  
surface mount package pins, 22  
symbol file, 37  
symbol information, 26  
synchronous with the bus clock, 138  
system performance analysis tool set,  
84

## T

Target Interface Module (TIM), 24,  
71, 118  
target system, 140  
target\_temp, 132  
temp\_target, 132  
testing hypotheses, 2  
time duration, 87  
time to insight, 2  
time-correlate, 2  
time-correlated measurement, 54  
timing analysis, 138, 140  
timing analyzer, 43  
timing markers, 20  
timing traces, 120  
tool set, 141  
TQFP packages, 23  
triangular waveform, 52, 71  
trigger, 141

## U

update\_display(), 129

---

# Index

---

update\_system(), 129  
USB busses, 43

## **V**

valid data, 138  
variable frequency interrupt  
  subsystem, 123  
variables, 131  
verifying your configuration, 136  
virtual analyzer, 85, 141  
virtual logic analyzers, 43  
VME busses, 43

## **W**

world-wide web, 26, 43

## **Z**

zoom, waveform display, 19

© Copyright Hewlett-Packard  
Company 1998  
All Rights Reserved.

Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

### Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in subparagraph (C) (1) (ii) of the Rights in Technical Data and Computer Software Clause in DFARS 252.227-7013. Hewlett-Packard Company, 3000 Hanover Street, Palo Alto, CA 94304 U.S.A. Rights for non-DOD U.S. Government Departments and Agencies are set forth in FAR 52.227-19 (c) (1,2).

### Document Warranty

The information contained in this document is subject to change without notice.

### Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose.

Hewlett-Packard shall not be liable for errors contained herein or for damages in connection with the furnishing, performance, or use of this material.

### Safety

This apparatus has been designed and tested in accordance with IEC Publication 1010, Safety Requirements for Measuring Apparatus, and has been supplied in a safe condition. This is a Safety Class I instrument (provided with terminal for protective earthing). Before applying power, verify that the correct safety precautions are taken (see the following warnings). In addition, note the external markings on the instrument that are described under "Safety Symbols."

### Warning

- Before turning on the instrument, you must connect the protective earth terminal of the instrument to the protective conductor of the (mains) power cord. The mains plug shall only be inserted in a socket outlet provided with a protective earth contact. You must not negate the protective action by using an extension cord (power cable) without a protective conductor (grounding). Grounding one conductor of a two-conductor outlet is not sufficient protection.
- Only fuses with the required rated current, voltage, and specified type (normal blow, time delay, etc.) should be used. Do not use repaired fuses or short-circuited fuseholders. To do so could cause a shock or fire hazard.
- Service instructions are for trained service personnel. To avoid dangerous electric shock, do not perform any service unless qualified to do so. Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.
- If you energize this instrument by an auto transformer (for voltage reduction), make sure the common terminal is connected to the earth terminal of the power source.
- Whenever it is likely that the ground protection is impaired, you must make the instrument inoperative and secure it against any unintended operation.
- Do not operate the instrument in the presence of flammable gasses or fumes. Operation of any electrical instrument in such an environment constitutes a definite safety hazard.
- Do not install substitute parts or perform any unauthorized modification to the instrument.
- Capacitors inside the instrument may retain a charge even if the instrument is disconnected from its source of supply.

### Safety Symbols



Instruction manual symbol: the product is marked with this symbol when it is necessary for you to refer to the instruction manual in order to protect against damage to the product.



Hazardous voltage symbol.



Earth terminal symbol: Used to indicate a circuit common connected to grounded chassis.

### WARNING

The Warning sign denotes a hazard. It calls attention to a procedure, practice, or the like, which, if not correctly performed or adhered to, could result in personal injury. Do not proceed beyond a Warning sign until the indicated conditions are fully understood and met.

### CAUTION

The Caution sign denotes a hazard. It calls attention to an operating procedure, practice, or the like, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product. Do not proceed beyond a Caution symbol until the indicated conditions are fully understood or met.

Hewlett-Packard  
P.O. Box 2197  
1900 Garden of the Gods Road  
Colorado Springs, CO 80901-2197, U.S.A.

---

## Product Warranty

This Hewlett-Packard product has a warranty against defects in material and workmanship for a period of one year from date of shipment. During the warranty period, Hewlett-Packard Company will, at its option, either repair or replace products that prove to be defective.

For warranty service or repair, this product must be returned to a service facility designated by Hewlett-Packard.

For products returned to Hewlett-Packard for warranty service, the Buyer shall prepay shipping charges to Hewlett-Packard and Hewlett-Packard shall pay shipping charges to return the product to the Buyer. However, the Buyer shall pay all shipping charges, duties, and taxes for products returned to Hewlett-Packard from another country.

Hewlett-Packard warrants that its software and firmware designated by Hewlett-Packard for use with an instrument will execute its programming instructions when properly installed on that instrument. Hewlett-Packard does not warrant that the operation of the instrument software, or firmware will be uninterrupted or error free.

## Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by the Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environmental specifications for the product, or improper site preparation or maintenance.

**No other warranty is expressed or implied. Hewlett-Packard specifically disclaims the implied warranties of merchantability or fitness for a particular purpose.**

## Exclusive Remedies

The remedies provided herein are the buyer's sole and exclusive remedies. Hewlett-Packard shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

## Assistance

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products. For any assistance, contact your nearest Hewlett-Packard Sales Office.

## Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Institute of Standards and Technology, to the extent allowed by the Institute's calibration facility, and to the calibration facilities of other International Standards Organization members.

## About this edition

This is the *HP 16600-Series Logic Analysis System Demo Guide*.

Publication number  
16600-97003, March 1998  
Printed in USA.

Print history is as follows:  
First edition, March 1998

New editions are complete revisions of the manual. Many product updates do not require manual changes, and manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual updates.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.