



Evaluation Guide (PC)

SuperTAP™
System Development Tool
for the Motorola PowerPC® MPC8XX™
Family of Microcontrollers

P/N 924-00100-00



Applied Microsystems Corporation



Applied
Microsystems
Corporation

Evaluation Guide (PC)

SuperTAP™
System Development Tool
for the Motorola PowerPC® MPC8XX™
Family of Microcontrollers

P/N 924-00100-00

March 1997

Copyright © 1997 Applied Microsystems Corporation.
All rights reserved.

Information in this document is subject to change without notice. Applied Microsystems Corporation reserves the right to make changes to improve the performance and usability of the products described herein.

Applied Microsystems Corporation's CodeTAP and SuperTAP products are protected under U.S. Patents 5,228,039 and 5,581,695. Additional patents pending.

Trademarks

CodeTAP is a registered trademark of Applied Microsystems Corporation.

SuperTAP, VSP-TAP, CodeICE, CodeCONNECT, CodeTEST, RTOS-Link, CPU Browser, NSE, Transparent Breakpoints and NetROM are trademarks of Applied Microsystems Corporation.

Other product names, trademarks, or brand names mentioned in this document belong to their respective companies.

Contents

Chapter 1 Getting Started

Applied Microsystems Corporation

Why choose Applied Microsystems for my Motorola project?	1
---	---

Customer support

Committed to your success	2
Contacting Customer Support	2

Demonstration overview

Running the examples	3
How long does it take?	3
Running in other targets	3
Evaluation Guide conventions	4

Radio interference warning

FCC Rules	5
EMC Directive	5

Operating requirements

Standard electrostatic precautions	7
--	---

Product overview

Emulator, BDM, and logic analyzer in one tool	10
Uses no target resources	10
Non-stop emulation (NSE)	10

Debug and track execution of cache-based code	10
Isolate complex software and hardware bugs	10
Configure complex integrated peripherals	11
Accelerate your firmware development	11
Troubleshoot prototype hardware	12
Monitor target conditions	12
Real-time visibility into your RTOS-based system	12
Compatibility with your tools and hosts	13
Non-intrusive performance analysis	13
Innovative target connection support	14
Electrical characteristics mimic the CPU	14

MWX-ICE Debugger for SuperTAP

Debugger features	15
Easy to use interface	15
Shortens the learning curve	16
Powerful Help system	16
Handles symbols in a variety of ways	16
Data and Inspector windows	16
Powerful macro support	17
Evaluates C, C++, assembler expressions	17

Using SuperTAP with MWX-ICE

Configuring the initregs	18
Starting the MWX-ICE debugger	19
Getting debugger help	20
Stopping emulation (leaving run mode)	21
Important information about leaving run mode	21
Loading the demonstration code	21

“Include” debugger command files	21
Viewing source-level and assembly code simultaneously	23
SuperTAP configuration information	24

Chapter 2

Examples

Running the examples	27
How long does it take?	27
Run the examples that most interest you	27
Complete all steps within an example	27

Accelerate the debug of prototype hardware

Crash proof debugging and testing	28
Built-in scope loops and memory diagnostics	28
Monitor target conditions	28
Example 1 – Crash-proof tool including trace history	29
Example 2 – Scope loops and peek/poke trace	31
Example 3 – Benefits of event system qualified trace	32
Example 4 – Isolate complex bugs with the event system	35
Monitor complex sets of conditions	35
Do more than just break	35

Accelerate firmware development using overlay memory

Description	39
Real-time code execution vehicle	39
Copy memory contents between the target and overlay	39
Example 5 – Using overlay memory	40

Controlling code execution

Do more than just break	41
-------------------------------	----

Description	41
Example 6 – Intuitive breakpoints on code and data.....	43
Example 7 – Hardware execution and access breakpoints.....	47

Capturing and viewing trace history

Description	49
Capturing “qualified” trace versus filtering continuous trace ...	49
Execute and trace target power-up or reset sequences	50
Dynamic trace display	50
Timestamp	50
Saving trace history in a file	50
Example 8 – Intelligent trace disassembly.....	51
Example 9 – Capturing and viewing trace while running.....	53

Examples of other time-saving features

Description	56
Example 10 – Configuring/debugging peripheral registers.....	57
Example 11 – Displaying high-level data structures.....	58
Example 12 – Modifying variables dynamically.....	60
Example 13 – Displaying and modifying memory.....	64

Conclusion

Appendix A

Cdemon Demonstration Program

initial()	68
step()	68
data()	69
init()	70
sort()	70

shuffle()	70
deal()	70
hit()	70
house()	70
run()	71
outled()	71
wait()	71

Chapter 1

Getting Started

This document provides a demonstration of the Applied Microsystems SuperTAP system integration tool with MWX-ICE debugger. It assumes the SuperTAP and debugger are installed and the debugger is connected to the emulator, described in the *Emulator Installation Guide* and Chapter 2 of the *MWX-ICE User's Manual*.

Applied Microsystems Corporation

Founded in 1979, Applied Microsystems is a leading ISO 9002-certified provider of integrated development systems for embedded microprocessor design. The company helps engineers develop and test quality products faster, more reliably, and at a lower cost through a growing family of compatible design tools.

Why choose Applied Microsystems for my Motorola project?

Applied Microsystems is an experienced and innovative leader in the field of microprocessor development tools. Motorola realized this when they made Applied Microsystems a preferred Motorola "Platinum" development tools partner.

Applied Microsystems has a long and successful record supporting Motorola microprocessors; from the 6809 to PowerPC. Over 17 years experience with Motorola microprocessors lets us provide tools with the critical features necessary to debug today's complex software applications in high performance embedded systems.

First and fastest, Applied gave the industry its first 33 Mhz 68020/030 emulator, its first 40 MHz 68040 emulator, and its first 66 MHz 68060 emulator.

Customer support

Committed to your success



Good support is as critical a part of your development system as the emulator or the debugger. Applied Microsystems Customer Support department employs a large staff of engineers and offers a full range of timely and effective support services.

Contacting Customer Support

PHONE: If you encounter trouble while evaluating this product, call Customer Support at 1-800-ASK-4AMC (1-800-275-4262).

FAX: To FAX Applied Microsystems, dial 1-206-883-3049.

EMAIL: If you have access to the internet, you can contact Applied Microsystems by sending email to support@amc.com.

WWW: Applied Microsystems maintains a World Wide Web site located at URL <http://www.amc.com>.

Demonstration overview

Chapter 1 of this guide introduces you to the SuperTAP and MWX-ICE debugger. After a product description, you start the debugger and become familiar with using its windowing system, load the demonstration program, and display SuperTAP configuration information. You must complete all of Chapter 1 before doing any of the examples in Chapter 2.

Running the examples

Chapter 2 contains examples demonstrating basic debugger operation and key debugging features of Applied Microsystems SuperTAP. The examples are organized into functional groups, with each group prefaced by a brief discussion of the function.

How long does it take?

It takes approximately two hours to complete all the examples. However, the examples are not interdependent; you can choose to run, in any sequence, the examples that most interest you.

You *must* complete each step within an example. The final steps of an example return settings to their original value, so you can do the examples in any order.

Running in other targets

The Cdemo demonstration code used in this guide was written for the Applied Microsystems demonstration board. Cdemo is loaded into emulator memory which has been configured (mapped) to replace (overlay) the desired address spaces of target memory. Because the program runs in overlay memory, the demonstration can be run without a target, with the emulator in isolation mode.

Although the program runs in overlay memory, all write cycles still go out to the target. The demonstration can be run in other embedded targets if the following memory considerations pose no problem:

<u>address range</u>	<u>type of memory</u>
0x000000 - 0x01ffff	read/write
0x100000 - 0x17ffff	read/write
0x1e0000 - 0x1fffff	read/write

Evaluation Guide conventions

Throughout the guide, an arrow in the margin indicates the beginning of procedures that you should follow. Below is an example of this type of procedure. (Do not follow the procedure at this time).

- **To restart the Cdaemon program:**
 - In the Command window Enter Command box, enter:
`restart`



The frog in the margin indicates that a feature is unique to Applied Microsystems products, and provides considerable benefit to you in debugging.

Warning



Warning messages appear before procedures and alert you to the danger of personal injury which may result unless certain precautions are observed.

Caution



Caution messages appear before procedures and indicate that damage may be done to the emulator or to your target system unless certain steps are observed.

Note



Notes indicate important information for the proper operation and installation of your emulator.

Radio interference warning

This equipment generates, uses, and can radiate radio frequency energy.

Caution



This instrument is intended for use in the development of microprocessor-based systems. At this stage of development, these target devices typically include no inherent design to limit the emissions of electromagnetic energy.

Precautions should be taken to prevent harmful radiation to radio communications and other nearby sensitive electronic systems by means of isolation, separation, or shielding, where necessary.

Use of this instrument in a residential area is likely to cause harmful interference, in which case, the user will be required to correct the interference at his own expense.

FCC Rules

It is temporarily permitted by regulation and has not been tested for compliance with the limits of Class A computing devices pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference.

EMC Directive

For compliance to the essential requirements of the EMC Directive 89/336/EEC, if a ground post is provided on the back of the chassis or on the external power supply, a properly bonded ground strap must be connected to it.

Operating requirements

Before setting up the SuperTAP, you should make sure that the operating environment is prepared.

Caution



CE-compliant systems: The AC third wire ground (safety ground) is not connected to the emulator's digital ground. For proper operation, your target's safety ground and digital ground should not be connected. If they are connected, operation will result in the emulator's DC voltage fuses (if any) blowing. If the target power supply's digital ground cannot be isolated from the safety ground, then temporarily connect a wire between the ground stud on the emulator external power supply and target digital ground. This ground wire is in addition to the ground wire between the power supply ground stud and AC ground that is required for CE compliance and is described later in this manual.

Under no circumstances should the third wire prong on the emulator power cord be removed or disconnected.

Caution



Non-CE-compliant systems: The AC third wire ground (safety ground) is connected to the emulator's digital ground. For proper operation, your target's safety ground and digital ground should also be connected. If they are not connected, the DC voltage fuses (if any) in the target or in the target power supply may blow. For correct operation, connect your target's safety ground and digital ground.

Under no circumstances should the third wire prong on the emulator power cord be removed or disconnected.

Caution



Be sure the emulator and target are plugged into outlets that have good electrical continuity on the third wire safety ground. If there is high electrical resistance on the safety grounds of either or both the target power supply and emulator power supply, unwanted ground current may be drawn through the emulator probe, damaging the emulator or target system. Good electrical continuity is also required to maintain the EN55022 Class B Conducted Radiation Specification of the emulator power supply.

Standard electrostatic precautions

This instrument contains static-sensitive components that are subject to damage from electrostatic discharge. Use standard ESD precautions when transporting, handling, or using the instrument and the target, when connecting/disconnecting the instrument and the target, and when removing the cover of the instrument.

Applied Microsystems recommends the use of the following precautions:

- Use wrist straps or heel bands with a 1 Megohm resistor connected to ground.
- On the work surface and floor, use static conductive mats with a 1 Megohm resistor connected to ground.
- Keep high static-producing items, such as non-ESD-approved plastics, tape and packaging foam, away from the instrument and the target.

The above precautions should be considered as minimum requirements for a static-controlled environment.

Caution



The emulator contains components that are subject to damage from electrostatic discharge. Whenever you are using, handling, or transporting the emulator, or connecting to or disconnecting from a target system, always use proper anti-static protection measures, including static-free bench pads and grounded wrist straps.

Product overview

The SuperTAP is a complete tool for system debugging and integration. You can use SuperTAP and MWX-ICE in full in-circuit emulation mode to bring up your target hardware, program flash memory, or execute code before target hardware is available. The SuperTAP supports external bus masters in multi-processor target systems. You can also use the SuperTAP and MWX-ICE in DPI-only mode to debug rack-mounted hardware or for field troubleshooting.

The SuperTAP system integration tool is one member of Applied's Design, Test, and Debug tool-set for MPC860 developers. Other members of this family include:

- CodeTEST™ software testing and verification tools let you optimize your software's performance and ensure quality and reliability in your finished product.
- NetROM™ universal target communications tool transforms your target into an easily accessible network node and provides a high-speed Ethernet connection between target and host.
- VSP-TAP™ lets you execute real product software against a model of your ASIC and to stimulate the ASIC design with real-world data. VSP-TAP works in concert with Viewlogic's modeling and simulation tools.
- CodeTAP® application development and debug tool combines full processor and target visibility with integrated support for for multiple debuggers including MWX-ICE and Wind River's Cross-wind.

Contact Applied Microsystems for detailed information about these tools.

Emulator, BDM, and logic analyzer in one tool

SuperTAP provides a high-quality yet cost-effective approach to developing and debugging Motorola MPC860 family embedded processor systems. Highly versatile, SuperTAP is an emulator, BDM, and logic analyzer in one tool.

SuperTAP physically replaces the target system's embedded processor, then communicates through an Ethernet or high-speed serial link to MWX-ICE C/C++ source-level debugger.

Uses no target resources

Unlike ROM monitors, SuperTAP uses no target resources like I/O, memory, or serial ports. Unlike other emulators, SuperTAP does not install a monitor in target memory or make use of any target resources. SuperTAP does not even require a fully functional target, as is common with prototype hardware.

Non-stop emulation (NSE)

NSE™ lets the target system continue to run even when you are defining events or uploading trace. This is made possible by the SuperTAP's dual-processor architecture. NSE can be particularly useful for targets that need to service interrupts on demand.

Debug and track execution of cache-based code

A large part of the impressive performance of the MPC860 comes from its two large code and data caches. If you turn them off to debug the software your product may no longer function. You need to debug in an environment that closely matches the way the real product will operate.

SuperTAP solves the problem with its technology for tracking code executed in cache. Now you can debug with the caches enabled to enjoy a more realistic target debug environment.

Isolate complex software and hardware bugs

Most software bugs are caused by data corruption (stack and data pointers becoming corrupt) and program misdirection (jumping to the wrong address). Or, the software may run

correctly but the target hardware does not respond as it should. SuperTAP addresses these problems quickly and efficiently, providing:

- Real-time emulation with an easy to use multi-windowed C/C++ source-level debugger supporting the popular PowerPC compilers
- A multi-thread event system which allows tracking and detecting the state of the program as well as the processor
- 64K by 128 bits of bus cycle trace history monitors 160 signals and provides raw bus cycle, assembly, and C/C++ source-level display modes
- 1, 4, or 8MB of overlay memory to hard code, replace, or extend target memory
- Data access breakpoints in RAM or ROM
- Execution breakpoints in RAM or ROM
- Trigger input/output capability for linking together multiple instruments

Configure complex integrated peripherals

Configuring the MPC860 peripherals can be a real challenge. The processor data book holds 23 chapters and over 1300 pages. SuperTAP's MWX-ICE CPU Browser simplifies developing and debugging device drivers by providing:

- A description of the function of each bit of each register
- Pulldown menus with the possible status for each bit
- The resulting hexadecimal register value for a particular configuration

Accelerate your firmware development

In isolation mode, SuperTAP can be operated stand-alone without a being plugged into a target. With your code loaded into overlay memory, isolation mode lets you begin debugging your program before target hardware is available.

Troubleshoot prototype hardware

You can verify the design of your hardware and quickly isolate manufacturing defects using built-in diagnostic test routines and scope loops provided by SuperTAP. Combined with an oscilloscope, SuperTAP can quickly isolate manufacturing errors in target hardware.

Monitor target conditions

SuperTAP monitors and reports target conditions such as:

- Processor clock frequency
- Target Vcc
- Bus error
- Reset asserted

Real-time visibility into your RTOS-based system

SuperTAP helps you debug both before and after your kernel is working on your target. Applied's RTOS-Link option provides several broad categories of support, including:

- Real-time insight to help develop your Board Support Package (BSP)
- A real-time tool that let's you debug your kernel initialization code
- Real-time trace of RTOS activity
- Display of individual task context and other system structures
- Task-qualified breakpoints and stack overflow detection
- Task profiling support
- System-level OS support to debug your Interrupt Service Routines

The benefit of integrated RTOS support is a substantial reduction in the time required to debug and optimize your application as it runs in your target.

The event system is specifically designed to help isolate problems in multi-threaded software systems found in kernel applications. The system is organized in a four-state-by-four-



group structure. Each group can be applied to a software thread and the four states can be used to isolate deeply nested bugs.

Stop-in-target mode permits a single target execution thread to be halted while maintaining additional program threads in target. You can determine how frequently the emulator interrupts the target operation and the maximum time it can spend out of target before restoring the un-halted threads. The emulator can interrupt the current operations, service its interrupts, restore the original bus state, and resume the original operations.

Compatibility with your tools and hosts

Host support includes:

- Sun 4 (Sun OS 4.1 and above)
- PC (Windows 95 or WinNT)
- HP9000 (HPUX 9.0)

Compiler support for all C/C++ compilers generating ELF/DWARF version 1.03 or ELF/stabs including:

- GNU (C/C++)
- DIAB data (C/C++)
- Greenhills (C)
- MRI (C)
- Metaware (C/C++)

SuperTAP used with Applied's RTOS-Link provides support for WindRiver and ISI real-time kernels.

Non-intrusive performance analysis

SuperTAP's timestamp capability greatly aids in viewing system activity and finding code bottlenecks.



For the most powerful software measurements, team SuperTAP with Applied's CodeTEST software verification and test tools. CodeTEST provides detailed Performance Analysis, Code Coverage Analysis, Memory Allocation Analysis, and

extended Source Trace Analysis. For details, request a CodeTEST specification sheet or view the information on Applied's web site (<http://www.amc.com>).

Innovative target connection support

The MPC860 adapter and connector support includes:

- BGA to PGA connector
- Logic Analyzer adapter
- Mirror adapter (leaves the processor on card)
- Rotation adapters (to facilitate target connection)
- Motorola ADS 8XX header adapters (leaves ZIF socket on card)

Electrical characteristics mimic the CPU

With its fully isolated probe-tip, the SuperTAP behaves electrically as close to the emulated microprocessor as is possible with today's technology. Zero-delay buffering ensures the most accurate emulation.

MWX-ICE Debugger for SuperTAP

MWX-ICE is a function rich, Sun 4, HP 9000/700, or PC-hosted, C/C++ source and assembly-level debugger, developed solely for debugging embedded applications. MWX-ICE's multiple windows, pull-down and pop-up menus, and point-and-click ease of use provide a fast, interactive debugging environment.

MWX-ICE gives you complete control over SuperTAP functions: you can selectively start and stop execution, view program execution trace with source disassembly, and display and modify CPU registers and memory.

Debugger features

MWX-ICE features include:

- Point-and-click encapsulation of the debugger language via command buttons and “notebooks”
- Diab data, GNU, Greenhills, and Metaware compiler support
- Full support of SuperTAP hardware including Trace, Event, and Overlay
- Powerful breakpoint conditions/actions
- C, C++, assembler expression evaluation
- Versatile and complete macro language
- Accepts batch mode command files

Easy to use interface

The MWX-ICE debugger is a “windowed” user interface. Commands are usually executed either of two ways: by typing the command in a Command window, or by clicking a button that activates the command. Also available for command execution and data entry are editable debugger Windows and Notebooks accessed through pull-down menus.

Shortens the learning curve

Users may find the more intuitive buttons easier to use while learning the debugger. If a button has a command line associated with it, that line will be printed in the Command window when the button is clicked. This shortens the learning curve for users who may ultimately prefer the speed of the command line by providing learning reinforcement with each button click.

Powerful Help system

MWX-ICE includes an extensive but easy-to-use hypertext Help system that saves you time otherwise spent searching through manuals.

Handles symbols in a variety of ways

Symbols include arrays, structures, static variables, register-based, and stack-based variables. Symbols can be displayed or changed by name, as declared in your program. You can display the type and scope of each symbol, and its value in binary, hex, ASCII, or decimal format. You can also display memory contents with absolute references or register-relative references using the Memory window.

Data and Inspector windows

MWX-ICE Data and Inspector windows provide access to high-level data structures and dynamic variables, and includes C++ object support. You can reference structure members, dereference pointers, and apply type overrides.

The Data window lets you easily monitor and examine data (and code) using C and C++ source constructs for a dynamic, real-time view of the data structure.

The Inspector window lets you view complex high-level data structures and quickly traverse linked lists.

Powerful macro support

Easily set up using an MWX-ICE “notebook”, macros allow full access to debugger features, source-code, and program variables. By using the “button editor” you can attach macros to the MWX-ICE user interface.

Macros can be tied to breakpoints, providing an excellent method to patch source-code on the fly, and to create program “stubs” for code not yet developed.

Evaluates C, C++, assembler expressions

An expression is a combination of operators and operands. MWX-ICE supports C, C++, and Assembler expressions.

MWX-ICE supports the complete C expression syntax. You can call your C functions, with arguments, from a C expression; exactly as you do in your code. This can be useful for testing the behavior of any newly created functions.

Using SuperTAP with MWX-ICE

Configuring the initregs

The SuperTAP maintains a copy of the values stored in some of the processor's chip-select and pin assignment registers. This copy is called the set of *initialization registers*, or **initregs**, because it is used whenever you resume operation after a target-generated or MWX-ICE command-driven processor reset. The **initregs** must match the values that the processor chip-select and pin-assignment registers have after you run your initialization code. When you save or restore the **initregs** without specifying a path and filename, MWX-ICE uses the file named **iregsxxx.dat** (where the *xxx* is replaced by your processor type: 860, 821, and so forth).

MWX-ICE is shipped with four pre-defined versions of the **iregsxxx.dat** file located in your *install_directory*\amc\ST8XX:

- **iregsxxx.dat.def** - (default) used with isolation mode or as a template to be customized for a specific target
- **iregs860.dat.amc** - used with the Applied Microsystems demonstration board
- **iregs860.dat.ads** - used with the Motorola ADS board
- **iregs860.dat.all** - useful for capturing the values of all the configuration registers

Note



The system default **iregsxxx.dat.def** file is located in *install_directory*\amc\ST8XX. The original **.def** file should never be modified.

- **To use the iregsxxx.dat.def file for isomode (no target):**
 - Copy the file:
install_directory\amc\ST8XX\iregsxxx.dat.def to your working directory as iregsxxx.dat.
- **To use the iregs860.dat.amc file for Applied's demonstration board:**
 - Copy the file:
install_directory\amc\ST8XX\iregs860.dat.amc to your working directory as iregs860.dat.

Starting the MWX-ICE debugger

The *Evaluation Guide* assumes the SuperTAP and debugger are installed and the debugger has been connected to the emulator, described in Chapter 2 of the *MWX-ICE User's Manual*.

- **To start MWX-ICE:**
 - From Program Manager, choose the **MWX-ICE 860** icon. MWX-ICE debugger starts with the Code and Command windows open, shown in Figure 1.

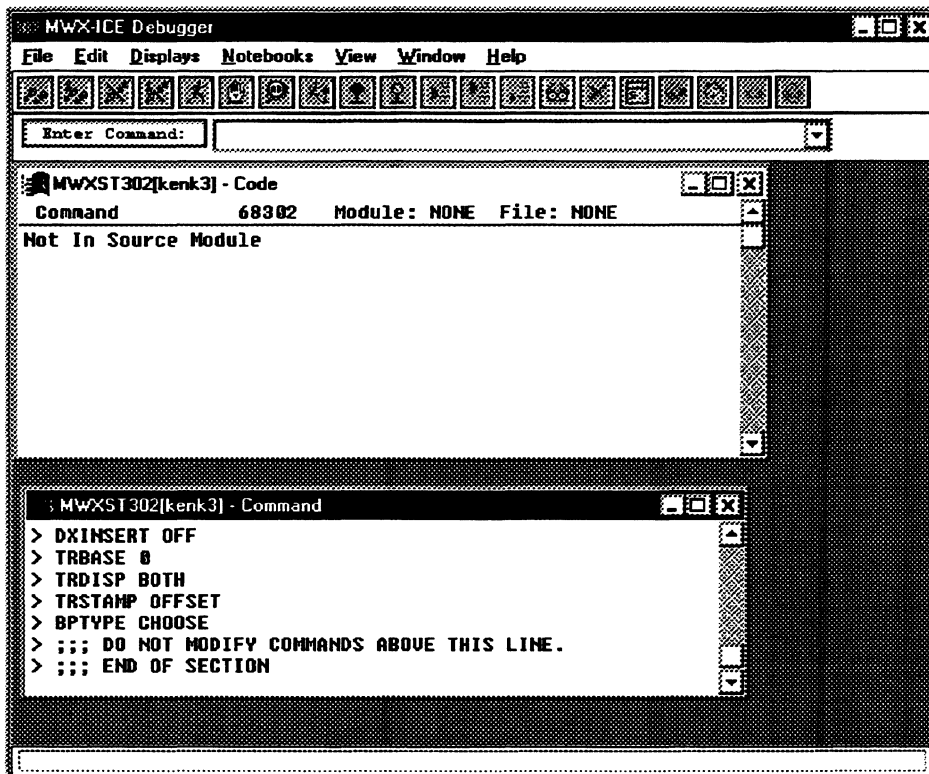


Figure 1 MWX-ICE successfully started

Getting debugger help



Context-sensitive Help is available for MWX-ICE. For information about Applied Microsystems and Applied Microsystems products, choose About MWX-ICE from the Help menu, or click the About Applied Microsystems button on a notebook page.

Stopping emulation (leaving run mode)



Important information about leaving run mode

The SuperTAP emulator operates in one of two modes: *run mode*, when the emulator is executing target code, and *pause mode*, when the emulator is not executing target code. In run mode the cursor turns into a “running man.” To leave run mode *at any time* and return to pause mode, click the Stop button in the tool bar.

Loading the demonstration code

Cdemon is the Applied Microsystems standard C/C++ demonstration program, providing examples of many code and data constructions used by programmers. The demonstration program is designed to be used with the SuperTAP and an Applied Microsystems demonstrator board.

Cdemon is composed of four major functions running in `main()`: `initial()`, `step()`, `data()`, and `run()`. The LEDs on the demonstrator board or a Data window monitoring `led_port` can be used to see the output of some of the functions. For a detailed explanation of the Cdemon demonstration program, see Appendix A of this guide.

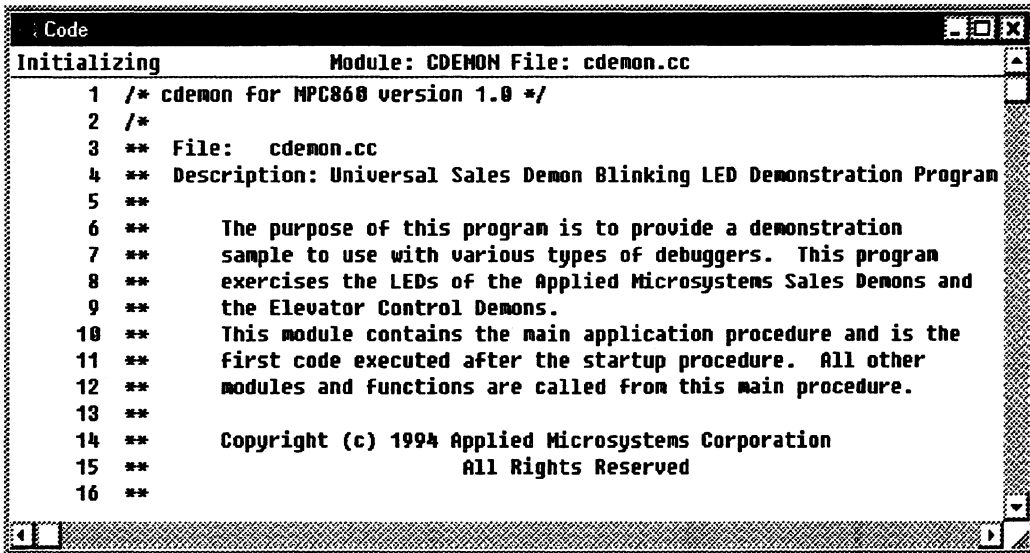
“Include” debugger command files

An include file is simply a file containing debugger commands that will be executed when the file is loaded by the debugger. The supplied include file, `cdemon.inc`, maps overlay then loads the Cdemon absolute file from the demo subdirectory.

► **To load the demonstration file:**

1. From the Displays menu, choose Code.
2. From the File menu, choose Include Commands.
3. Using the file browser, navigate to the *install_dir\amc\ST8XX\demo* directory.
4. Select *cdemon.inc*, then choose OK.

Observe the Command window for SuperTAP status and executed commands. After downloading the file, the Code window displays the source module, shown in Figure 2.



```
Code
Initializing          Module: CDEMON File: cdemo.cc
1 /* cdemo for MPC860 version 1.0 */
2 /*
3 ** File:  cdemo.cc
4 ** Description: Universal Sales Demon Blinking LED Demonstration Program
5 **
6 **     The purpose of this program is to provide a demonstration
7 **     sample to use with various types of debuggers. This program
8 **     exercises the LEDs of the Applied Microsystems Sales Demons and
9 **     the Elevator Control Demons.
10 **    This module contains the main application procedure and is the
11 **    first code executed after the startup procedure. All other
12 **    modules and functions are called from this main procedure.
13 **
14 **    Copyright (c) 1994 Applied Microsystems Corporation
15 **                All Rights Reserved
16 **
```

Figure 2 Cdemo demonstration file successfully loaded

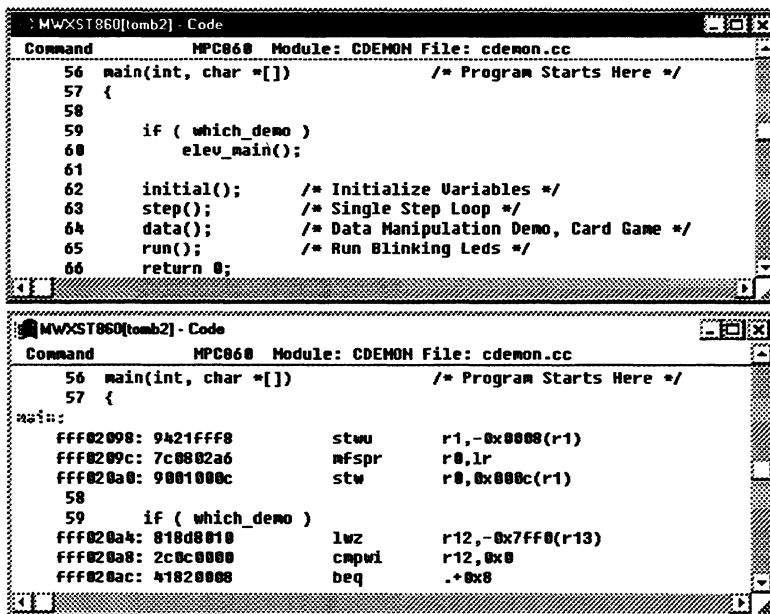
Viewing source-level and assembly code simultaneously

The Windows menu “Copy This Window” button lets you open multiple instances of a window. Below, you open two Code windows; one window displaying source-level and the other displaying interleaved source-level and assembly. The two windows stay synchronized during all emulator operations such as single-stepping, running to breakpoints, and restarting the code.

To open another Code window and display assembly:

1. Make the Code window active.
2. From Displays menu, choose Copy This Window.
3. In Mode button for the new Code window, choose Assembly.

You can arrange the two Code windows for the best viewing, shown stacked in Figure 3.



The figure shows two overlapping windows from the MWXST860 emulator. The top window displays source-level code for the file 'cdemon.cc'. The bottom window displays the same source-level code with assembly instructions interleaved. The assembly instructions are aligned with the source code lines they correspond to.

```
Command      MPC860  Module: CDEMON File: cdemon.cc
56 main(int, char *[])      /* Program Starts Here */
57 {
58
59     if ( which_demo )
60         elev_main();
61
62     initial();      /* Initialize Variables */
63     step();         /* Single Step Loop */
64     data();         /* Data Manipulation Demo, Card Game */
65     run();          /* Run Blinking Leds */
66     return 0;

```

```
Command      MPC860  Module: CDEMON File: cdemon.cc
56 main(int, char *[])      /* Program Starts Here */
57 {
*:*:*
fff02098: 9421fff8      stwu    r1,-0x0008(r1)
fff0209c: 7c0802a6      mfspr  r0,r1
fff020a0: 9061000c      stw    r0,0x000c(r1)
58
59     if ( which_demo )
fff020a4: 818d8010      lwz    r12,-0x7ff0(r13)
fff020a8: 2c0c0000      cmpwi  r12,0x0
fff020ac: 41820000      beq    .+0x0

```

Figure 3 Displaying source-level and assembly code

SuperTAP configuration information

When starting a debug session you should take a quick look at the state of the debugger and emulator. This is particularly true if someone else has used the emulator between your sessions. You can easily examine or modify the emulator's state by using the Emulator Configuration window.

If you ever need to call Customer Support for assistance, the information displayed with the following commands and windows can help them resolve your call.

➤ **To display operating environment information:**

1. In the Enter Command box of the Command window, enter:

```
stat all
```

This lists details about the current operating environment.

2. From the File menu of the View status window, choose Close Window, being careful *not* to inadvertently select Exit Debugger.

➤ **To display emulator component and revision information:**

- In the Enter Command box of the Command window, enter:

```
hwconfig
```

This lists emulator component, firmware, and software revision levels and installed options.

➤ **To view the emulator configuration**

- From the Displays menu, choose Emulator Configuration.

The Emulator Configuration window appears.

You can use the Emulator Configuration window, shown in Figure 4, to view and modify the options that control the state of the debugger and emulator:

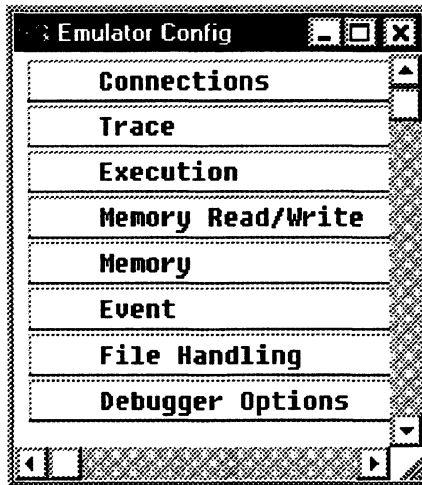


Figure 4 Emulator Configuration window

Connections The Connections button brings up the Connections window. Use this window to define and connect to emulators.

Execution The Execution button opens the Execution configuration dialog box. Use this dialog box to set the emulator execution options.

Trace The Trace button opens the Trace configuration dialog box. Use this dialog box to set emulator trace collection and display options.

Memory The Memory button opens the Memory configuration dialog box. Use this dialog box to set memory access attributes.

Event The Event button opens the Event configuration dialog box. Use this dialog box to set the emulator event system options.

File Handling The File Handling button opens the File Handling configuration dialog box. Use this dialog box to specify the upload and download format for non-IEEE-695 object files, and to other download options.

Debugger Internal The Debugger Internal button opens the Debugger Internal configuration dialog box. Use this dialog box to set input and output radix and other debugger options.

Chapter 2

Examples

This chapter provides examples demonstrating the Applied Microsystems SuperTAP system integration tool with MWX-ICE debugger. You must complete all of Chapter 1 before doing any of the examples.

The examples are organized into functional groups, with each group prefaced by a brief discussion of the function:

- Accelerate the debug of prototype hardware
- Accelerate firmware development using overlay
- Intuitive breakpoints on code and data
- Controlling code execution - breakpoints and event system
- Capturing and viewing trace history
- Examples of other time-saving features

Running the examples

How long does it take?

It takes approximately two hours to complete all the examples.

Run the examples that most interest you

All the examples can be run independent of each other. Because the examples are not interdependent, you can choose and run, in any order, the examples that most interest you.

Complete all steps within an example

You *must* complete each step within an example. The final steps of an example return settings to their original value, so you can do the examples in any order.

Accelerate the debug of prototype hardware



Crash proof debugging and testing

SuperTAP incorporates a dual-processor architecture where an *emulation processor* replaces the target processor while a second *control processor* handles communications with the debugger and monitors the target processor activity. This insures that a target processor crash will not cause the emulator to “hang”.

Built-in scope loops and memory diagnostics

A full suite of scope loops and memory diagnostic programs included with the debugger let you verify the design of your hardware and quickly isolate manufacturing defects.

Some companies use the diagnostic routines as part of their regression tests. The tests can be left running overnight to locate thermal problems. Also, the tests can run concurrent with the trace and event systems, useful for locating subtle bugs.



Monitor target conditions

SuperTAP monitors and reports target conditions such as:

- Processor clock frequency
- Target Vcc
- Bus error
- Reset asserted

Target monitoring features can isolate conditions that cause other emulators to hang. This ability helps in situations where you are debugging new and unproven hardware.

In addition, the power monitoring circuitry provides electrical protection to the target and emulator, lessening the potential for damage and reducing the risk of repairs and lost debugging time.

Example 1 – Crash-proof tool including trace history



Target system and processor initialization routines are often the most difficult code to debug, particularly on prototype targets. SuperTAP lets you execute to a breakpoint set in a target power-up or reset sequence, while collecting trace history, without hanging the debugger. This lets you debug startup and initialization code. To illustrate this, you:

- Enter Dynamic Run mode.
- Set a breakpoint at the address of the Cdaemon startup routine `_start` (a few instructions after the RESET vector fetch.
- Issue a reset command to reset the CPU. If the emulator is plugged into an Applied Microsystems demonstrator board, you can use the demonstrator board RESET button to reset the CPU.

This same procedure also works for a full target power-up sequence; you can actually cycle target power and still hit a breakpoint.

This example is written for the Applied Microsystems demonstrator board. If you are using your own target and code, you can modify the breakpoint address to the startup routine used in your code.

- **To restart the Cdaemon program and enter dynamic run mode:**
 1. From the File menu, choose Restart:
 2. In the Enter Command box, enter:

```
drun
```

Observe the LED on the demonstrator board incrementing.
- **To set an instruction breakpoint at `_start`:**
 - In the Enter Command box, enter:

```
bi _start
```

► **To run to the breakpoint:**

- In the Enter Command box, enter:

`reset`

-OR-

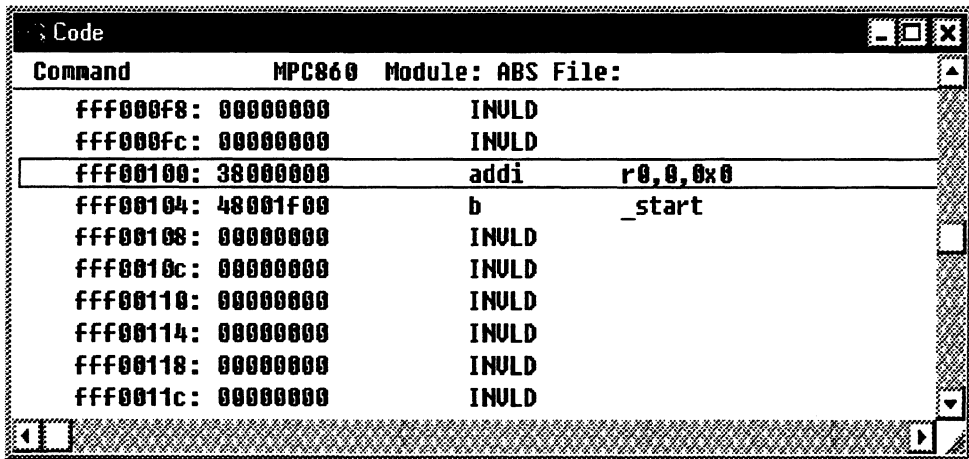
If the emulator is plugged into an Applied Microsystems demonstrator board, press then release the RESET button located on the edge of the demonstrator board.

SuperTAP breaks emulation at the beginning of the startup procedure `_start`, shown in Figure 5.

► **To clear the instruction breakpoint at `_start`:**

- In the Enter Command box, enter:

`clear 1`



The screenshot shows a window titled "Code" with a table of assembly instructions. The table has columns for "Command", "MPC860", and "Module: ABS File:". The instruction at address fff00100 is highlighted, showing the command "addi" with parameters "r0,0,0x0". The instruction at address fff00104 is "b _start". Other instructions are marked as "INULD".

Command	MPC860	Module: ABS File:
fff000f8: 00000000	INULD	
fff000fc: 00000000	INULD	
fff00100: 38000000	addi	r0,0,0x0
fff00104: 48001f00	b	_start
fff00108: 00000000	INULD	
fff0010c: 00000000	INULD	
fff00110: 00000000	INULD	
fff00114: 00000000	INULD	
fff00118: 00000000	INULD	
fff0011c: 00000000	INULD	

Figure 5 Breakpoint at START (Assembly Code window)

Example 2 – Scope loops and peek/poke trace

Built in scope loops, cyclic redundancy checks (CRC-16), and memory diagnostic programs save you from writing your own routines to test memory or to stimulate memory for use with other tools, such as an oscilloscope. When running a scope loop or memory diagnostic, the read and write cycles are captured in trace memory (peek/poke trace). This information can be used to diagnose malfunctioning target memory or I/O.

Diagnostics 2 through 7 are used to perform reads or writes of selected memory.

➤ **To perform a continuous read**

1. In the Enter Command box, enter:

```
diag 2,0xa0000000..0xa0000400
```

2. From the toolbar, choose Stop to stop the test.

Diagnostic numbers 0 and 1 perform simple and complex diagnostics on the selected memory.

➤ **To perform a complex memory test**

1. In the Enter Command box, enter:

```
diag 0,0xa0000000..0xa0000400
```

2. From the toolbar, choose Stop to stop the test.

Use the `crc` command with a range argument to perform a CRC-16 of the specified range. The command will return a hex value for the CRC.

➤ **To perform a cyclic redundancy check**

- In the Enter Command box, enter:

```
crc 0x0xa0000000..0xa0000400
```

Example 3 – Benefits of event system qualified trace



Quite often there will be only a few cycles of interest out of the millions of cycles executed in a real-time run of the code. There are two ways of dealing with this:

1. Capturing, in real-time, a trace of only the cycles of interest. This is a qualified trace.
2. Post-processing (filtering out unwanted cycles) a large and expensive trace RAM buffer full of all executed and pre-fetched instructions.

The first is by far a quicker and more accurate method than the second. With SuperTAP's four-level event system architecture and trace control actions, you can capture a qualified trace based on a complex set of conditions, including program events, target hardware events, and the CPU bus state.

The example demonstrates this ability by capturing only the first 10 memory write cycles directed to the in-memory representation of the demonstrator board LEDs.

- **To restart the Cdaemon program:**
 - From the File menu, choose Restart.
- **To configure the trace system:**
 1. From the Displays menu, choose Emulator Configuration.
 2. In the Emulator Configuration window, choose Trace.
 3. In the Trace configuration window, choose:
Collection State at Run:Don't Accumulate at Run
This keeps trace capture turned off unless enabled by the event system.
 4. In the Trace configuration window, choose:
Clear Buffer at Run:Discard Current Contents
This clears the trace buffer when the emulator enters run mode.

5. In the Trace configuration window, choose:

Collection Qualification:Bus Cycles

This configures trace for bus qualified capture.

6. In the Trace configuration window, choose Apply to accept the values, then close the window.

➤ **To configure the event system:**

1. In the Emulator Config window, choose Event.

2. In the Event configuration window, choose:

Counter # 1: Initial ValueReset to Zero at Run

This resets the counter to zero when entering run mode.

3. In the Event configuration window, choose Apply to accept the values, then close the window.

4. In the Enter Command box, enter:

```
when addr==&led_port && status==wr then trone,ctrlinc
```

Note: *led_port* is the symbol for in-memory representation of the LEDs (0x80000000). The & tells the event system to look at the address of *led_port*.

5. In the Enter Command box, enter:

```
when ctrl==10 then break
```

➤ **To run the target code:**

■ From the toolbar, choose Go.

SuperTAP breaks emulation after tracing the first 10 writes to the in-memory representation of the demonstrator board LEDs.

➤ **To display trace:**

■ From the Displays menu, choose Emulator Trace.

Only the 10 write cycles are in trace, shown in Figure 6.

- **To clear the event system:**
 - In the Enter Command box, enter:


```
whenclr all
```

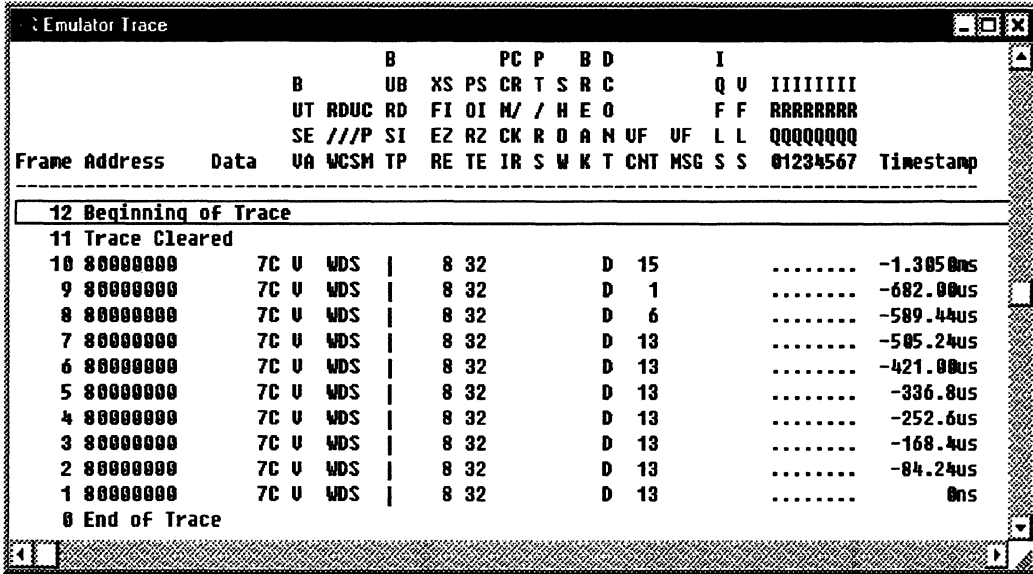


Figure 6 Emulator Trace window - qualified trace capture of the first 10 writes to the demonstrator board LEDs

Example 4 – Isolate complex bugs with the event system



Sometimes running to a basic breakpoint and examining trace history does not provide information specific enough to debug your target's code or hardware. You may also want the emulator to perform some action other than stopping code execution when the conditions become true. You may need to trigger an oscilloscope after a complex set of CPU bus cycle conditions become true; for example, the fifth write that a specific subroutine makes to a certain memory location. At other times you may want to trace only specific types of bus cycle information under certain conditions.

The event system supplies the mechanism to define conditions and take actions. This mechanism allows the emulator to perform various actions based on conditions of complexity far surpassing that of simple breakpoints.

The event system is used primarily for debugging nested or sequential problems. By defining a logical sequence of events, you can locate a hard to find bug that may exist only under a particular set of circumstances.

The event system is implemented with emulator hardware and can be used in both RAM and ROM regions.

Monitor complex sets of conditions

SuperTAP event system comparators comprise 80 bits of information, including address, data, status, and external signals. The comparators can use “don't-care” masks and can cover ranges. Using these comparators, you can monitor a complex set of conditions, including program events, target hardware events, and the CPU bus state.

Do more than just break

When the event system conditions have been met, SuperTAP can perform a variety of actions including:

- Break emulation asynchronously or synchronously

- Trace one or many CPU bus cycles
- Set/increment/reset memory or register
- Set or reset event system flags
- Switch between event system groups
- Enable or disable timestamp
- Call a user specified function
- Generate a trigger output to an external instrument

The example is a simple case requiring more than a breakpoint. You set up the event system so that only the cycles within a certain function `outled()` are traced. To do this, you define two events to start trace capture when `outled()` is entered, and stops trace capture when `outled()` transfers program execution.

➤ **To restart the Cdaemon program:**

- From the File menu, choose Restart.

➤ **To set up initial trace conditions:**

1. From the Displays menu, choose Emulator Configuration.
2. In the Emulator Configuration window, choose Trace.
3. In the Trace configuration window, choose:

Collection State at Run:Don't Accumulate at Run

This keeps trace capture turned off unless enabled by the event system.

4. In the Trace configuration window, choose:

Clear Buffer at Run:Discard Current Contents

This clears the trace buffer when the emulator enters run mode.

5. In the Trace configuration window, choose:

Collection Qualification:Cycles needed for Disassembly

This configures trace to capture information for disassembly.

6. In the Trace configuration window, choose Apply to accept the values, then minimize the window.

► **To set up event statements to start and stop tracing:**

1. In the Enter Command box, enter:

```
ps outled
```

From the output of the **printsymbols (ps)** command, you can easily determine the address range of the function **outled()**.

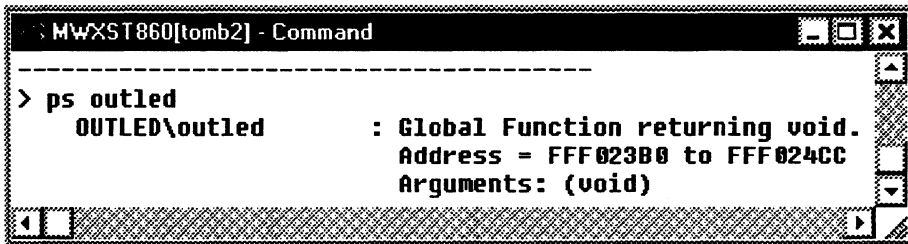


Figure 7 Printsymbols output for outled()

2. In the Enter Command box, enter:

```
when addr==outled then tron
```

This event statement turns tracing on at the start address of the function **outled()**. Notice a View window is opened when you enter the command.

3. In the Enter Command box of the Command window, enter:

```
when addr==0xffff024cc then troff,break
```

This turns tracing off at the end of the function **outled()**, then breaks emulation. (Note: In this case the **troff** is not really necessary since trace capture always stops when a break occurs. **Troff** is most useful when you want to capture a trace of something while continuing to emulate.)

► **To run until the break condition is met:**

■ From the toolbar, choose Go.

➤ **To view trace in the Emulator Trace window:**

1. From the Displays menu, choose Emulator Trace.

This opens the Emulator Trace window with the default display mode (Raw Display) active.

2. In the Display Format box, select the Assembly and Source checkboxes.

Only the source lines and their associated assembly code for the function `outled()` are displayed in the trace buffer window, shown in Figure 8.

➤ **To clear the event system:**

- In the Enter Command box, enter:
`whenc1r all`

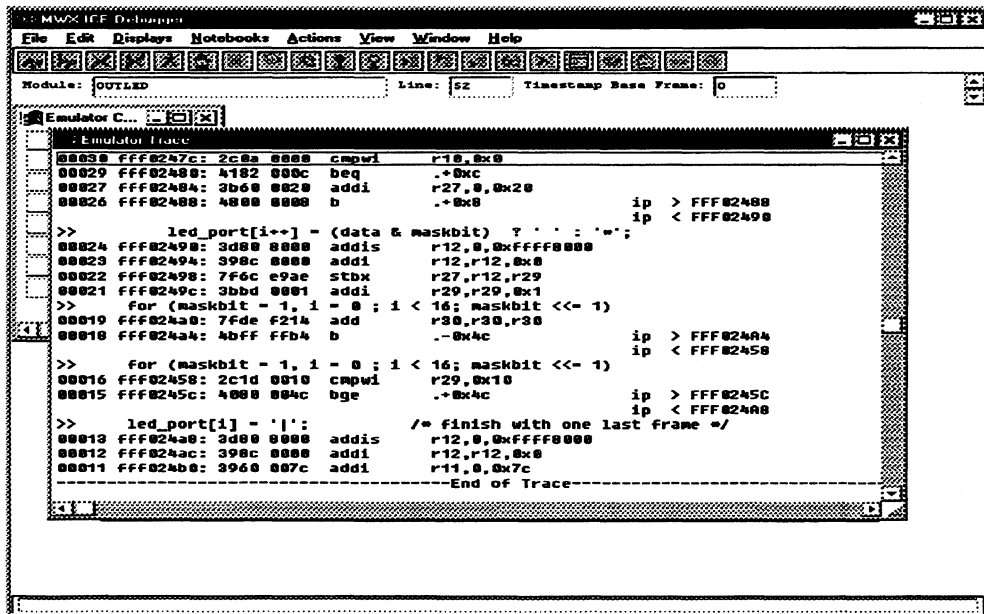


Figure 8 Emulator Trace window - Trace of `outled()`

Accelerate firmware development using overlay memory

Description



Real-time code execution vehicle

Overlay provides stable substitute memory into which you can load your code and data. Using overlay, you can begin debugging your code before target memory hardware is operational. Overlay memory accelerates the code debug, re-make, and test cycle. Using the high-speed net connection (5 MB/sec) you can quickly download your fixed code into overlay rather than burn a new EEPROM.

SuperTAP can be equipped with an optional 1 MB, 2 MB, or 4 MB of overlay memory (zero wait states to 25 MHz, 1 wait state to 40 MHz).

Overlay memory can be allocated in minimum 128Kb segments (a 128Kb *granularity*). Overlay supports 8, 16, or 32 bit ports and DMA accesses by external bus masters.

Copy memory contents between the target and overlay

SuperTAP lets you easily copy the contents of target memory into overlay memory or from overlay into target memory. You can use this capability when you need to copy the contents of your target ROM (or PROM) into overlay memory, to avoid burning a new ROM. Also, copying the contents of target ROM into overlay lets you set software breakpoints within ROM memory space.

Example 5 – Using overlay memory

To use overlay memory you assign (map) it to address ranges and access types (read/write or read-only). Overlay memory can replace target memory or substitute for memory that does not exist in the target.

In this example you view the overlay memory map set up by the `cdemon.inc` include file.

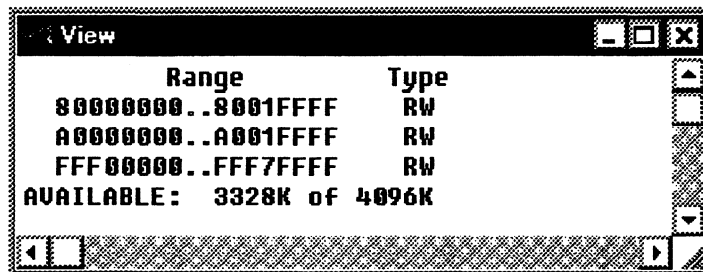
► **To view the current overlay memory map:**

1. In the Enter Command box of the Command window, enter:

```
map
```

Observe the current overlay map, shown in Figure 9. All three ranges of mapped overlay were assigned Read/Write access and there is 256Kb of overlay still available for mapping.

2. Close or minimize the View Memory Map window.



Range	Type
80000000..8001FFFF	RW
A0000000..A001FFFF	RW
FFF00000..FFF7FFFF	RW

AVAILABLE: 3328K of 4096K

Figure 9 Overlay memory mapped by `cdemon.inc` include file

Controlling code execution

When debugging code, the ability to control code execution is essential. SuperTAP's breakpoints and event system let you:

- Stop asynchronously at any time
- Execute code until reaching a particular location or satisfying a set of conditions
- Single step lines of source or assembly code
- Step into or over function calls

To provide this ability SuperTAP supports four types of breakpoints:

- Asynchronous
- Hardware execution and access
- Software execution
- External

Do more than just break

When breakpoint conditions are met, SuperTAP can perform any of the following actions:

- Stop emulation (break)
- Execute a C expression
- Log the value of an expression in a file
- Execute a debugger macro

Description

Asynchronous breaking capability lets you stop code execution at any time by clicking the MWX-ICE Stop button. This ability can help in situations where you need to stop a program that is executing incorrectly and has bypassed a breakpoint you set.

Hardware execution and access breakpoints use the SuperTAP's hardware and do not consume any target resources. When a memory access occurs that matches the breakpoint condition, microprocessor execution stops. Hardware execution and access breakpoints can be set over target RAM or ROM.

Software breakpoints replace the instructions in the target program with a special opcode that forces a specific behavior in the microprocessor. When the breakpoint occurs, SuperTAP halts execution and places the original instruction back into memory.

External breakpoints allow an external trigger-in signal from the target or from a piece of test equipment, such as a logic analyzer, to cause the SuperTAP to “break” out of emulation. Or, the SuperTAP can generate a trigger-out signal to trigger a logic analyzer or storage scope.

SuperTAP provides one BNC trigger input and one BNC trigger output pin to support both types of external breakpoints.

Example 6 – Intuitive breakpoints on code and data

Software breakpoints must be located in RAM, so that the special opcode may be written to target memory. This poses no problem, since SuperTAP's overlay memory can easily be used for setting breakpoints where no target RAM exists.

MWX-ICE offers a variety of methods that make setting and clearing breakpoints easy. In this example, you sample the various methods including:

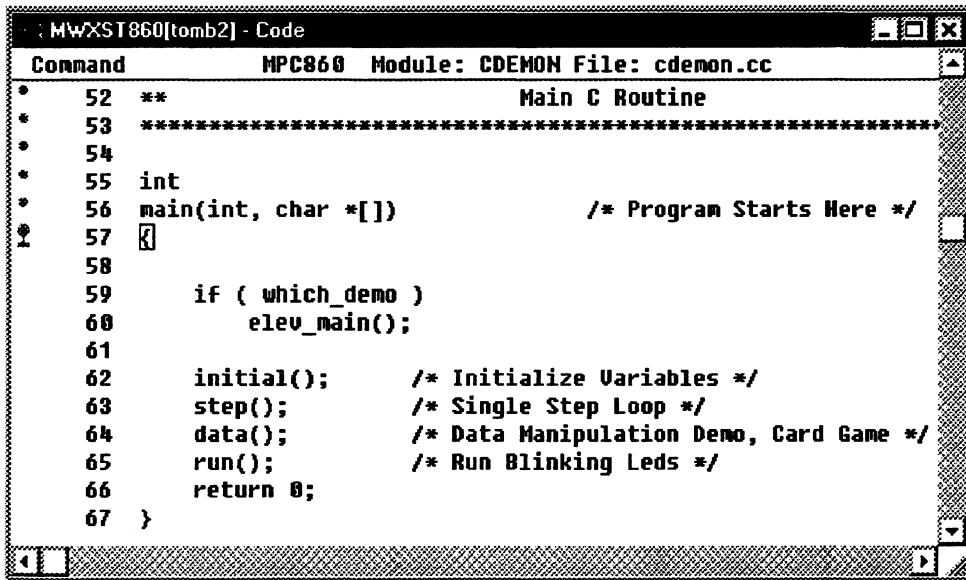
- Using the Execution Control notebook
 - Choosing the BreakI button with a source line selected
 - Dragging the BreakI button to a source line
 - Using the Breakpoints window
-
- **To restart the Cdaemon program:**
 - From the File menu, choose Restart.
 - **To set a permanent breakpoint using a notebook:**
 1. From the Notebooks menu, choose Execution Control.
 2. From the Execution Control menu, choose Set and clear breakpoints.
 3. In the Break page Start Address box, type:
`main`
 4. Choose the Set Break button, to set the breakpoint.

Notice that “BreakInstruction main” is echoed in the Command window. By observing and remembering the commands generated while using notebooks and buttons, you can quickly start using the command line interface.

5. From the toolbar, choose the Go button to run to the breakpoint.

Notice that the Code window box indicating the current execution line has moved to the beginning of main(). Also notice the breakpoint icon (a small stop sign) in the left side of the current execution line, shown in Figure 10.

6. In the Execution Control notebook, choose Close to close the window.




```
Command      MPC860  Module: CDEMON File: cdemo.cc
52  **                               Main C Routine
53  *****
54
55  int
56  main(int, char *[])              /* Program Starts Here */
57  █
58
59      if ( which_demo )
60          elev_main();
61
62      initial();                    /* Initialize Variables */
63      step();                       /* Single Step Loop */
64      data();                       /* Data Manipulation Demo, Card Game */
65      run();                        /* Run Blinking Leds */
66      return 0;
67  }
```

Figure 10 Breakpoint at beginning of main()

► **To set a permanent breakpoint using the BreakI button and a selected line:**

1. In the Source Code window, select the source line containing the step() function by double-clicking anywhere in that line.

When selected, there will be at least one character of the line highlighted.

2. Choose the BreakI button  to set the breakpoint.

Notice the breakpoint icon displayed to the left of the step() source line.

3. From the toolbar, choose Go to run to the breakpoint.

► **To set a permanent breakpoint dragging the BreakI button to a source line:**

1. In the Source Code window, click and hold the BreakI button and drag the breakpoint icon to the source line containing the data() function.

Notice the breakpoint icon displayed to the left of the data() source line.

2. From the toolbar, choose Go to run to the breakpoint.

► **To run to a specific source line (temporary breakpoint):**

1. In the Source Code window, select the source line containing the run() function by double-clicking anywhere in that line.
2. From the toolbar, choose the GoUntil button (man running to stop sign) to run to the run() function.

► **To view permanent breakpoints:**

- From the Displays menu, choose Breakpoints.

Notice the three breakpoints displayed in the window; the top one set at main() using the Execution Control notebook, the second one set at step() by choosing the BreakI button with a line selected, and the third set at data() by dragging the BreakI button. The Breakpoints window is shown in Figure 11.

The screenshot shows a window titled "Breakpoints" with a table of breakpoint information. The table has four columns: Address, Type, Symbol, and Command. There are three rows of data, each with a small icon to its left.

	Address	Type	Symbol	Command
📌	FFF02098	Instr	\CDEMON\main\#57	main
📌	FFF020B8	Instr	\CDEMON\main\#63	\CDEMON\#63:4
📌	FFF020BC	Instr	\CDEMON\main\#64	\CDEMON\#64:3

Figure 11 Displaying information about a breakpoint

► **To clear all permanent breakpoints:**

1. In the Enter Command box of the Command window, enter:
`clear`
2. Close the Breakpoints window.

Example 7 – Hardware execution and access breakpoints

Hardware execution breakpoints use the SuperTAP's hardware and do not consume any target resources. Unlike software execution breakpoints, which must be set in RAM, hardware execution breakpoints can be set over RAM or ROM. Using either method, execution stops when the processor executes the instruction.

Access breakpoints are useful for detecting errant accesses to memory locations. For example, you may want to break emulation if your code attempts to write over a constant pointer.

In the example, you set a write access breakpoint (**bw**) at the memory address (`&led_port` is 0x8000000) of the in-memory representation of the demonstrator board LEDs. Read access (**br**) and read-or-write access (**ba**) breakpoints are set similarly.

➤ **To restart the Cdaemon program:**

- From the File menu, choose Restart.

➤ **To set a write access breakpoint:**

1. In the Enter Command box, enter:

```
bw &led_port
```

2. From the toolbar, choose Go to run to the breakpoint.

SuperTAP breaks emulation at the initialization routine's write to 0x8000000 (`led_port`'s first address), shown in Figure 12.

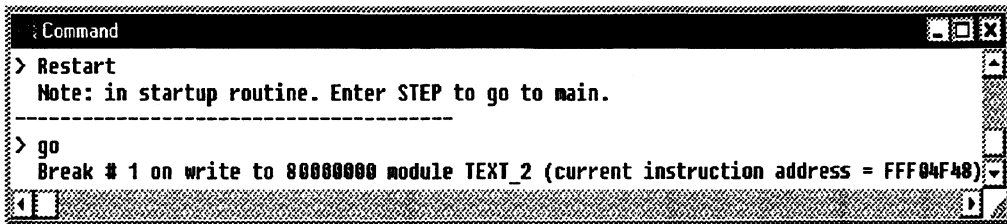


Figure 12 Break on initialization (write) of *led_port*

3. From the toolbar, choose Go to run to the breakpoint.

SuperTAP breaks emulation in the `outled()` function at the write to the in-memory representation of the demonstrator board LEDs.

➤ **To monitor the variable *led_port*:**

1. From the Displays menu, choose Data.
2. In the Data window Expression field, enter:
`led_port`
3. From the Data window View menu, choose Show (char*) as String.
4. From the toolbar, choose Go, to run to the breakpoint a few times while you watch the LEDs on the demonstrator board increment or the variable *led_port* increment in the Data window.
5. Close the Data window.

➤ **To clear the write access breakpoint:**

- In the Enter Command box, enter:
`clear 1`

Capturing and viewing trace history

With trace history you can capture and record, in real-time, the execution history of the processor as SuperTAP executes the target program. Using trace history, you can verify the correct performance of the software and find errors that may occur in the program's execution. Trace lets you move from the environment of "hunt-and-peck" to one where real evidence is easily captured and displayed, helping you isolate and correct the bug.

The intelligent trace disassembly feature - an industry first - dramatically increases productivity by displaying instructions correlated with register values and bus cycles.

Description

Trace history offers 32K x128 bits of information including all active MPC860 signals and 32 bits of timestamp information. You can display raw bus cycles, full source-level, assembly-level, or mixed source and assembly-level trace information with MWX-ICE's four trace display modes.

Capturing "qualified" trace versus filtering continuous trace

Quite often there will be only a few cycles of interest out of the millions of cycles executed in a real-time run of the code. There are two ways of dealing with this:



1. Capturing, in real-time, a trace of only the cycles of interest. This is a qualified trace.
2. Post-processing (filtering out unwanted cycles) a large and expensive trace RAM buffer full of all executed and pre-fetched instructions.

The first is by far a quicker and more accurate method than the second.

With a four-level event system architecture and “trace one cycle” action, SuperTAP lets you capture a qualified trace based on a complex set of conditions, including program events, target hardware events, and the CPU bus state.



Execute and trace target power-up or reset sequences

You can execute a target power-up or reset sequence while collecting trace history, without hanging the debugger. With this you can more easily debug startup code.



Dynamic trace display

SuperTAP allows you to view and upload trace history without stopping or even pausing emulation. This means you can view your program's activity without disturbing its real-time operation. Example 9 in the Event System section of this guide demonstrates this ability.

Timestamp

Timestamp information provides an accurate measurement of elapsed time between cycles displayed in trace. SuperTAP provides timestamp information to 40 nsec.

Saving trace history in a file

The log command lets you easily store trace history in a file by naming a Journal file in an MWX-ICE notebook then simply displaying trace. You can even log trace to a file *without* stopping emulation.

After trace is saved, you can edit the file to add comments for future reference. For example, comments may be added to aid in documenting failure conditions.

Example 8 – Intelligent trace disassembly



The “intelligent” trace disassembler boosts productivity by providing insight into the operation of the processor. The disassembler tracks the state of processor registers, letting you easily see how data is passed during code execution.

➤ **To restart the Cdeemon program:**

- From the File menu, choose Restart.

➤ **To set up initial trace conditions:**

1. From the Displays menu, choose Emulator Configuration.
2. In the Emulator Configuration window, choose Trace.
3. In the Trace configuration window, choose:

Collection State at Run:Begin Accumulating at Run

4. In the Trace configuration window, choose:

Clear Buffer at Run:Discard Current Contents

This clears the trace buffer when the emulator enters run mode insuring that all cycles captured in trace are from the current code execution.

5. In the Trace configuration window, choose Apply to accept the values, then minimize the window.

➤ **To execute to the function step():**

1. In the Code window, scroll down to the step() function.
2. Select that line by double-clicking the function step().
3. From the toolbar, choose GoUntil.

➤ **To view trace in the Emulator Trace window:**

1. From the Displays menu, choose Emulator Trace.

This opens the Emulator Trace window with the default display mode (Raw Display) active. Raw trace display includes address, data, active processor signals, and timestamp infor-

mation. The Frame numbers provide an index of where the cycle occurred in execution history, the lowest Frame number being the most recent instruction executed.

2. From the View menu, deselect Raw and select Assembly and Source.

The source lines and their associated assembly code for program execution to the function step() are displayed, shown in Figure 13. The Emulator Trace Action menu provides many useful trace utilities that let you easily:

- Set breakpoints using trace information
- Set the current scope based on trace frame
- Search trace for any string patterns
- Scroll through the trace buffer
- Clear the trace buffer
- Change trace display configuration
- Change timestamp format and offset base frame

```

Emulator Trace
>> for (maskbit = 1, i = 0 ; i < 16; maskbit <<= 1)
00032 fff02458: 2c1d 0010 cmpwi    r29,0x10
00031 fff0245c: 4000 004c bge     .+0x4c          ip > FFF0245C
                                ip < FFF024A8
>> led_port[i] = '|';          /* finish with one last frame */
00029 fff024a8: 3d80 0000 addis   r12,0,0xffff8000
00028 fff024ac: 398c 0000 addi    r12,r12,0x0
00027 fff024b0: 3960 007c addi    r11,0,0x7c
00026 fff024b4: 7d6c e9ae stbx    r11,r12,r29
>> }
00025 fff024b8: bb61 000c lww     r27,0x000c(r1)
00023 fff024bc: 8001 0024 lwz     r0,0x0024(r1)
00021 fff024c0: 7c08 03a6 mtspr   lr,r0
00019 fff024c4: 3021 0020 addi    r1,r1,0x20
00013 fff024c8: 4e00 0020 blr     ip < FFF0211C
>> }
00011 fff0211c: 8001 000c lwz     r0,0x000c(r1)
00010 fff02120: 7c08 03a6 mtspr   lr,r0
00008 fff02124: 3021 0000 addi    r1,r1,0x8
00007 fff02128: 4e00 0020 blr     ip < FFF020B8
>> step();          /* Single Step Loop */
-----Execution Breakpoint-----
-----End of Trace-----

```

Figure 13 Emulator Trace window - mixed source and assembly trace

Example 9 – Capturing and viewing trace while running



In this example, SuperTAP emulates in dynamic mode; you can display trace while still emulating. You capture the `run()` function in trace using the event system, then view the trace in raw and disassembled formats.

➤ **To restart the Cdaemon program:**

- From the File menu, choose Restart.

➤ **To set up the initial trace condition:**

1. From the Displays menu, choose Emulator Configuration.
2. In the Emulator Configuration window, choose Trace.
3. In the Trace configuration window, choose:

Collection State at Run:Don't Accumulate at Run

This keeps trace capture turned off unless enabled by the event system.

4. In the Trace configuration window, choose:

Clear Buffer at Run:Discard Current Contents

This clears the trace buffer when entering run mode.

5. In the Trace configuration window, choose:

Collection Qualification:Cycles needed for Disassembly

This configures trace for bus qualified capture.

6. In the Trace configuration window, choose Apply to accept the values, then minimize the window.

➤ **To set up event system:**

1. In the Enter Command box, enter:

```
when addr==outled then tron
```

2. In the Enter Command box, enter:

```
when addr==0xffff024c4 then troff
```

The View Event System window should display the two event system statements.

➤ **To enter dynamic run mode:**

■ In the Enter Command box, enter:

```
drun
```

➤ **To display a “snapshot” of trace while still running target code:**

1. In the Enter Command box, enter:

```
drt
```

This displays trace history in raw format in the Command window.

2. In the Enter Command box, enter:

```
dtb
```

This displays trace history in interleaved source and assembly format in the Command window.

Notice in Figure 14 that the register values on the right are matched to the source lines on the left. This is the “intelligent” trace disassembly feature.

```

Command
ip < FFF0240C
>>      led_port[i++] = (data & maskbit) ? ' ' : '*';
00025 fff0240c: 3b60 002a  addi    r27,0,0x2a
>>      led_port[i++] = (data & maskbit) ? ' ' : '*';
00024 fff02490: 3d80 8000  addis   r12,0,0xffff8000
00023 fff02494: 398c 0000  addi    r12,r12,0x0
00022 fff02498: 7f6c e9ae  stbx    r27,r12,r29
00021 fff0249c: 3bdd 0001  addi    r29,r29,0x1
>>      /* Build ascii chars in led port array from pattern */
>>      for (maskbit = 1, i = 0 ; i < 16; maskbit <<= 1)
00019 fff024a0: 7fde f214  add     r30,r30,r30
00018 fff024a4: 4bff ffb4  b       .-0x4c          ip > FFF024A4
                                ip < FFF02458
>>      for (maskbit = 1, i = 0 ; i < 16; maskbit <<= 1)
00016 fff02458: 2c1d 0010  cmpwi   r29,0x10
00015 fff0245c: 4000 004c  bge     .+0x4c          ip > FFF0245C
                                ip < FFF024A8
>>      for (maskbit = 1, i = 0 ; i < 16; maskbit <<= 1)
>>      {

```

Figure 14 Dynamic trace display - disassembled source and assembly

➤ **To exit dynamic run mode and clear the event system:**

1. In the Enter Command box, enter:

```
dstop
```

2. In the Enter Command box, enter:

```
whenclr all
```

This clears all event system statements.

Examples of other time-saving features

Description

This section contains examples of other useful, time-saving features of SuperTAP with MWX-ICE:

- Configure, understand, and debug peripheral registers
- Displaying high-level data structures
- Monitoring and modifying variables dynamically
- Displaying and modifying memory

Example 10 – Configuring/debugging peripheral registers



MWX-ICE includes the CPU Browser™ and Register Browser™ windows, to save you time you might otherwise spend referencing a technical manual for register bit meanings. The POR 0 PCMCIA Option 0 window, is shown in Figure 15.

► **To browse the register:**

1. From the Displays menu, choose CPU Browser.
2. From the CPU Browser Menu, choose PCMCIA Interface.
3. From the CPU Browser window, choose POR0 PCMCIA Option 0.

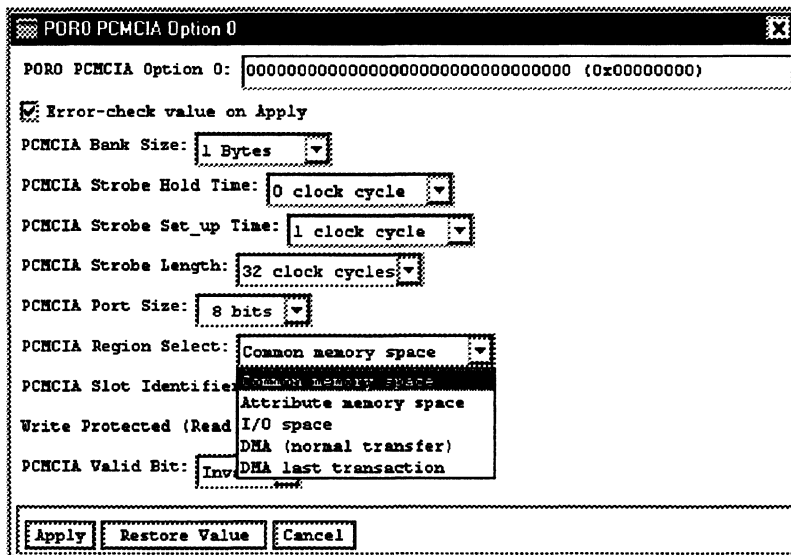


Figure 15 Browsing a register

Example 11 – Displaying high-level data structures

The MWX-ICE Inspector window lets you view complex high-level data structures and quickly traverse linked lists.

- **To restart the Cdemon program:**
 - From the File menu, choose Restart.

- **To run to `house()`, a function in `data()`:**
 - In the Enter Command box, enter:
`g DATA\house`
Emulation breaks at the beginning of `house()`, a function within `data()`. At this point, all of the cards in the blackjack game have been dealt.

- **To display the structure *players*:**
 1. From the Displays menu, choose Inspector.
 2. In the Inspect Symbol or Expression box, enter the following:
`players`
The Inspector window displays a break down of *players*. You can scroll or resize the Inspecting window for the best display of the structure.
 3. From the Inspector window View menu, choose Show (char*) as String.
 4. From the Inspector window, choose S>> for *player[01]*.
The Inspector window displays a break down of *player[01]*, the second element of *players*.
In Figure 16, you can see *player1* “Warf” won, having been dealt 2 cards for a total of 21 points.
 5. Close the Inspector window.

Tag	Type	Name	Value	Comment (if any)
S>>	player	*(player *) 0x800003E4		// (players)[01]
	QUOTED STRING	.name	"Warf"	
[]>	card[5]	.cards	0x800003EE	
	int	.points	21	
	int	.dealt	2	

Figure 16 Displaying the structure *players*

Example 12 – Modifying variables dynamically



MWX-ICE provides windows for working with program variables.

- Data window, for monitoring variables.
- Inspector window, for viewing and modifying variables.
- Register window, for viewing and modifying register-based variables.

In static mode, they are updated only at each single-step, breakpoint, or program halt. In dynamic mode, emulation periodically pauses then re-starts, updating each window when emulation is re-entered.

- **To restart the Cdaemon program:**
 - From the File menu, choose Restart.
- **To enter dynamic run mode:**
 - In the Enter Command box, enter:
`drun`
- **To dynamically monitor the variable *led_port*:**
 1. From the Displays menu, choose Data.
 2. In the Data window Expression box, enter the following:
`led_port`
This places the variable *led_port* in the Data window, shown in Figure 17.

- **To enter dynamic update mode:**
 - In the Enter Command box, enter:
`dupdate 1000`
 - This causes MWX-ICE to continuously poll the emulator for the value of *led_port*.
 - Observe the variable *led_port* in the Data window and the LEDs on the evaluation board incrementing.

- **To exit dynamic update mode:**
 - From the toolbar, choose Stop to exit dynamic update mode.



Figure 17 Dynamic mode - Monitoring the variable *led_port* dynamically

- **To dynamically modify the variable *direct*:**
 1. From the Displays menu, choose Inspector.
 2. In the Inspect Symbol or Expression box, enter the following:
`direct`
 - The variable *direct* controls the direction of the demonstrator LED's counting, either *left* or *right*.

3. In the Inspector window local menu, choose the long button to the left of the value *left*.

This opens up a dialog box used to change the value of the inspected variable, shown in Figure 18.

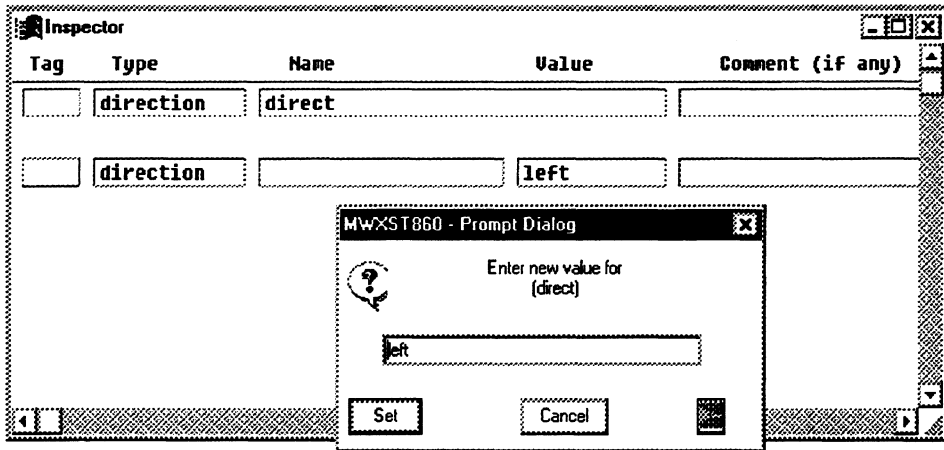


Figure 18 Dynamic mode - Entering a variable's new value dynamically

4. In the Enter new value field of the dialog box, type:
right
5. In the dialog box, choose Set to accept the new value.

The Inspector window for *direct* should reflect the new value *right*, shown in Figure 19. Observe that the demonstrator LEDs are now decrementing.

Tag	Type	Name	Value	Comment (if any)
	direction	direct		
	direction		right	

Figure 19 Dynamic mode - Modifying a variable dynamically

► **To exit dynamic run mode and return to pause mode:**

1. In the Enter Command box, enter:

`dstop`

This forces SuperTAP from dynamic run mode back into pause mode.

2. Close the Data and Inspector windows.

Example 13 – Displaying and modifying memory

MWX-ICE provides means for acting on a block of memory. Using either the command line or the Memory Commands notebook, you can clear, move, set, read, write, or log a block of memory.

➤ **To restart the Cdaemon program:**

- From the File menu, choose Restart.

➤ **To display memory:**

1. From the Displays menu, choose Memory.
2. In the Start Address box, enter:

```
&led_port
```

This displays a block of memory beginning with the address of *led_port*.

3. From the View menu, choose Show ASCII.

This displays the ASCII values for the corresponding memory data.

➤ **To manipulate a block of memory:**

- From the Notebooks menu, choose Memory Commands.

The Memory Commands menu provides the following tools for working with memory:

- Copy—copy the contents of one block of memory to another.
- Fill—fill memory with a given value.
- Compare—compare the contents of two memory blocks.
- Search—search through memory for a pattern.
- Examine stack—display values from a particular stack level.

Conclusion



Thank you for using the Applied Microsystems SuperTAP.

Now that you have completed the demonstration of SuperTAP with MWX-ICE debugger, you can refer to the MWX-ICE *User's Manual* for more information about using the debugger.

If this has been a SuperTAP pre-sales demonstration, please contact your local sales office for pricing and ordering information.

Appendix A

Cdemon Demonstration Program

Cdemon is the Applied Microsystems standard C/C++ demonstration program, providing examples of many code and data constructions used by programmers. An in-memory representation of the LEDs (*led_port*) may be used to see the output of some of the functions.

Cdemon is composed of two discrete programs. The default C program writes to the LEDs and plays a simple hand of blackjack. The other, a C++ program, simulates an elevator. A variable named *which_demo* determines which of the two demonstration programs is executed. The card game program runs by default. If *which_demo* is set to 1, then the elevator program is executed.

A functional block of the LED-lighting/blackjack game is shown in Figure 20.

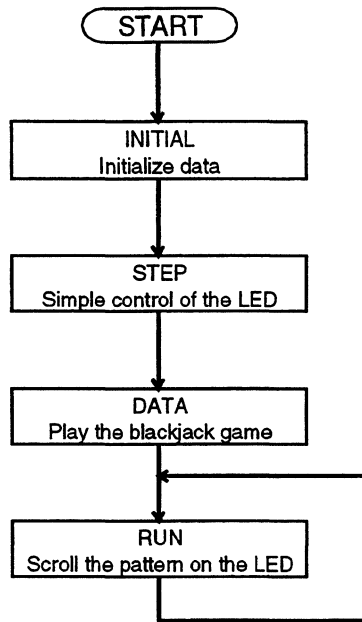


Figure 20 Flowchart of Cdemon

initial()

The function `initial()` initializes the three global LED-control variables, *pattern*, *speed*, and *direct*. These variables control the byte pattern, the speed, and the direction of LED pattern rotation, respectively.

After initializing the global variables, `initial()` passes control to `step()`.

step()

The function `step()` performs five loops of a simple LED control process, and passes control to `data()`.

`Step()` declares the loop-control variable *loops*. In each of its five loops, `step()` calls `outled()` 17 times, each time passing `outled()` a one-byte argument which represents the pattern displayed on the demonstration board LEDs.

`Outled()` then writes the pattern to the LEDs (pointed to by *dot_port*) and to an in-memory representation of the LEDs (symbolically named *led_port*). Each of the 17 arguments passed to `outled()` by each loop in `step()` represents one LED pattern.

While the execution of `step()` can be observed with the trace function, the purpose of `step()` is to demonstrate, in a single-stepped fashion, the relationship between the code and the LEDs.

The most basic control of `step()` comes from single-stepping while observing and modifying the loop-control variable *loops*. *Loops* may be observed with the Data window, and observed or changed with the Inspector window. Setting *loops* to a high value will lengthen the time spent in `step()`, while setting *loops* to zero will very quickly cause program control to be passed to `data()`.

Because `step()` produces a repeating cycle of data on the bus, predictable data-value conditions are available to the event system.

`data()`

The function `data()` plays a five-handed game of blackjack with four players and a dealer. When the game is won, `data()` passes control to `run()`. `Data()` requires no input and generates no output, and is simply a code environment with interesting data structures.

The primary data structures are:

1. *card*, a structure defining the value and suite of each card.
2. *player*, a structure containing each player's name, a hand (array) of five cards, a point total, and a card count.

3. *card_deck*, a union with various array types describing the cards in the four suits, the cards in the two sub-decks for shuffling, and the cards in the shuffled deck.

After the declarations, `data()` initializes player names and sets cards dealt and points for each player to zero, then executes the following functions.

init()

This function initializes players and dealer structures.

sort()

This function sets up a 52-word block of memory as a deck of cards.

shuffle()

This function divides the deck into two 26-word blocks and interleaves them, simulating the shuffling process.

deal()

This function deals one card to each player, including the dealer.

hit()

This function deals cards to each player until *points* \geq 18, or cards dealt = 5.

house()

This function deals cards to the dealer until *points* \geq 17, or cards dealt = 5.

The players each draw for cards while they have less than 21 points and more than 18 points. The dealer uses a similar routine to draw cards until his hand contains more than 17 points. The game concludes after one round.

The `data()` function only executes once. To replay the game, reset the program to return to the beginning of the code, and run the code until it reaches a temporary breakpoint set at the beginning of the `data()` function.

run()

The function `run()` writes a string from left to right (or right to left, depending on the value of the variable *direct*) to the LED's endlessly. Rather than using separate statements like `step()`, `run()` uses a "while" control structure under the direction of the *speed* and *direct* variables from `initial()`. Program control stays with `run()`.

`Run()` declares external functions `outled()` and `wait()`, declares the byte *maskbit*, the integer *cputype*, the loop-control variable *i*, and the constant *forever = 1*.

outled()

This function writes an 8-bit value to the LEDs (pointed to by *dot_port*) and to *led_port*, the in-memory representation of LEDs.

wait()

This function sets the actual delay according to the value of two arguments, *cputype* and *speed*.

The mechanics of `run()` can be observed by changing the values of *direct* and *speed*, and then running without breakpoints. The effects of changed variables in the LED-control task can be observed directly at the LEDs or at the Data window while monitoring *led_port*.

Index

A

Applications support
 contacting 2

B

Breakpoint
 access 42
 asynchronous 41
 deleting 46
 external 42
 hardware execution 42
 hardware, example 47, 48
 software 42
 temporary, example 43, 45

C

Cdemon
 data() 69
 detailed description 67
 direct variable 61
 initial() 68
 led_port 69
 led_port variable 60
 loading 21
 outled() 71
 primary data structures 69
 run() 71
 step() 68
 wait() 71
Customer Support
 contacting 15, 18

D

Debugger

 features 15

Demonstration
 running in your own target 3
Demonstration code
 detailed description 67
 loading 21
digital ground 6
Directory Chooser 21
Dynamic trace 50

E

Electrical characteristics 14
Electrostatic discharge 7
EMI 5
Emulation
 stopping and leaving run mode 21
ESD 7
Event system
 actions possible 35
 comparators 35

F

FCC,EMC 5
File Chooser 21

G

ground 6

H

Help 20
 command line 20
 customer support, contacting 2

I

Interference 5

M

Mapping

overlay memory, example 40

Memory

displaying and modifying, example 64

mapping, example 40

overlay 39

MWX-ICE debugger

features 15

O

Online help 20

Overlay memory

copying between overlay and target 39

parallel writes to target 3

size, 1 MB, 2 MB, or 4 MB 39

P

Pause mode

definition 21

Power-up

tracing during target power-up 50

Q

Qualified trace 49

R

Radio interference 5

Reset

tracing during reset 50

S

safety ground 6

Static-sensitivity 7

Structures

displaying, example 58

SuperTAP

configuration information 24

Support

customer 2

T

Target monitoring features 28

Timestamp 50

Trace history

description 49

dynamic 50

qualified 49

qualified versus filtered 49

V

Variables

dynamically monitoring, example 60



Applied Microsystems Corporation

Applied Microsystems Corporation maintains a worldwide network of direct offices committed to quality service and support. For information on products, pricing, or delivery, please call the nearest office listed below. In the United States, for the number of the nearest local office, call 1-800-426-3925.

CORPORATE OFFICE

Applied Microsystems Corporation
5020 148th Avenue Northeast
P.O. Box 97002
Redmond, WA 98073-9702
(206) 882-2000
1-800-426-3925
Customer Support
1-800-ASK-4AMC (1-800-275-4262)
TRT TELEX 185196
FAX (206) 883-3049
Internet Home Page: <http://www.amc.com>

EUROPE

Applied Microsystems Corporation Ltd.
AMC House
South Street
Wendover
Buckinghamshire
HP22 6EF United Kingdom
44 (0) 296-625462
Telex 265871 REF WOT 004
FAX 44 (0) 296-623460

JAPAN

Applied Microsystems Japan, Ltd.
Arco Tower 13 F
1-8-1 Shimomeguro
Meguro-ku
Tokyo 153, Japan
81-3-3493-0770
FAX 81-3-3493-7270

Part No.	Revision History	Date
924-00100-00	Initial release of <i>Evaluation Guide</i> (PC) for SuperTAP for the MPC860 Family.	3/97

