*Uni*ROM
support files for
Paradigm Debug RT

USER'S MANUAL

Release 2.0

TechTools
PO BOX 462101 Garland,TX 75046-2101
Voice: (972) 272-9392 FAX: (972) 494-5814 email: support@tech-tools.com

Introduction

Thank you for selecting a TechTools product. We have made every attempt to provide a quality product at an affordable price. Our goal is to provide tools for the Engineer and Technician that are inexpensive, but fully functional. If you have any problems or comments, please don't hesitate to call or FAX and let us know.

The Paradigm Support files allow one to combine the power of *Uni*ROM with that of Paradigm DEBUG. This powerful combination provides inexpensive, Source-level, Symbolic debugging of x86 and NEC V-series based embedded systems with minimal target intrusion.

The Paradigm Support files from TechTools provide an interface between Paradigm DEBUG and *Uni*ROM, enhancing the capabilities of Paradigm DEBUG with inexpensive Hardware support. *Uni*ROM adds the following enhancements to Paradigm DEBUG:

Allows Breakpoints in EPROM.

Unlike a traditional ICE, *Uni*ROM plugs into the target EPROM or FLASH socket rather than the CPU socket. This approach permits full use of Software Breakpoints in the EPROM space, eliminating the need for expensive overlay memory or Hardware Breakpoints. Software breakpoints are more versatile than Hardware breakpoints and are not fooled by pre-fetch queues or caches.

Eliminates the need to dedicate a target Serial port for debug.

All Host/Kernel communications are conducted through a Memory-Mapped UART functionality within the *Uni*ROM called URCOM. This functionality eliminates the need for a dedicated UART on the target for DEBUG, or frees up an existing UART for the target application's use. URCOM can be located anywhere within the emulated memory space.

Eliminates the need for extra RAM to hold the Application.

Since *Uni*ROM allows Paradigm DEBUG full Read/Write access to the EPROM space, the Application can reside in EPROM space, even during debug. This eliminates the need to install extra target RAM to load the Application for debugging.

Eliminates the need for a separate EPROM for the Kernel.

*Uni*ROM can download the Kernel into EPROM space before starting Paradigm DEBUG. The Application can be loaded into EPROM space before or after starting Paradigm DEBUG. The ability to load BOTH into the same EPROM space eliminates the need for a separate BOOT ROM for the Kernel.

Applications run "IN-PLACE" for less intrusive debugging.

The Application can be debugged in the exact same memory space it will occupy in its released form. In addition, the target RAM memory map will also match its released form. This brings the system one step closer to its release configuration.

Fast download of applications and the remote Kernel.

The Application and the Kernel can be downloaded with URTerm or URLoad before starting Paradigm DEBUG at greater than 400 KBaud, independent of the Target speed. Additionally, the Application may be downloaded from within Paradigm DEBUG at 115Kbaud.

*NOTE: URLoad.exe can be used to download applications that are located within the memory space emulated by *Uni*ROM. All Applications, regardless of location, can be downloaded from within Paradigm DEBUG as usual.

*Uni*ROM interfaces to the target through the EPROM or FLASH socket. This eliminates any dependencies on CPU speed, package style, pin-out, internal peripherals, MIN/MAX modes or other manufacture's variants. This independence allows *Uni*ROM to work with ALL of your Intel x86 and NEC V-Series target systems.

Overview

These instructions are intended to SUPPLEMENT the user's manuals that come with Paradigm DEBUG and *Uni*ROM, NOT REPLACE THEM. The following sections assume that you have read and are familiar with those manuals. Their contents are not duplicated here.

REQUIREMENTS:

A complete development systems consists of the following:

- *Uni*ROM
- URCOM option board for *Uni*ROM
- Paradigm Support diskette and instructions
- Paradigm Debug RT Version 4.0 or later
- Paradigm LOCATE Version 5.0 or later

The following steps were organized to isolate and detect problems as they occur, making it far easier to troubleshoot. Each step assumes the previous step was completed successfully, so do NOT skip any steps that fail. Stop and correct problems as they are uncovered. If you get stuck on a step, call for tech support. If each step is completed in sequence, you should have a full functioning setup by the time you start up Paradigm DEBUG!

We recommend that the system be brought up incrementally as follows:

Installation and Configuration

Installing and Testing *Uni*ROM

Refer to Chapters 1 and 2 of the *Uni*ROM User's Manual for complete instructions on installing and testing *Uni*ROM.

Do NOT skip connecting the RESET line. It allows *Uni*ROM to hold the target in reset during trasfers to insure consistent loads and target start-up.

Installing Paradigm Debug

Install Paradigm Debug, PDRemote and Locate as instructed in their manuals, using the build that most closely matches your target configuration.

Installing the Support Files

Copy the files from the TechTools "*Uni*ROM support for Paradigm Debug" diskette into the PDREM sub-directory. This will REPLACE your existing DCOMMS.C file with a *Uni*ROM specific version.

Configuration

Configure PDRT

Configure Paradigm DEBUG to operate at the 115200 BAUD. This is accomplished by editing the PDRT186.INI file as follows.

```
DEVICE = COM2
SPEED = 115200
```

Configuring PDREM

Edit the URCOM_BASE define at the top of the DCOMMS.C file to reflect the address segment you selected for the URCOM port. This can be any unused address segment that lands within *Uni*ROM's emulation space. The default location is 0xFFFC. NOTE that this is a SEGMENT definition. The actual linear target address is 16 times this value. The default setting of 0xFFFC places the URCOM at 0xFFFC0 in the target's memory space; 64 bytes below the top of memory.

If you are emulating a 16 or 32 bit EPROM DATA PATH, change the SCALE definition to 2 or 4 respectively. Note that this refers to the data path to the EPROMs, which is not necessarily as wide as the PROCESSOR's data path. Many targets will use 32 bit processors with 8 or 16 bit data paths to the EPROMs.

Edit the PDREM.CFG file to generate a BINARY file. Some of the PDREM.CFG files are already configured for BINARY files. Others are configured to generate extended INTEL HEX. To place PDREM at the top of memory, use:

"hexfile binary offset=0xFD000 size=12".

Edit any other configuration items in PDREM.CFG to match your system. For example, if your target uses an 80186 variant, you may need to configure

waitstates or some of the chip select registers to enable the EPROM, SRAM or other peripherals.

MAKE your PDREM kernel as instructed in the PDREMOTE section of the Paradigm DEBUG manual.

Configuring a Script File

You will find 2 sample script files in your PDREM directory called “pdrem8.cfg” and “pdrem16.cfg”. If your target uses a single 8 bit EPROM for its code, copy “pdrem8.cfg” to “UNIROM.CFG”. If your target uses a single 16bit EPROM or 2 8 bit EPROMS, copy “pdrem16.cfg” to “UNIROM.CFG”.

Edit “UNIROM.CFG” to reflect your configuration. The following sections discuss each item that may need to be modified. The *Uni*ROM User’s Manual documents each command and configuration item in detail.

Port Selection

This should be set to the port you intend to use for Paradigm Debug communications.

Valid PORT settings are COM1, COM2 COM3 or COM4.

Device SIZE and Type

The SIZE parameter is the size of the memory being emulated by THIS EMULATION BOARD in KBYTEs. Notice that a 27512 is a 512Kbit device or 64KBytes total. Also note that 16 bit EPROMs can be confusing. A 27240 is a 4Mbit device, organized as 256Kx16. When emulating 16bit devices, EACH emulation board (lower and upper) are emulating 1/2 of the device. Therefore we would enter “256” for the SIZE parameter in each board definition.

The type parameter simply indicates the type of device *Uni*ROM is emulating. If your target sockets are wired for FLASH devices, select FLASH, otherwise select EPROM

The ‘F0000’ or ‘E0000’ in this command is the ADDRESS parameter. This should be the REAL PHYSICAL ADDRESS of the start of emulation memory.

Reset

DON’T overlook this one. When debugging a program (or anything else) it is very important that your tools operate consistently. Configuring and connecting the reset line helps to insure that consistency.

Fill

We use this command to initialize the RAM underlying the URCOM board. This insures that the target will see that the URCOM has not been initialized yet and will NOT make false decisions based on the data in the RAM at the URCOM’s location. This initialization capability eliminates the need to hard-code it within the monitor as “define constant” statements. It also allows us to release reset and wait for target activity BEFORE initializing the CONSOLE connection (and

therefore URCOM). The target monitor will hit the initialization code long before URCOM is initialized. This insures a consistent start by forcing the target to wait for URCOM initialization.

The first parameter is URCOM's address. This should be set to URCOM's actual address (from the target's view). The second parameter should be set to the last address occupied by URCOM. URCOM occupies 16 bytes in the LOWER emulation board. If your target uses two or more emulation boards in an interleaved configuration, URCOM will space 32 or 64 bytes, respectively. This is the LAST byte to fill, for a total of 16, 32 or 64 bytes, depending on the address scaling. The third parameter is the fill value. Leave it set to 01.

Console

The CONSOLE line specifies that we should initiate a CONSOLE connection with the URCOM option board. The second parameter specifies URCOM's ADDRESS. It should be the same as the first parameter in the FILL command. The "RX" specifies that we should generate interrupts when we place data in the target's RECEIVE register. We will not be servicing interrupts on the target side yet, but it does no harm to go ahead and let *Uni*ROM generate them. The final parameter is the CONSOLE connection BAUD rate. This MUST match the baud rate the debugger will use to communicate with the target.

Loading the Kernel into *Uni*ROM

Type "URLOAD" to load the kernel into *Uni*ROM. Urload looks in the current directory for the default script file named "UNIROM.CFG".

Verifing Basic Kernel operation

Watch for Target activity on the target status LED. If the kernel is alive and polling for commands, the target status LED should show continuous activity. If it does NOT show activity, then *Uni*ROM is not being accessed. If there is no target activity, check the following:

Verify all *Uni*ROM settings, in particular, the device type, size and length.

Start URTERM and use the READ command to verify that the Kernel LOOKS OK. For instance, if we are emulating the boot ROM, look for a long jump (EA xxxx xxxx) at 0xFFFF0. If we are emulating a ROM-SCAN device, look for the characteristic 55AA at the start of ROM. If you can not identify the kernel, review the file type, offset and upload procedures. Also review the steps taken to build the kernel. Verify that PDREM.CFG is configured so that the kernel is LOCATED to execute in *Uni*ROM's memory space.

If the kernel LOOKS OK but *Uni*ROM still shows no target activity, then verify the kernel configuration. Verify that PDREM.CFG accurately reflects your target's memory map. In particular, verify that the Kernel's Data and Stack are mapped to real RAM addresses. Also verify all initcode directives in the PDREM.CFG file to insure that you are properly configuring all chip selects, wait-states , etc.

Verify that URCOM is placed in un-used space. Look for the "01" bytes at the selected URCOM address. The default script placed them at 0xFFFC0. If your initcode directives produce a lot of code, that code may bleed into this address. If you do not see several FF pad characters before the URCOM address, move the ini-code section lower in memory to give it more room. Edit the ??LOCATE in PDREM.CFG file to the following:

```
class   ??LOCATE = 0xffe0
```

This will allow an additonal 256 bytes of space for the initialization code.

Starting Paradigm DEBUG

If all goes well, you should see the Paradigm DEBUG sign-on message. If not, you may get a link time-out or the PC may hang. If Paradigm DEBUG does not start up, verify that baud rate and com port selections in the PDRT186.INI file and the PORT definition line in UNIROM.CFG match.

Always RESET *Uni*ROM and reload the kernel after after any type of Paradigm DEBUG failure, to eliminate any communications problems and to insure a consistent start.

Modify for interrupt mode

The following steps assume that a working POLLED MODE driver has been built and debugged, allowing us to concentrate only on the changes necessary for INTERRUPT operation.

Configure *Uni*ROM for Interrupt mode operation.

Edit the UNIROM.CFG file to set the INTERRUPT active level and driver type. This is the only change to *Uni*ROM's configuration required to support interrupt mode.

Connect Interrupt line

To run the Kernel in Interrupt mode, you will need to connect the *Uni*ROM IRQ line (pin 7) to the target's interrupt request input. We suggest using the NMI interrupt if possible.

Rebuild the Kernel for Interrupt Mode operation

Edit TARGET.H to indicate the Target interrupt number to use and to enable interrupt operation.

If necessary, edit DCOMMS.C to reflect your Target's interrupt structure. It defaults to a PC type environment with an 8259 interrupt controller located at 0x20.

If you are using a processor with an internal interrupt controller (like the 80c186EC), you may also need to modify some of the initcodes in the PDREM.CFG.

Refer to the PDREM manual for additional information.

Build the Kernel (PDREM) as instructed in the PDREM manual.

Load the new Kernel

Use URLOAD to load the new Kernel to *Uni*ROM as before.

Verify basic Kernel operation

Reset *Uni*ROM and watch for target activity on the status LED.

Start Paradigm DEBUG

If all goes well, you should see the Paradigm DEBUG sign-on message. If not, you may get a link time-out or the PC may hang. If Paradigm DEBUG does not start up, RESET *Uni*ROM and try again. If problems persist, review the interrupt controller setup, interrupt polarity and drive type configurations and the interrupt connection point on the target. Try returning to a polled mode driver to establish a base-line.

Normal Operation

Once you have a working setup, *Uni*ROM is VERY easy to use. In normal day-to-day debugging, follow these steps:

1. Apply power to *Uni*ROM
2. Apply power to the Host system.
3. Run "URLOAD kernel.ini" to upload the Kernel to *Uni*ROM.
4. Apply power to the target system.
5. Start Paradigm DEBUG.
6. Start debugging!

General NOTES:

*Uni*ROM should always be powered up before the target and powered down after the target.

You should always QUIT Paradigm Debug BEFORE turning off the target.

Troubleshooting

Most problems should be diagnosed and corrected during the configuration steps described above. However, the following problems could be encountered after basic operation is established.

Paradigm DEBUG is showing that the Target is RUNNING, but you can not stop it.

The Kernel can be configured for Polled mode or Interrupt driven operation. When configured for Polled mode, the is no way to interrupt a run away target program from within Paradigm DEBUG. You may RESET the target to regain control or re-configure the Kernel to operate in interrupt mode. You may also connect an external switch to your NON-MASKABLE Interrupt line.

Paradigm DEBUG tries to load, but just responds with "....." or a "Link Time-out" message.

If you never have had the system working, follow through the sequential steps outlined above. This should pinpoint the problem. If this finds a problem you do not know how to solve, call us for suggestions.

If you HAD a working setup at one time and this problem just started, try pressing the RESET switch on *Uni*ROM and then re-start Paradigm DEBUG. Also verify the serial port connections. If the problem persists, re-trace the steps outlined above to pinpoint the cause of the problem.

Paradigm loads, but locks-up intermittently

Any communications failure between Paradigm Debug and the PDREM kernel will cause the PC to hang. Since the target is managing the communications, the PC may hang if the target dies or stops responding to interrupts. This can be particularly annoying during the initial bring-up. Running Paradigm Debug under Windows helps because it will often allow you to kill the program when this happens. You may also find that adding “FLAGS = -f “ to the PDRT186.INI file helps.

If the system works well in POLLED mode, but LOCKS-UP in interrupt mode, the problem may be due to improper Interrupt polarity or driver type configuration, or an inproper target connection point.

Noise on the interrupt line can also cause this problem by generating false interrupts to the interrupt controller. The NMI interrupt is usually more forgiving to this than others. You may also try using a SPECIFIC END OF INTERRUPT instead of the NON-SPECIFIC type. This is usually more immune to false triggers.

You should also verify that no other signal is driving this same interrupt line, causing a conflict. Of course you should also verify that the interrupt controller is configuration properly and that the end of the interrupt routine cleans-up properly.

The target may not be capable of keeping up at 115200 BAUD. This is not a very likely problem because *Uni*ROM provides large FIFOs that are capable of buffering entire packets. However, it is possible. To change to a different baud rate, three items need to be modified. Set the BAUD= line in "kernel.ini" to the new BAUD rate. Also change the default BAUD rate in the Setup/Baud menu within *Uni*ROM (and STORE it). Finally, change the BAUD rate in PDRT186.ini to match.

If you are running under Windows, be sure to use the PIF provided by Paradigm. In extreme cases, you may need to reduce the BAUD rate or close down other applications that are competing for CPU resources. You may also try adding "pace 40" to the "[CFG]" section of the script file.

Trouble loading Applications into UniROM space

If you can load and debug applications in RAM, but get errors when trying to load applications into *Uni*ROM space, the target may be having problems writing to *Uni*ROM's memory space. Verify that the target CAN write to *Uni*ROM. If *Uni*ROM is plugged into an EPROM socket, you need to connect the target's write signal to *Uni*ROM's external write input. If *Uni*ROM is plugged into a FLASH socket, verify that *Uni*ROM is configured for FLASH operation.

In order to write to *Uni*ROM, the target must:

- Enable the chip select during the write operation.
- Provide a BI-DIRECTIONAL data path to the CPU
- Provide a write signal to the socket (or external input)

Some targets will NOT provide this functionality, because the board designer did not envision anybody wanting to write to the code space.

You may choose to write a simple test routine that writes a message into some unused memory location within the space emulated by *Uni*ROM. You could then use URTERM or Paradigm DEBUG to view those locations to determine if the write succeeded.

If your target uses a 16 or 32 bit data path to its EPROMs, you may encounter another problem when attempting to write to this space. Many targets are NOT CAPABLE of doing BYTE-WIDE WRITES to the EPROM/FLASH sockets. Often a single chip select line is used to activate ALL EPROMs in the code bank. This means that ANY WRITE to ONE of the EPROMS would result in writing garbage into the remaining EPROMs. The ideal solution would be modify the target's chip selects to qualify the EPROM chip selects with A0 (and possibly A1) to allow individual byte selection capability.

There is another work-around. TechTools has modified one of Paradigm's modules (HELPERS.C) to force WORD WIDE accesses to the code space for all code loading and breakpoint patching. At the time of this writing, this was experimental. Please contact TechTools for information on its availability and use.

If you need additional assistance, please call or FAX:

TechTools

PO BOX 462101
Garland TX 75046-2101

Voice: (972) 272-9392 FAX: (972) 494-5814 email: support@tech-tools.com