Embedded Systems Development Tools

PO Box 462101 Garland, TX 75046-2101

Voice: (972) 272-9392 FAX: (972) 494-5814
EMAIL: SALES@TECH-TOOLS.COM

*Application NOTE: Uni*ROM, an over-view

Let me start with an over-view of exactly what *Uni*ROM does and how it is used. *Uni*ROM can operate in a stand-alone configuration, or can be integrated into any Software Monitor or Debugger which uses a remote kernel and serial based communications. I will cover each configuration below.

STAND-ALONE OPERATION

In stand-alone operation, *Uni*ROM provides all of the features listed under "*Uni*ROM features" and "Standard Debug Support" in the brochures. To summarize, *Uni*ROM provides high speed memory emulation, dual port access to emulation memory, 8 watch points, Virtual UARTs and Virtual Console modes.

*Uni*ROM has built-in intelligence which provides a full menu based interaction with the user through either the parallel interface or the serial port interface. The user may use any terminal emulator program on any host with a standard serial port to communicate with *Uni*ROM. In addition, we provide a batchable loader for very quick, fully automated uploads and a DOS based terminal emulator. Both of these programs will work through the serial or parallel ports on PC hosts.

In its most basic mode of operation, *Uni*ROM provides very advanced memory emulation with software configured EPROM and FLASH pinouts, full target signal termination, high speed uploads, independent power supply, target power sense, 4 user control outputs, configurable (Active high/low, tri-state/bipolar)target reset line, configurable (Active high/low, tri-state/bipolar) target interrupt line, 4 status inputs and built-in self-test. These features alone make *Uni*ROM a very high-end memory emulator at a very attractive price.

In addition to the above mentioned features, *Uni*ROM comes standard with a full dual-port memory architecture, Virtual UARTs and Virtual Consoles. The dual-port memory architecture allows the user to view and edit emulation memory while the target is running, without inserting wait-states or depending on the target for anything. This opens the door to many debugging techniques. Some of our customers are using this to "SPY" on the target's operations. Some are placing key variables (loop counters,message queues, pointers, etc.) in CODE space so that they can be viewed by *Uni*ROM without interrupting the target. Others are having the target write messages or critical information (active task ID, ring-buffer over/under flow flags, iteration counts, pointer values (looking for NULLs) etc.) into CODE space. In addition to "SPYING" on the target, some customers are WRITING commands into memory while the target is running. The target then periodically looks at the chosen locations for commands and acts on them. Other customers use this feature to force error conditions or to simulate events. For instance, one could fill a ring buffer with a string or packet of information, then adjust the ring-buffer pointers to simulate the receipt of a message. This would

allow them to test the parse routines without actually having to generate the packets externally.

The Virtual UARTS are a memory-mapped interface which resemble standard hardware UARTs. This formalizes bi-direction communications between the target and host through a common mechanism. The Virtual UARTs can be placed on any 4 byte boundary within the memory space being emulated by *Uni*ROM.

The Virtual CONSOLE path is a mechanism which redirects the *Uni*ROM serial port to the Virtual UARTS, making *Uni*ROM look invisible to those communications. One can switch between CONSOLE mode and COMMAND mode at will. This can be used to re-direct a target program's serial I/O to the terminal emulator being used to control *Uni*ROM, eliminating the need to jump back and forth between *Uni*ROM control and target interaction. In addition, this functionality can be used to ADD a communications path to a target for passing debugging information, trace messages, commands or status between the target and HOST. We have some customers that write debug information to the *Uni*ROM Virtual UART during product development AND in the released firmware. This allows the firmware to follow the EXACT same code paths during normal operation as during debug. When a problem arises, an Engineer can plug in a *Uni*ROM and watch the debug/trace messages to gain insight into the problem without AFFECTING the system.

THIRD-PARTY DEBUGGING SUPPORT

The Virtual UARTs, Virtual CONSOLE paths and on board intelligence allow one to easily integrate *Uni*ROM into any software debugger that uses a serial port and a remote kernel or monitor for operation. Software debuggers depend on a small Kernel of code on the target to help it gain access to the target. This kernel is provided by the debugger manufacturer and is designed to be as small and unobtrusive as possible. Most kernels understand only about a dozen simple binary command packets (read/write memory, read/ registers, read/write I/O, single step, etc.). The user interface, symbol databases and breakpoint tables reside on the HOST side of the link. The HOST comunicates with the kernel through a serial link.

In this role, *Uni*ROM enhances software debuggers by eliminating some of thier weaknesses. These 3rd party debuggers are intrusive to the target and require a varying amount of target resources. *Uni*ROM helps to minimize those intrusions, to make the debugger look more like a hardware in-circuit emulator. In effect, *Uni*ROM provides the missing hardware support.

*Uni*ROM eliminates the need for a target serial port for debugging. If the user's target has a serial port, it most likely has a purpose (other than debugging). It may not be possible to use this for debugging and still be able to operate the target. *Uni*ROM's Virtual UARTs allows debuggers to communicate through *Uni*ROM, leaving the target's serial port for its intended use or eliminating the need to add a serial port for debugging.

The second most intrusive aspect of software debuggers is the need for additional SRAM. A software debugger typically sets break-points by writing trap, break-point or jump instructions into the CODE. Obviously, it can not do this if the code is in ROM. In a typical setup, a software debugger will require that the user downloads the target application into RAM so that it can write these breakpoints. Single stepping is accomplished through the same mechanism in some architectures. *Uni*ROM eliminates this requirement by allowing the application to reside within *Uni*ROM, and allowing the target to WRITE to *Uni*ROM. This eliminates the need for additional SRAM for debugging, and allows the application to run "IN-PLACE" for a more accurate emulation.

*Uni*ROM eliminates these limitations for any software debugger without special drivers or custom versions of the debugger. All kernel based debuggers allow the user to configure the target kernel communications routines to match the target's serial port and interrupt structure. When using *Uni*ROM with one of these systems, the user simply configures the kernel to talk to *Uni*ROM's Virtual UART. Since these UARTs look very much like a standard hardware UART, the modifications are very straight-forward. This one simple modification to the kernel side communications routines is all that is needed to provide this level of support to any software debugger that uses serial port communications and a remote kernel. We call this LEVEL 1 support.

*Uni*ROM has additional features which allow us to provide higher levels of support to some debuggers. Downloadable Protocol Extensions allow us to customize *Uni*ROM to a specific debugger to unlock additional features that are normally available only to an ICE. These features include DYNAMIC mode operation, real-time watch windows and trace board support. These protocol extensions allow us to add this support to industry standard debuggers WITHOUT REQUIRING CUSTOM VERSIONS of the debugger. They work in conjunction with standard, off-the-self software debuggers, which can be purchased directly from the manufacturer or from us. We will be releasing custom extensions for popular debuggers soon.

SPECIAL 8051 NOTES:

The 8031 family is NOT CAPABLE of writing to its CODE space. Many Engineers will over-lay the code and data spaces to allow for debugging. If the user's target is designed with this feature, everything we have said so far applies to them. If, however, the target has separate DATA and CODE spaces, the user will not be able to WRITE to the CODE space. This would preclude any communications FROM the target TO the HOST. We offer an option board for *Uni*ROM called URCOM. This option plugs into expansion connectors within *Uni*ROM and provides a very special Virtual UART interface to the target. URCOM allows full bi-direction communications between the target and host WITHOUT any TARGET WRITES. The target receives data from the HOST through a standard RXBUFFER and RXFULL FLAG type of interface. However, the target can TRANSMIT to the HOST by doing a special 3 READ sequence. Hardware within URCOM senses this sequence and interprets it as a WRITE to the TX BUFFER. URCOM fills the TX BUFFER for the target and sets the TXFULL flag.

The URCOM option allows a HARVARD architecture target (like the 8031 without code overlays) to communicate through shared CODE space. However, it does NOT permit the debugger to patch CODE (for breakpoints). We will eliminate this final restriction in the near future with a protocol extension which will take advantage of *Uni*ROM's dual-port architecture to do those patches from *Uni*ROM's side of the memory, rather than the target's side. Until then, users will need to over-lay their CODE and DATA space to be able to set breakpoints on an 8031 type board.

CONCLUSION

As you can see, *Uni*ROM is very flexible and provides several different levels of support. You can start with a standard *Uni*ROM and do a considerable amount of debugging. We have many customers that use *Uni*ROM in a stand-alone configuration. In addition, you can combine *Uni*ROM with free or very low cost monitor type programs for assembly level debugging. If your needs (or budgets) grow, you can add in third-party source level debuggers later. Additionally, *Uni*ROM will work with ALL of your targets and CPUs, providing a common tool that will grow with you.