



## Mini Debugger



**Bruno Paillard  
et  
Alex Boudreau**

July 27, 2001  
Rev 2: October 12, 2002



1	INTRODUCTION	4
2	USER INTERFACE	5

## 1 Introduction

The mini debugger described in this document allows the developer to interactively:

- Reset the Signal Ranger board.
- Download a DSP COFF executable file to DSP memory.
- Launch the execution of code from a specified address.
- Read and write CPU registers.
- Read and write DSP memory.
- Clear, program and verify the Flash Boot ROM in the SP2 version of Signal Ranger

This mini debugger can be used to explore the DSP's features, or to test DSP code during development.

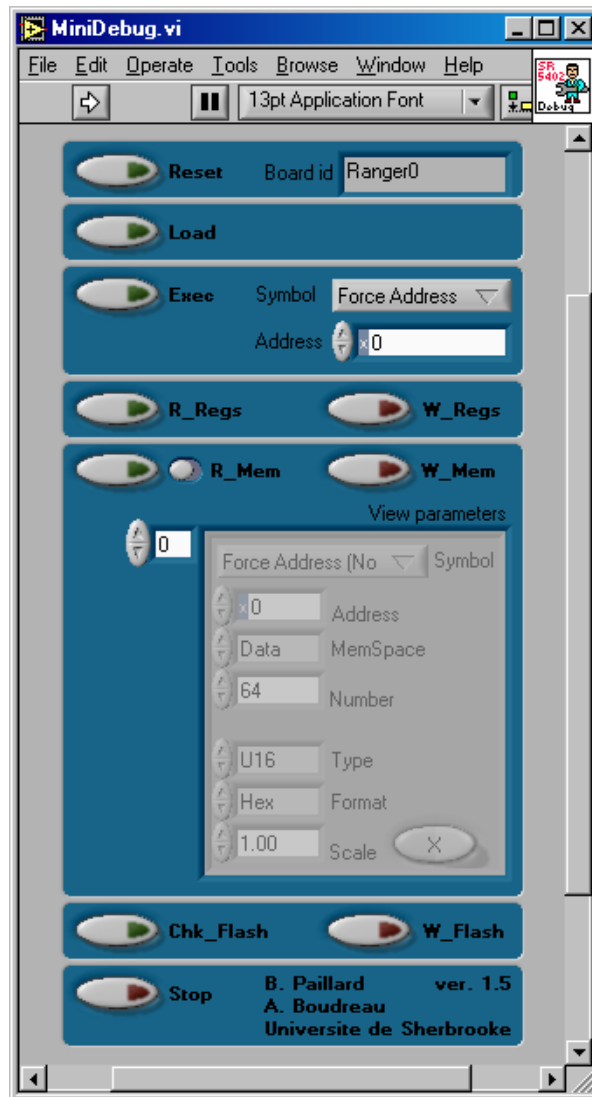


Figure 1 - Front panel of the mini debugger

## 2 User interface

- **Board id** Selects which Signal Ranger board to use for debugging. If only one is connected to the USB port, "Ranger0" should be used. If not, change the index to match the desired board. Boards are assigned a number from 0 to N at connexion time, in the order of connexion.
- **Reset** Resets the board and reloads the kernel. All previously executing DSP code is aborted.
- **Load** Loads a DSP COFF file. The application presents a file browser to allow the file selection. The file must be a legitimate C5402 COFF1 or COFF2 file.

**Note** It is recommended to always reset the board before proceeding to a load. Otherwise, the execution of the user DSP code may interfere with the load operation. The board is not systematically reset before a load because it may be useful to load a data section while the user code is running.

**Note** Contrary to the previous versions of the Mini-debugger, version 1.5 does not systematically reset the DSP when it is run. This modification has been made to allow the mini-debugger to communicate and gain control of a DSP code that has loaded from Flash ROM at power-up with Signal Ranger SP2. The reset button can still be used to manually reset the DSP at any time.

- **Exec** Forces execution to branch to the specified label or address. The DSP code that is activated this way should contain an *Acknowledge* in the form of a *Host Interrupt Request (HINT)*. Otherwise the USB controller will time-out, and an error will be detected by the PC after 5s. The simplest way to do this is to include the *acknowledge* macro at the beginning of the selected code. This macro is described in the two demo applications.
  - **Symbol** or **Address** is used to specify the entry point of the DSP code to execute. **Symbol** can be used if a COFF file containing the symbol has been loaded previously. Otherwise, **Address** allows the specification of an absolute branch address. **Address** is used only if **Symbol** is set to the "Force Address (No Symbol)" position.  
When a new COFF file is loaded, the mini-debugger tries to find the **\_c\_int00** symbol in the symbol table. If it is found, and its value is valid (different from 0) **Symbol** points to its address. If it is not found, **Symbol** is set to the "Force Address (No Symbol)" position.
- **R\_Regs** Reads the CPU registers mapped in RAM at address 0000<sub>H</sub>, and presents the data in an easy to read format.

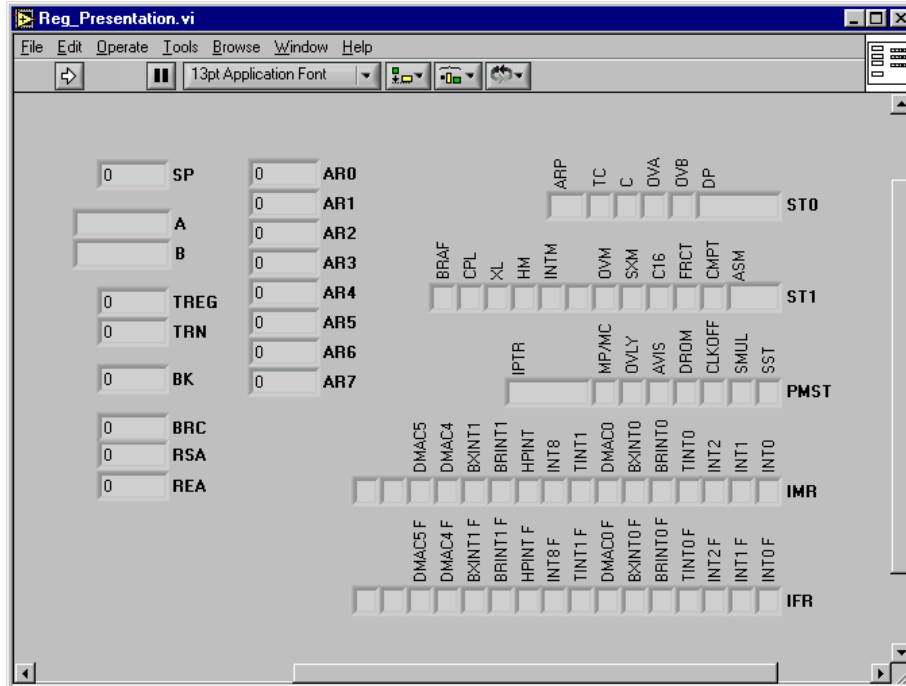


Figure 2 - Register presentation panel

- W\_regs** Allows to write most of the CPU registers mapped in RAM at address 0000<sub>H</sub>. Some fields are greyed out and cannot be written, either because the kernel uses them and would restore them after modification (ST0, AR1, AR2, AR3), or because their modification would compromise its operation (SP, BRAF, INTM, IPTR, OVLY, HPINT).

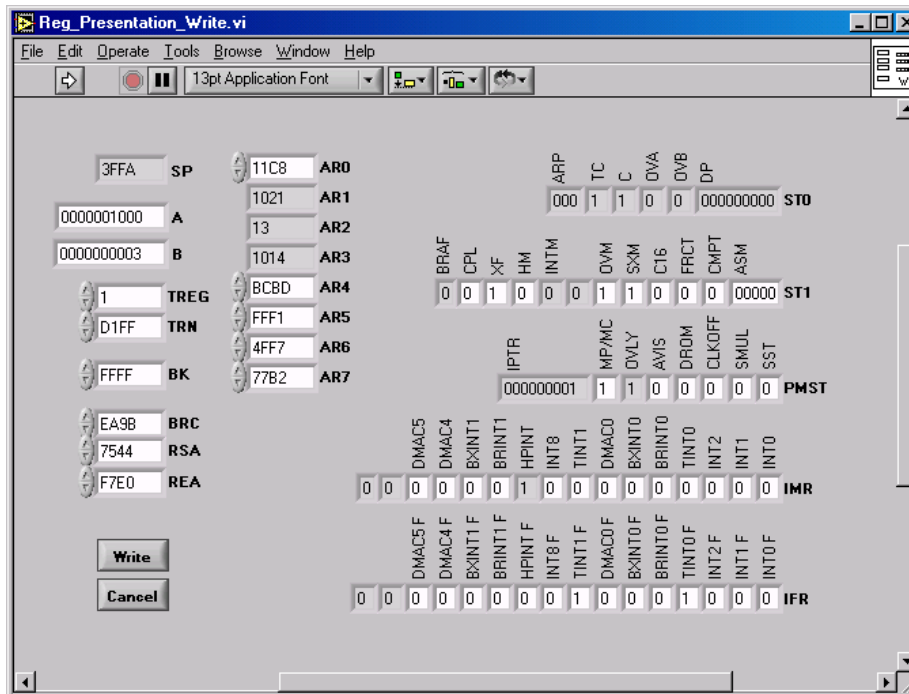


Figure 3 - Register write presentation panel

- **R\_Mem** Reads DSP memory and presents the data to the user. The small slide beside the button allows a continuous read. To stop the continuous read, simply replace the slide to its default position. The *View parameter* array is used to select one or several memory blocks to display. Each index of the array selects a different memory block. To add a new memory block, simply advance the index to the next value, and adjust the parameters for the block to display. To completely empty the array, right-click on the index and choose the “Empty Array” menu. To insert or remove a block in the array, advance the index to the correct position, right-click on the *Symbol* field, and choose the “Insert Item Before” or “Delete Item” menu.

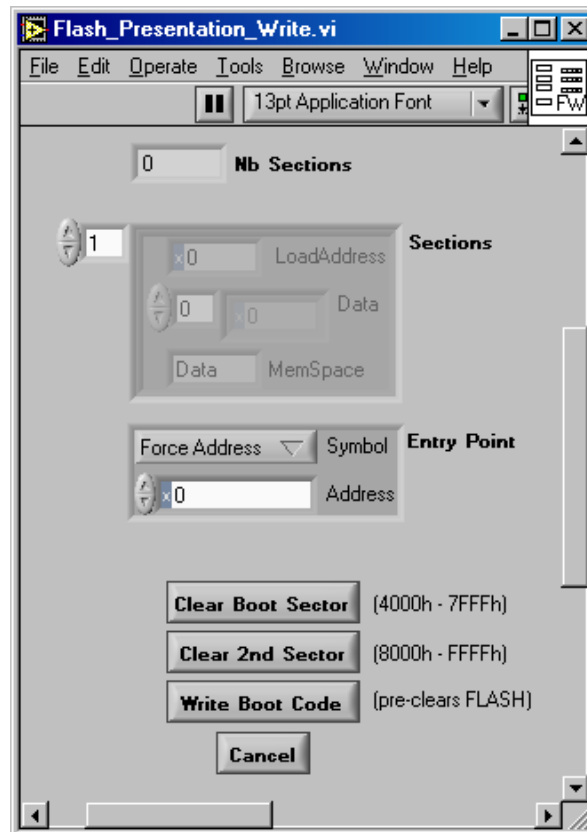
For each block:

- **Symbol** or **Address** is used to specify the beginning of the memory block to display. **Symbol** can be used if a COFF file containing the symbol has been loaded previously. If **Symbol** is set to a position other than “Force Address (No Symbol)”, **Address** is forced to the value specified in the COFF file for this symbol.  
The list of symbols is cleared when a new COFF file is loaded, or when the Mini-Debugger is stopped and run again. It is not cleared when the DSP board is reset. By right-clicking on the Symbol field, it is possible to remove or insert an individual element.
- **MemSpace** indicates the memory space used for the access. The position “???” (Unknown) defaults to an access in the Data space. If **Symbol** is set to a position other than “Force Address (No Symbol)”, **MemSpace** is forced to the value specified in the COFF file for this symbol.
- **Number** specifies the number of bytes or words to display.
- **Type** specifies the data type to display. Three basic widths can be used: 8 bits, 16 bits, and 32 bits. All widths can be interpreted as signed (*I8, I16, I32*), unsigned (*U8, U16, U32*), or floating point data. The native DSP representation is 16 bits wide. When presenting 8-bit data, the bytes represent the high and low parts of 16-bit memory registers. They are presented MSB first and LSB next. When presenting 32-bit data (*I32, U32* or *Float*), the beginning address is automatically aligned to the next even address. The upper 16 bits are taken as the first (even) address and the lower 16 bits are taken as the next (odd) address. This follows the standard bit ordering for 32-bit data, as explained in Texas Instruments document *SPRU131f “TMS320C54x DSP reference set – Volume 1 CPU and peripherals”*.
- **Format** specifies the data presentation format (Hexadecimal, Decimal or Binary).
- **Scale** specifies a scaling factor for the graph representation.
- **X or 1/X** specifies if the data is to be multiplied or divided by the scaling factor.



addresses are indicated in the first column. In *Graph* mode, each memory block is scaled, and represented by a separate line of a graph.

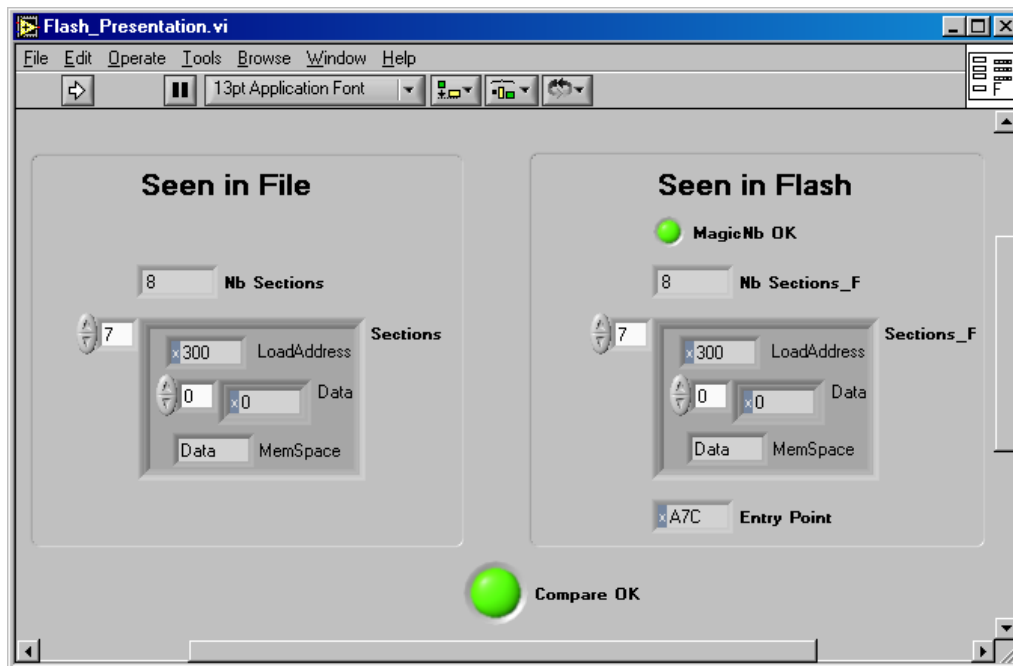
- **W\_Mem** Allows the memory contents to be read and modified. The function first reads the memory, using the *View\_parameters*, and presents a Text panel similar to the one presented for the *R\_mem* function. The user can then modify any value in the panel, and press the *Write* button to write the data back to memory. Several points should be observed:
  - Even though data entry is permitted in any of the cells of the panel, only those cells that were displayed during the read phase (those that are not empty) are considered during the write.
  - When writing 8-bit data, one should only attempt to write an even number of bytes. This is because physical accesses occur on 16-bits at a time. If a write is attempted with an odd number of bytes, the last one (which is not paired) will not be written.
  - Data must be entered using the same type and format as were used during the read phase.
- **W\_Flash** Allows the erasure or programming of the Flash Boot ROM in the SP2 version of Signal Ranger. This button brings a browsing window to the foreground. The user is asked to specify the executable (.out) file that should be programmed into Flash. After this operation the following interface is brought to the foreground.



**Figure 6**

The operation systematically resets the DSP to avoid any interference from an already executing DSP code.

- **Nb Sections** gives the number of sections to be loaded into Flash ROM. Note that empty sections in the .out executable file are eliminated. Note that sections to be loaded in the program space above address 4000<sub>H</sub> are also eliminated after a warning message.
  - **Sections** Lists the Load Address, Memory space and words of data and code to be loaded, section-by-section.
  - **Entry Point** Specifies the entry point of the code. If the symbol **\_c\_int00** is found in the .out file, the selector is positioned there at first. The user may choose another symbol from the drop-down list, or may force an hexa-decimal address. The address specified is only taken if the drop-down list points to the "Force Address" entry.
  - **Clear Boot Sector** This button clears the first sector of the Flash (including the magic number at address 4000<sub>H</sub>. This is all that is required to prevent the DSP from booting from Flash ROM at power-up.
  - **Clear Second Sector** This button clears the second sector of Flash.
  - **Write Boot Code** This button processes the .out executable file and programs the Flash so that the DSP boots from it at power-up. The required sectors of Flash are erased as required prior to the programming.
- **Chk\_Flash** Verifies the programming of the Flash. This button brings a browsing window to the foreground. The user is asked to specify the executable (.out) file that should be checked against the Flash. After this operation the following interface is brought to the foreground.



**Figure 7**

The operation systematically resets the DSP to avoid any interference from an already executing DSP code.

- **Stop** Stops the execution of the mini debugger.

**Note:** When presenting, or writing 32 bit data words (I32, U32 or Float), the PC performs 2 separate accesses (at 2 successive memory addresses) for every transferred 32-bit word. In principle, the potential exists for the DSP or the PC to access one word in the middle of the exchange, thereby corrupting the data.

For instance, during a read, the PC could upload a floating point value just after the DSP has updated one 16-bit word constituting the float, but before it has updated the other one. Obviously the value read by the PC would be completely erroneous. Symetrically, during a write, the PC could modify both 16-bit words constituting a float in DSP memory, just after the DSP has read the first one, but before it has read the second one. In this situation The DSP is working with an "old" version of half the float, and a new version of the other half. These problems can be avoided if the following precautions are observed:

When the PC accesses a group of values in the data or program space (this is not true for the I/O space), it does so in blocks of up to 32 16-bit words at a time. Each of these 32-word block accesses is atomic (the DSP cannot do any operation in the middle of the PC access). Therefore the DSP cannot "interfere" in the middle of any single 32-bit word access.

This alone does not guarantee the integrity of the transferred values, because the PC can still transfer a complete block of data in the middle of a DSP operation on this data. To avoid this situation, it is sufficient to also render the DSP operation on any 32-bit data atomic (by disabling interrupts for the length of the operation), then the accesses are atomic on both sides, and data can safely be transferred 32 bits at a time.