# LEON DSU Monitor User's Manual

Version 1.0
February 2002

**Jiri Gaisler**
**Gaisler Research**

Gaisler Research

jiri@gaisler.com

*LEON DSU Monitor User’s Manual*

# 1 Introduction

## 1.1 General

DSUMON is a debug monitor for the LEON processor debug support unit. It includes the following functions:

- Read/write access to all LEON registers and memory
- Built-in disassembler and trace buffer management
- Downloading and execution of LEON applications
- Breakpoint and watchpoint management
- Remote connection to GNU debugger (gdb)
- Auto-probing and initialisation of LEON peripherals and memory settings

## 1.2 Supported platforms and system requirements

DSUMON supports three platforms: solaris-2.8, linux-2.2/glibc-2.2 and windows98/NT/2K. The windows version requires that cygwin-1.3.9 or higher is installed. Cygwin can be downloaded from sources.redhat.com.

## 1.3 Obtaining DSUMON

The primary site for DSUMON is http://www.gaisler.com/, where the latest version of DSUMON can be ordered and evaluation versions downloaded.

## 1.4 Installation

DSUMON can be installed anywhere on the host computer - for convenience the installation directory should be added to the search path. The commercial versions use a license file which should be pointed to by the environment variable DSUMON_LICENSE_FILE, otherwise $HOME/dsumon.lic or ‘dsumon.lic’ in the current directory are used.

## 1.5 Problem reports

Please send problem reports or comments to dsumon@gaisler.com.

# 2 Operation

## 2.1 Overview

DSUMON can operate in two modes: standalone and attached to gdb. In standalone mode, LEON applications can be loaded and debugged using a command line interface. A number of commands are available to examine data, insert breakpoints and advance execution.

When attached to gdb, DSUMON acts as a remote gdb target, and applications are loaded and debugged through gdb (or a gdb front-end such as ddd).

## 2.2 Staring DSUMON

The LEON DSU uses a dedicated uart to communicate with an outside monitor. The uart uses automatic baud-rate detection. To succefully attach DSUMON, first attach the serial cable between the target board and the host system, then power-up and reset the target board, and finally start DSUMON using the -uart option in case the DSU is not connected to /dev/ttyS0 of your host (or /dev/ttya on solaris hosts). Note that the DSUEN signal on the LEON processor has to be asserted for the DSU to operate.

When DSUMON first connects to the target, a check is made to see if the system has been initialised with respect to memory, UART and timer settings. If no initialisation has been made (= debug mode entered directly after reset), the system first has to be initialised before any application can run. This is performed automatically by probing for available memory banks, and detecting the system frequency. The initialisation can also be forced by giving the -i switch at startup. The detected system settings are printed on the console:

```
jiri@venus $ ./dsumon -i

 LEON DSU Monitor, version 1.0
 Copyright (C) 2001, Gaisler Research - all rights reserved.
 Comments or bug-reports to jiri@gaisler.com

 port /dev/ttyS0 at 115200 baud

 clock frequency       :  24.9 MHz
 register windows      :  8
 instruction cache     :  4 kbytes, 16 bytes/line
 data cache            :  2 kbytes, 16 bytes/line
 hardware breakpoints  :  4
 trace buffer          :  128 lines
 ram width             :  32 bits
 ram banks             :  2
 ram bank size         :  2048 kbytes
 stack pointer         :  0x403ffff0
dsu>
```

## 2.3 Command line options

DSUMON is started as follows on a command line:

**dsumon** [*options*] [*input_files*]

The following command line options are supported by DSUMON:

**-abaud** *baudrate*

Use *baudrate* for UART 1 & 2. By default, 38400 baud is used.

**-banks** *ram_banks*

Overrides the auto-probed number of populated ram banks.

**-baud** *baudrate*

Use *baudrate* for the DSU serial link. By default, 115200 baud is used.

**-c** *file* Reads commands from *file* instead of stdin. If the file .dsumonrc exists in the home directory, it will be automatically be executed.

**-freq** *system_clock*

Overrides the detected system frequency. Use with care!

**-gdb** Listen for gdb connection directly at start-up.

**-i** Force a system probe and initialise LEON memory and peripherals settings.

**-port** *gdbport*

Set the port number for gdb communications. Default is 1235.

**-ram** *ram_size*

Overrides the auto-probed amount of ram.

**-romrws** *waitstates*

Set *waitstates* number of waitstates for rom reads.

**-romwws** *waitstates*

Set *waitstates* number of waitstates for rom writes.

**-romws** *waitstates*

Set *waitstates* number of waitstates for both rom reads and writes.

**-ramrws** *waitstates*

Set *waitstates* number of waitstates for ram reads.

**-ramwws** *waitstates*

Set *waitstates* number of waitstates for ram writes.

**-ramws** *waitstates*

Set *waitstates* number of waitstates for both ram reads and writes.

**-stack** *stackval*

Set *stackval* as stack pointer for applications, overriding the auto-detected value.

**-uart** *device*

By default, DSUMON communicates with the target using /dev/ttyS0 (/dev/ttya on solaris). This switch can be used to connect to the target using other devices. e.g '-uart /dev/cua0.

*input_files*

Executable files to be loaded into memory. The input file is loaded into the target memory according to the entry point for each segment. The only recognized format is elf32.

# 3 Standalone mode

## 3.1 Internal commands

DSUMON dynamically loads libreadline.so if available on your host system, and uses `readline()` to enter/edit monitor commands. If libreadline.so is not found, `fgets()` is used instead (no history and poor editing capabilities). Below is description of commands that are recognized by the monitor when used in standalone mode:

**ahb** [*length*] Print only the AHB trace buffer. The *length* last AHB transfers will be printed, default is 10.

**batch** *file* Execute a batch file of DSUMON commands.

**bre** [*address*] Adds a software breakpoint at *address*. If *address* is omitted, prints all breakpoints.

**del** [*num*] Deletes breakpoint *num*. If *num* is omitted, all breakpoints are deleted.

**cont** [*count*/time]

Continue execution at present position.

**dis** [*addr*] [*count*]

Disassemble [*count*] instructions at address [*addr*]. Default values for count is 16 and *addr* is the program counter address.

**echo** *string* Print <string> to the simulator window.

**float** Prints the FPU registers

**gdb** Listen for gdb connection.

**go** [*address*]

The **go** command will set pc to *address* and npc to *address* + 4, and resume execution. No other initialisation will be done. If address is not given, the default load address will be assumed.

**hbre** [*address*] Adds a hardware breakpoint at *address*. If *address* omitted, prints all breakpoints.

**help** Print a small help menu for the DSUMON commands.

**hist** [*length*] Print the trace buffer. The *length* last executed instructions or AHB transfers will be printed, default is 10.

**inst** [*length*] Print only the instruction trace buffer. The *length* last executed instructions will be printed, default is 10.

**init** Do a hardware probe to detect system settings and memory size, and initialize peripherals.

**load** *files* Load *files* into simulator memory.

**leon** Display LEON peripherals registers.

**mem** [*addr*] [*count*]

Display memory at *addr* for *count* bytes. Same default values as for **dis**.

**quit** Exits the monitor.

**reg** [*reg_name value*]

Prints and sets the IU registers in the current register window. **reg** without parameters prints the IU registers. **reg** *reg_name value* sets the corresponding register to *value*. Valid register names are psr, tbr, wim, y, g1-g7, o0-o7 and l0-l7. To view the other register windows, use **reg** *w*n, where n is 0 - 7.

**reset** Re-initialises the processor and on-chip peripherals to the initially probed values.

**step** [*count*] Resumes the execution at the present position and executes one instruction. If an *count* is given, execution will stop after the specified number of instructions.

**run** [*count/time*]

Initialises the processor and starts execution from the last load address. Any set breakpoints remain.

**tmode** [*proc* | *ahb* | *both* | *none*]

Select tracing mode between none, processor-only, AHB only or both.

**watch** [*address*]Adds a hardware watchpoint at *address*. If *address* omitted, prints all breakpoints.

**wmem** <*address*> <*value*>

Write simulated memory. Only full 32-bit words can be written.

Typing a ‘Ctrl-C’ will interrupt a running program. Short forms of the commands are allowed, e.g **c**, **co**, or **con**, are all interpreted as **cont**.

## 3.2 Running applications

To run a program, first use the **load** command to download the application and the **run** to start it:

```
dsu> lo stanford
section: .text at 0x40000000, size 60992 bytes
section: .data at 0x4000ee40, size 1904 bytes
total size: 62896 bytes (93.7 kbit/s)
dsu> run

Program exited normally.
dsu>
```

The output from the application appears on the normal LEON UARTs and thus not in the DSUMON console. Note that the loaded applications should be compiled with LECCS and **not** run through mkprom. Before an application can be re-executed, it must re-loaded on to the target to initialise the data segment (.data).

## 3.3 Inserting breakpoints

Instruction breakpoints are inserted using the **break** or **hbreak** commands. The **break** command inserts a software breakpoint (ta 1), while hbreak will insert a hardware breakpoint using one of the IU watchpoint registers. To debug code in read-only memories (e.g. prom), only hardware breakpoints can be used. Note that it is possible to debug any ram-based code using software breakpoints, even where traps are disabled such as in trap handlers.

## 3.4 Displaying registers

The current register window can be displayed using the **reg** command:

```
dsu> reg

          INS       LOCALS      OUTS     GLOBALS
   0:  403FFDF8   403FFE08   00000004   00000000
   1:  403FFDE8   40008E74   00000003   08000000
   2:  00000004   40008D38   403FFDF8   00000003
   3:  40017950   00000020   00000083   50000000
   4:  400178AC   00000040   403FFE00   00000001
   5:  00000029   00000040   00000000   00000770
   6:  403FFE20   00000000   403FFD88   00000001
   7:  4000238C   00000000   40007B6C   00000000

 psr: 000000E3   wim: 00000080   tbr: 40000060   y: 00000000

 pc:  40007b84  mov  %i1, %l1
 npc: 40007b88  ld  [%fp - 0x28], %o0
```

Other register windows can be displayed using **reg w*n***, when ***n*** denotes the window number. Use the float command to show the FPU registers (if present).

### 3.5 Displaying memory contents

Any memory loaction can be displayed using the **mem** command:

```
dsu> mem 0x40000000

 40000000  a0100000  29100004  81c52000  01000000    ....)..... .....
 40000010  91d02000  01000000  01000000  01000000    .. .............
 40000020  91d02000  01000000  01000000  01000000    .. .............
 40000030  91d02000  01000000  01000000  01000000    .. .............

dsu>
```

### 3.6 Disassembly of memory

Any memory location can be disassembled using the dis command:

```
dsu> di 0x40000000 5

 40000000  a0100000   clr  %l0
 40000004  29100004   sethi  %hi(0x40001000), %l4
 40000008  81c52000   jmp  %l4
 4000000c  01000000   nop
 40000010  91d02000   ta  0x0
```

Note that also the contents of the instruction cache can be disassembled:

```
dsu> dis 0x90140000 5

 90140000  03100000   sethi  %hi(0x40000000), %g1
 90140004  82106000   or  %g1, %g1
 90140008  81984000   mov  %g1, %tbr
 9014000c  4000356c   call  0x9014d5b0
 90140010  01000000   nop
```

### 3.7 Using the trace buffer

Depending on the LEON configuration, the trace buffer can store the last executed instruction, the last AHB bus transfers, or both. The trace buffer mode is set using the **tmode** command. Use the ahb, inst or hist commands to display the contents of the buffer. Below is an example debug session that shows the usage of breakpoints, watch-points and the trace buffer:

```
jiri@venus:~/dsumon2$ ./dsumon -i

 LEON DSU Monitor, version 1.0
 Copyright (C) 2001, Gaisler Research - all rights reserved.
 Comments or bug-reports to jiri@gaisler.com

 port /dev/ttyS0 at 115200 baud

 clock frequency        :  24.9 MHz
 register windows       :  8
 instruction cache      :  4 kbytes, 16 bytes/line
 data cache             :  2 kbytes, 16 bytes/line
 hardware breakpoints   :  4
 trace buffer           :  128 lines
 mixed cpu/ahb tracing  :  yes
 ram width              :  32 bits
 ram banks              :  2
 ram bank size          :  2048 kbytes
 stack pointer          :  0x403ffff0
dsu> lo stanford
section: .text at 0x40000000, size 60992 bytes
section: .data at 0x4000ee40, size 1904 bytes
total size: 62896 bytes (93.9 kbit/s)
dsu> tm both
combined processor/AHB tracing
dsu> bre 0x40007b84
breakpoint 1 at 0x40007b84
dsu> wa 0x403ffdfc
watchpoint 2 at 0x403ffdfc
dsu> bre
num   address        type
  1 : 0x40007b84    (soft)
  2 : 0x403ffdfc    (watch)
dsu> run
data watchpoint at pc 0x40001a00 reached
dsu> ah
   time      address   type   data    trans size burst mst lock resp
  120828317  4000975c  read   81e80000   3    2    1    0    0    0
  120828324  40004578  read   30800017   2    2    1    0    0    0
  120828326  4000457c  read   d21221b8   3    2    1    0    0    0
  120828330  90000000  write  000045f9   2    2    0    1    0    0
  120828334  400045d4  read   81c7e008   2    2    1    0    0    0
  120828336  400045d8  read   81e80000   3    2    1    0    0    0
  120828338  400045dc  read   9de3bf90   3    2    1    0    0    0
  120828344  40006c08  read   c13fbfd0   2    2    1    0    0    0
  120828346  40006c0c  read   40000928   3    2    1    0    0    0
  120828346  90000000  read   000055f9   2    2    0    1    0    0
dsu> in
   time      address   instruction                      result
  120828287  400096c0  sethi  %hi(0x40013800), %o0 [40013800]
  120828294  400096c4  ldd  [%o0 + 0x220], %f2     [3ff00000 00000000]
  120828304  400096c8  fcmped  %f0, %f2            [3ff00000]
  120828314  400096cc  nop                         [00000000]
  120828315  400096d0  fbule  0x40009754           [00000000]
  120828316  400096d4  sethi  %hi(0x40013800), %o0 [40013800]
  120828320  40009754  ldd  [%fp - 0x38], %f0      [bfe8ab1d 4daa6a20]
  120828325  40009758  ret                         [40009758]
  120828328  4000975c  restore                     [00000000]
  120828337  40004578  ba,a  0x400045d4            [00000000]
dsu> del 2
dsu> bre
num   address        type
  1 : 0x40007b84    (soft)
dsu> cont
breakpoint 0 at 0x40007b84 reached
dsu> hi
  120828287  400096c0  sethi  %hi(0x40013800), %o0 [40013800]
  120828294  400096c4  ldd  [%o0 + 0x220], %f2     [3ff00000 00000000]
  120828304  400096c8  fcmped  %f0, %f2            [3ff00000]
  120828314  400096cc  nop                         [00000000]
  120828315  400096d0  fbule  0x40009754           [00000000]
  120828316  400096d4  sethi  %hi(0x40013800), %o0 [40013800]
```

```
   120828317             ahb read,  mst=0, size=2    [4000975c 81e80000]
   120828320  40009754  ldd  [%fp - 0x38], %f0      [bfe8ab1d 4daa6a20]
   120828324             ahb read,  mst=0, size=2    [40004578 30800017]
   120828325  40009758  ret                          [40009758]
   120828326             ahb read,  mst=0, size=2    [4000457c d21221b8]
   120828328  4000975c  restore                      [00000000]
   120828330             ahb write, mst=1, size=2    [90000000 000045f9]
   120828334             ahb read,  mst=0, size=2    [400045d4 81c7e008]
   120828336             ahb read,  mst=0, size=2    [400045d8 81e80000]
   120828337  40004578  ba,a  0x400045d4             [00000000]
 dsu>
```

When printing executed instructions, the value within brackets denotes the instruction result, or in the case of store instructions the store address and store data. The value in the first column displays the relative time, equal to the DSU timer. The time is taken when the instruction completes in the last pipeline stage (write-back) of the processor. In a mixed instruction/AHB display, AHB address and read or write value appear within brackets. The time indicates when the transfer completed, i.e. when HREADY was asserted.

Note:, when switching between tracing modes the contents of the trace buffer will not be valid until execution has been resumed and the buffer refilled.

# 4 GDB interface

## 4.1 Attaching to gdb

DSUMON can act as a remote target for gdb, allowing symbolic debugging of target applications. To initiate gdb communications, start the monitor with the **-gdb** switch or use the DSUMON **gdb** command:

```
bash-2.04$ dsumon -gdb

jiri@venus:~/tmp/ibm/vhdl/dsumon2$ ./dsumon -gdb

 LEON DSU Monitor, version 1.0
 Copyright (C) 2001, Gaisler Research - all rights reserved.
 Comments or bug-reports to jiri@gaisler.com

gdb interface: using port 1235
```

Then, start gdb in a different window and connect to DSUMON using the extended-remote protocol:

```
(gdb) tar extended-remote venus:1235
Remote debugging using venus:1235
0x40007b84 in __mulsf3 ()
(gdb) lo
```

While attached, normal DSUMON commands can be executed using the gdb **monitor** command. Output from the DSUMON commands, such as the trace buffer history is then displayed in the gdb console:

```
(gdb) mon hi
   time      address   instruction                 result
   21768987  400011dc  or  %g4, 0x240, %g4         [4000ee40]
   21768990  400011e0  sethi  %hi(0x4000f400), %g3 [4000f400]
   21768995  400011e4  or  %g3, 0x1b0, %g3         [4000f5b0]
   21768996  400011e8  subcc  %g3, %g4, %g5        [00000770]
   21769000  400011ec  cmp  %g4, %g2               [00000000]
   21769008  400011f0  ble  0x40001208             [00000000]
   21769016  400011f4  ld  [%g4], %g6              [00000001]
   21769018  40001208  call  0x400052a0            [40001208]
   21769020  4000120c  nop                         [00000000]
   21769023  400052a0  save  %sp, -112, %sp        [403fff00]
(gdb)
```

## 4.2 Debugging of applications

To load and start an application, use the gdb **load** and **run** command.

```
(gdb) lo
Loading section .text, size 0xee40 lma 0x40000000
Loading section .data, size 0x770 lma 0x4000ee40
Start address 0x40000000 , load size 62896
Transfer rate: 50316 bits/sec, 278 bytes/write.
(gdb) bre main
Breakpoint 1 at 0x400052a4: file stanford.c, line 1033.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/jiri/ibm/vhdl/dsumon2/stanford

Breakpoint 1, main () at stanford.c:1033
1033        fixed = 0.0;
(gdb)
```

To interrupt simulation, Ctrl-C can be typed in both GDB and DSUMON windows. The program can be restarted using the GDB **run** command but a **load** has first to be executed to reload the program image on the target. Software trap 1 (ta 1) is used by gdb to insert breakpoints and should not be used by the application.

## 4.3 Detaching

If gdb is detached using the **detach** command, the monitor returns to the command prompt, and the program can be debugged using the standard DSUMON commands. The monitor can also be re-attached to gdb by issuing the **gdb** command to the monitor (and the **target** command to gdb).

DSUMON translates SPARC traps into (unix) signals which are properly communicated to gdb. If the application encounters a fatal trap, execution will be stopped exactly on the failing instruction. The target memory and register values can then be examined in gdb to determine the error cause.

## 4.4 Some gdb support functions

DSUMON detects gdb access to register window frames in memory which are not yet flushed and only reside in the processor register file. When such a memory location is read, DSUMON will read the correct value from the register file instead of the memory. This allows gdb to form a function traceback without any (intrusive) modification of memory. This feature is disabled during debugging of code where traps are disabled, since not valid stack frame exist at that point.

DSUMON detects the insertion of gdb breakpoints, in form of the ta 1 instruction. When a breakpoint is inserted, the corresponding instruction cache tag is examined, and if the memory location was cached the tag is cleared to keep memory and cache synchronised.

For correct operation of certain gdb commands such as modification of variables in memory, the LEON processor must be configured with data cache snooping enabled.

## 4.5 Limitations of gdb interface

All nominal gdb debug commands are supported by DSUMON with the exception of hardware data watchpoints.

It is not possible to debug applications in prom since gdb uses software breakpoints by default.

Source-level stepping using the gdb **step** or **next** commands is somewhat slow, but this depends on gdb rather than DSUMON. During these operations, gdb inserts and removes many breakpoints, and fetches the complete processor context after each break, leading to a few seconds delay for each step.

Do not use the gdb **where** command in parts of an application where traps are disabled (e.g.trap handlers). Since the stack pointer is not valid at this point, gdb might go into an infinite loop trying to unwind false stack frames.