VSRV VSOS SHELL

VSRV1

All information in this document is provided as-is without warranty. Features are subject to change without notice.

Revision History			
Rev.	**Date**	**Author**	**Description**
0.03	2025-05-30	HH	Initial release of documentation.

Contents

List of Figures

1 Introduction

The VSRV VSOS Shell is a powerful tool that allows controlling VSRV from an external interface, e.g. the UART.

This document will explain how to use the VSOS Shell Environment.

After the disclaimer and definitions in Chapters 2 and 3, an overview of the Shell is given in Chapter 4, *Overview*. It is followed by requirements in Chapter 5, *Requirements*.

Instruction on using the shell is given in Chapter 6, *Using the Shell Environment*.

The document ends with Chapter 7, *Latest Document Version Changes*, and Chapter 8, *Contact Information*.

2 Disclaimer

VLSI Solution makes everything it can to make this documentation as accurate as possible. However, no warranties or guarantees are given for the correctness of this documentation.

3 Definitions

CBR Constant BitRate. Bitrate of stream will be the same for each compression block.

DSP Digital Signal Processor.

I-mem 32-bit Instruction Memory.

LSW Least Significant (16-bit) Word.

MSW Most Significant (16-bit) Word.

RISC Reduced Instruction Set Computer.

VBR Variable BitRate. Bitrate will vary depending on the complexity of the source material.

VS_DSP[6] VLSI Solution's DSP core.

VSIDE VLSI Solution's Integrated Development Environment.

VSOS VLSI Solution's Operating System.

X-mem 16/32-bit X Data Memory.

Y-mem 16/32-bit Y Data Memory.

4 Overview

The VSRV VSOS Shell is a new, powerful tool that allows controlling VSRV from an external interface, e.g. the UART.

Using the VSOS Shell interface it is possible to create an powerful companion to the RISC-V / Linux system running under the same hood, not necessarily requiring any specific VS_DSP[6] programming skills.

Nevertheless, for those who are familiar with programming under VSOS, the VSOS Shell Environment makes it possible to create convenient shell commands / programs. By combining these building blocks it is possible to create more complex systems.

5 Requirements

Before using the VSOS Shell Environment, you need to have the following building blocks:

- VSRV1 CAT Board or something similar.
- Latest version of VSOS files downloaded from the VSDSP Forum web site.
- UART or USB->UART cable connected between DevBoard and PC for using the UART interface. Data speed is 115200 bps, format is 8N1.
- Your favorite UART Terminal Emulation program installed on the PC. Note that the current VSOS Shell uses only the line feed character (0x0a) for line feed, and no carriage return (0x0d), so the terminal emulation program needs to be able to handle that (see examples in Chapter 5.1, *Terminal Program Settings*). For Linux computers, microcom is a good option. For Microsoft Windows computers, PuTTY and TeraTerm have been tested and found working.

When all of this is in order, you are ready to use the VSOS Shell Environment.

5.1 Terminal Program Settings

This chapter shows recommended settings for two example terminal programs, tested by VLSI. Other terminal programs can also be used, provided that the crucial parameters are set in a similar way.

5.1.1 Linux / Settings for microcom

On ubuntu-based systems, Microcom can be installed with:
`sudo apt-get install microcom`

After connecting the USB-UART cable between your PC and teh CAT Board, and checking the device name (`ls -lart /dev` and/or `sudo dmesg`), you may start microcom as follows, at 115200 bps, 8N1:
`microcom -p /dev/ttyUSB0 -s 115200`).

If you get the following message
`Exitcode 2 - cannot open device /dev/ttyUSB0`
it usually means that you are not in a group allowed to access the serial port. You can either run microcom as root with sudo, or, preferably, add yourself to the dialout group. Run:
`sudo adduser myusername dialout`
then logout and login from the whole windowing system (or reboot) for the change to take effect. (Replace `myusername` with your username.)

5.1.2 Windows / Settings for PuTTY

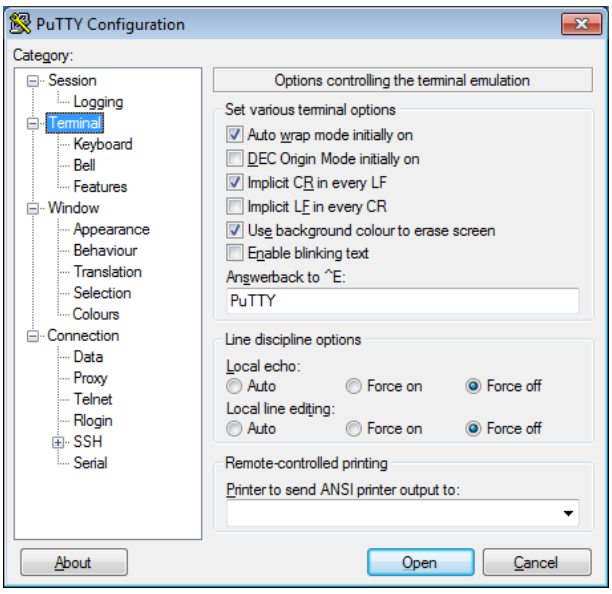

Figure 1: PuTTY Configuration: Terminal

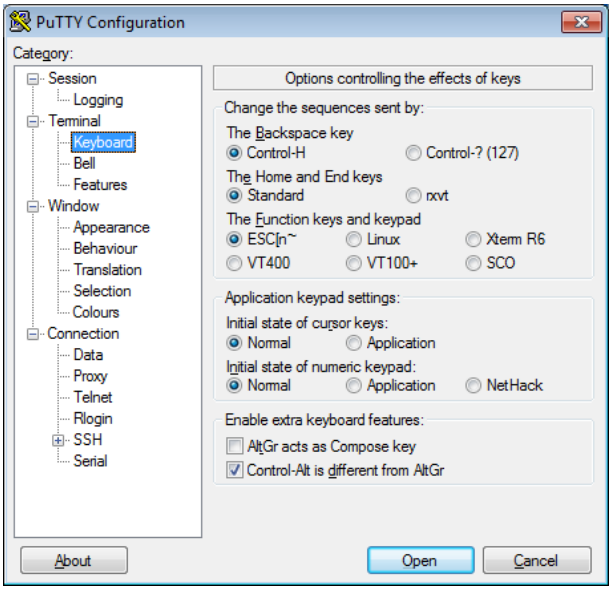

Figure 2: PuTTY Configuration: Keyboard

Figure 1 shows the terminal emulation settings for PuTTY. Make sure you check the “Implicit CR in every LF” box. In the keyboard settings in Figure 2, you may set Backspace key to either Control-H or Control-?.

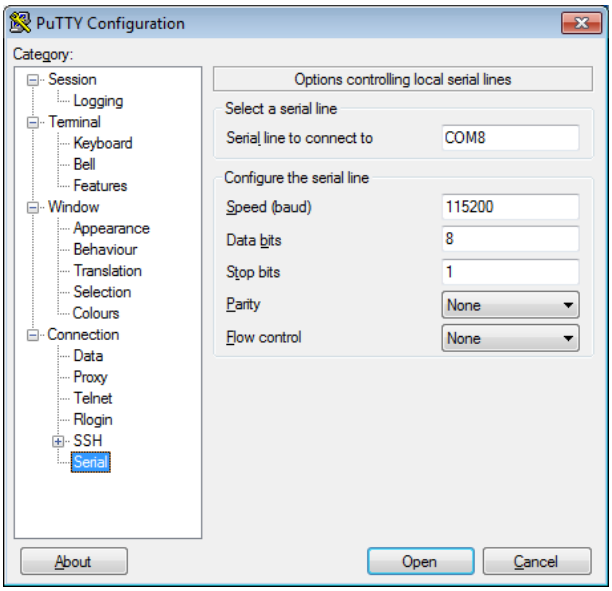

Figure 3: PuTTY Configuration: Serial

Figure 3 shows the serial communication parameters for 115200 bps, 8N1. For binary file transfers to work, it is important to disable flow control.

5.1.3 Windows / Settings for Tera Term

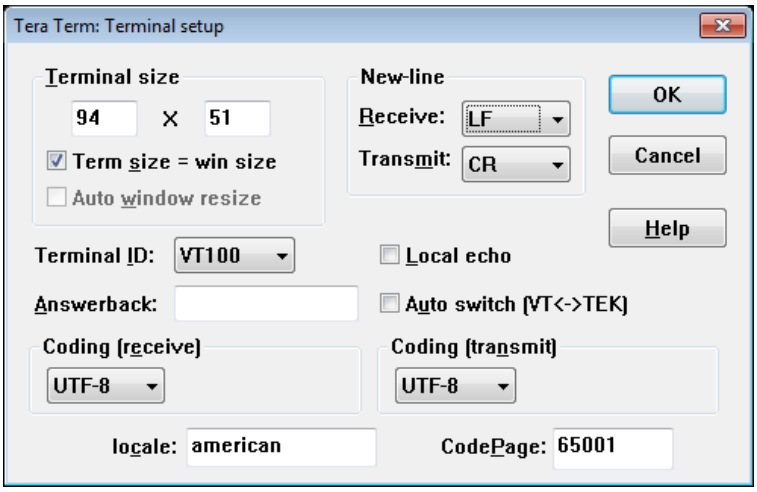

Figure 4: Tera Term: Terminal Setup

Figure 4 shows how to set line feeds and basic terminal emulation mode (VT100) in Tera Term.

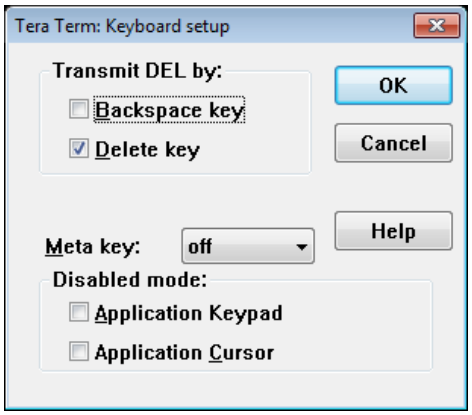

Figure 5: Tera Term: Keyboard Setup

Figure 5 shows a working keyboard setup for Tera Term.

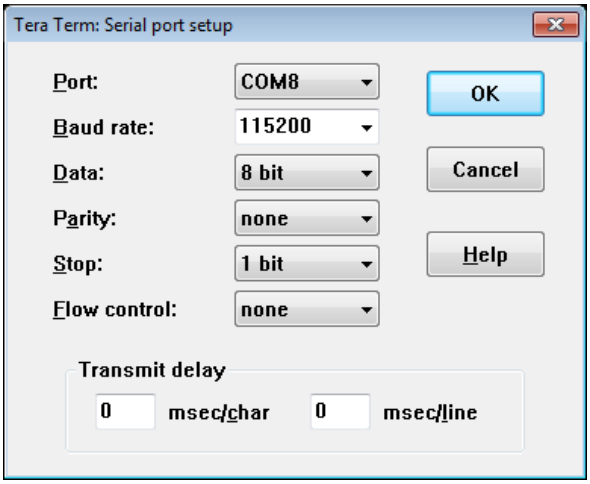

Figure 6: Tera Term: Serial Port Setup

Figure 6 shows how to set serial parameters 115200 bps, 8N1, without flow control.

6 Using the Shell Environment

The shell environment have the following features:

- Very small (roughly 1/4 kilowords code size).
- Executes programs / loads drivers from the S:SYSR/ folder.
- Can also execute .DR3 programs from other locations if full path is provided.
- Interactive command line editing.
- Command history:
  - Up/down arrow (vt100 terminal).
  - “history” shows command history.
  - “!!” repeats last command.
  - “!-n” repeats n’th last command. E.g. “!-1” is equivalent to “!!”.
  - “!xx” repeats last command that started with xx.
- Concept of current directory.
- Buffered, interrupt-handled UART stdio driver.
- Ctrl-C to notify programs that they should close.
- Ctrl-C three times to hard reset VSRV (can be disabled with command line options to echo).

This package contains several programs for the shell, many of which take command line arguments.

The shell commands can also be accessed from the C environment through the function RunProgram() in <apploader.h>.

Example:

```
ioresult errorCode = RunProgram("dir", "-st -r")
```

is equivalent to typing

```
S:>dir -st -r
```

in the VSOS Shell command line. Usually programs return S_OK for a successful run, and some other value if there were problems.

The shell environment also allows dynamic loading and unloading of system drivers with the command DRIVER (Chapter 6.1.14). This is very useful in a memory-limited system with lots of functionality.

6.1 Shell Command Programs and VSOS Libraries

This section presents some of the shell programs and drivers included in this package. They are located in the VSOS system disk's directory S:SYSR/ .

The program are in rapid development, so this documentation will necessarily be somewhat out of date most of the time.

To get more information on how to run these programs / use these libraries:

- If the file is a shell command and not a driver, try running it with the "-h" command line option.
- Some shell commands give help if run without any parameters.
- Source code for most of the shell commands / libraries is available in the VSOS 3.60 Root and Libraries Source Code Package (or its current version), in the directory *solutions*. Many of the solutions contain contain README.TXT files that contain very useful documentation beyond the scope of this document.

Many programs offer a "-h" command line option to show what options they can handle. Also running a program without any parameters may give useful information.

6.1.1 audiodec

Audio decoder main library.

Not ported yet as of writing this (2025-05-30).

6.1.2 aui*, auo*, aux*

Audio input and output drivers. For details on how to use the audio drivers, read the separate document *VSRV VSOS Audio Subsystem*, which will be released before end of Q3/2025.

6.1.3 AuInput

Controls stdaudioin or other audio inputs. For details, read the separate document *VSOS Audio Subsystem*.

6.1.4 AuOutput

Controls stdaudioout or other audio outputs. For details, read the separate document *VSOS Audio Subsystem*.

6.1.5 BadBit

```
Usage: BadBit [-h|-v|+v|+f|-f] fileName
-f|+f Fatal errors on|off
-v|+v Verbose on|off
-h Show this help
```

Read a Bad RAM Bit list from the SPI Flash and map away problematic memory locations.

6.1.6 cd

```
Usage: cd [-v|+v] [-h] dir
-v|+v   Turn verbose on/off
-h      Show this help
Note: cd without parameters shows the current directory
      cd :: lists all available devices
```

Change or print current directory or list all devices.

6.1.7 CopyF

```
Usage: CopyF [-h] [src dst]
src     File to be copied
dst     Destination file
-h      Show this help
```

Copy a file. Long file names are lost when copying.

6.1.8 Crc32

```
Usage: Crc32 [-v|+v|-i|+i|-h] file1 [...]
-v|+v   Verbose on|off
-i|+i   Ignore CRC32 calculation on|off
-h      Show this help
```

Print size and CRC32 checksum for a file.

Crc32 can also be used as a read speed test with the “-i” option, e.g.

```
S:>time Crc32
errCode 0, time:  0.038s
```

```
S:>time Crc32 -i d:CompleteAlbum.mp3
BYTES 0x0cfcc6db
errCode 0, time: 22.966s
```

In the example the read speed was 0xcfcc6db Byte / (23.037 s - 0.040 s) = 9503340 B/s, or 9281 KiB/s, or 9.06 MiB/s.

6.1.9 DDRLoad

```
Usage: DDRLoad [-r|+r|-c|+c|-s|+s|-v|+v|-h] [-aaddr] [-bfile] [f1.vri [...]]
-r|+r Verify VRI file after writing on|off
-c|+c Calculate file CRC32 on|off
-s|+s Initialize DDR using 12 MHz CPU on|off
-v|+v Verbose on|off
-aaddr Set address for binary load to addr (current 0x80000000)
-ah Set address for binary load to highest seen addr (0x00000000)
-bfile Load binary file
-B Load binary blob pointed to by symbol _ddrBlob
-h Show this help
f1.vri Load VRI file f1.vri
```

Load data files to DDR memory, then start the RISC-V processor.

See also:
RVParam (Chapter 6.1.24).

6.1.10 dec*

Decoder libraries for different compressed audio formats used by the audiodec library.

If you want to use decwav_pcm from SYS_everything/decwav_pcm.dl3 of the VSOS Root and Libraries Source Code Package, first copy it to your S:SYSR/ directory, then rename it decwav.dl3.

See also:
audiodec (Chapter 6.1.1), enc* (Chapter 6.1.17).

6.1.11 del

```
Usage: Del [-v|+v|-h] file|dir
-v|+v   Verbose on/off
-h      Show this help
```

Deletes a file or recursively a directory.

6.1.12 Dir

```
Usage: dir [-s|-sn|-st|-ss|+s|-r|+r|-d|+d|-a|+a|-f|+f|-h] [path]
-s|-sn  Sort files by name (default)
-st     Sort files by date
-ss     Sort files by size
+s      Don't sort files (faster with large directories)
-r      Reverse sort (if sort selected)
+r      Forwards sort (if sort selected)
-d      Show file date
+d      Don't show file date
-a      Only audio files
+a      All files
-f      Fast listing (don't show play time for MP3 files with -a)
+f      Slower listing (show play time also for MP3 files with -a)
-v      Verbose on
+v      Verbose off
-h      Show this help page
```

dir lists the contents of a directory. It can optionally examine each file and list only those ones that are audio media files.

If "-sn" is defined but the directory is too large to be sorted in memory, dir lists the files in file system order.

The output of dir is as e.g. as follows:

```
D:ManMachine/>dir +s
-    3. 03_MET~1.WAV   63755708 2015-07-17 11:38:16 03_Metropolis.wav
-    4. 04_THE~1.WAV   39508940 2015-07-17 11:38:26 04_TheModel.wav
-    5. 05_NEO~1.WAV   94268204 2015-07-17 11:38:36 05_NeonLights.wav
-    6. 06_THE~1.WAV   58616588 2015-07-17 11:38:44 06_TheManMachine.wav
-    7. 01_THE~1.WAV   65656124 2015-07-17 11:37:58 01_TheRobots.wav
-    8. 02_SPA~1.WAV   62633804 2015-07-17 11:38:08 02_Spaceleb.wav
```

The fields are as follows:

1. Entry type. 'D' is a directory, '-' is a normal file, and 'L' is a volume label.
2. The file system order of the entry
3. Short name
4. File size in bytes
5. File date in YYYY-MM-DD format
6. File time in hh:mm:ss format
7. Long name, displayed with UTF-8 encoding

When listing audio files, the output format is different:

```
D:ManMachine/>dir -a
-    7. 01_THE~1.WAV    6:12.2  44100 2 2015-07-17 11:37:58 01_TheRobots.wav
-    8. 02_SPA~1.WAV    5:55.1  44100 2 2015-07-17 11:38:08 02_Spaceleb.wav
-    3. 03_MET~1.WAV    6:01.4  44100 2 2015-07-17 11:38:16 03_Metropolis.wav
```

```
-    4. 04_THE~1.WAV    3:44.0  44100 2 2015-07-17 11:38:26 04_TheModel.wav
-    5. 05_NEO~1.WAV    8:54.4  44100 2 2015-07-17 11:38:36 05_NeonLights.wav
-    6. 06_THE~1.WAV    5:32.3  44100 2 2015-07-17 11:38:44 06_TheManMachine.wav
```

Now the fields are as follows:

1. Entry type. 'D' is a directory, '-' is a normal file, and 'L' is a volume label.
2. The file system order of the entry
3. Short name
4. File playback time in m:ss.t format
5. Sample rate
6. Number of audio channels
7. File date in YYYY-MM-DD format
8. File time in hh:mm:ss format
9. Long name, displayed with UTF-8 encoding

Determining playback times for MP3 files requires scanning the whole file, and may be slow. Because of this, unless +f is defined with -a, MP3 file playback times are shown as 0:00.0.

With the -v option, Dir prints more information about each file, including its cluster chains. It also checks if the cluster chain length is consistent with file length. An example of a very fragmented file below:

```
S:>dir s:sys
[...]
-   37. HIRESREC.DL3      67620 2018-01-31 09:23:06 HiResRec.dl3
        First block 0x150,  Cluster list: 0x29, 0x34, 0x68, 0x6a, 0x8a-0x96
        Total cluster chain 0x11, file size 0x11, OK
[...]
```

6.1.13 DiskFree

```
Usage: DiskFree [-p x|-H|+H|-v|-h] [D:]
        There may be multiple drive parameters.
        Drive parameters must be after -p|-H options.
-p x    Store last result in KiB to X-memory 32-bit pointer pointed to by x
        (Usable for calling with RunLibraryFunction(); makes operation silent)
-f x    Stop looking after finding x MiB free memory
        Option must be before drive name
-H|+H   Display sizes in human-readable form on/off
-v|+v   More/less verbose (twice makes even more/less verbose)
D:      Print results for drive D (:: means all drives)
-h      Show this help

Cluster printout:
#       All clusters used
+       Some clusters used
.       All clusters free
```

Displays the amount of free space on a drive, or, with the verbose option, also disk size and its fill state. By giving the verbose option twice, also print a map of allocated clusters of a disk.

Because of the way FAT allocation tables work, running DiskFree on a larger drive may take significant amounts of time; e.g. analyzing a 4 GiB SD card can take up to a second. used.

Example with a large 1 TB SD card, formatted to FAT32:

```
S:>diskfree -v -v -H d:
Cluster  4194304 KiB per symbol: # all used, + partially used, . all free
0x0000000 +##+....+####+.....................................................
0x0200000 ...................................................................
0x0400000 ...................................................................
0x0600000 ...................................................................
0x0800000 ...................................................................
0x0a00000 ...................................................................
0x0c00000 ...................................................................
0x0e00000 ...................................................................
0x1000000 ...................................................................
0x1200000 ...................................................................
0x1400000 ...................................................................
0x1600000 ...................................................................
0x1800000 ...................................................................
0x1a00000 ...................................................................
0x1c00000 ...........................................................
Drv Used  Free Total  Use%  Name
D:  8.4G  945G  953G    1%  SD/SD Card
```

6.1.14 driver

```
Usage: DRIVER +libname|-libname|?libname
```

Can be used to load, unload, and list libraries in memory.

Examples:

- `driver +auiadc 48000 line1_1 line1_3` loads the AUIADC.DL3 driver to memory (if not already there), and gives it parameters.
- `driver -auiadc` unloads the AUIADC.DL3 driver from memory if no other programs are accessing it. Only the beginning of the driver name needs to be written.
- `driver ?auiadc` checks whether the driver is currently in memory.

See also:
Frags (Chapter 6.1.18).

6.1.15 echo

```
Usage: echo [-n|+n|-e|+e|-r|+r|-w|+w|-u|+u|-|-h] string
-n      No newline
+n      Output newline
-e|+e   Turn shell interactive echo mode off/on (now: on)
-r|+r   Turn RAW tty mode off/on (now: off)
        When on, three Ctrl-C's will not reset board
-u|+u   Turn UTF-8 clean mode off/on (now: off)
        When on, UTF-8 codes 0xF000-0xFFFF will work
-b|+b   Turn console Ctrl-C boot prevention off/on (now: off)
-w|+w   Wait/Don't wait 10 ms after printing before exit
-       End of parameters
-h      Show this help
Note: String may contain escape codes such as \" and \n
```

Echoes its input on the screen, and/or set TTY mode.

Options "-e" and "+e" respectively. The non-interactive mode may be easier to handle when the shell is used with a microcontroller.

Options "-r" and "+r" switch the RAW tty mode off and on, respectively. When RAW tty mode is off, sending three consecutive Ctrl-C characters (ASCII code 0x03) to VSRV will cause a hardware reset.

Options "-u" and "+u" switch the UTF-8 clean mode off and on, respectively. When UTF-8 clean mode is off, sending a UTF-8 code point between 0xF000...0xFFFF to VSRV will cause the system to go into vs3emu connection mode. When UTF-8 clean mode is on, no character codes will force the system to go to vs3emu connection mode. You then need to run vs3emuc (Chapter 6.1.34) for that.

Options "-r" and "+r" switch triple Ctrl-C boot off and on, respectively. While often a convenient shorthand, botting on a triple Ctrl-C is not usually a wanted operation when, for example, running a terminal to the Linux side of VSRV.

Option "-w" may help when it is important that the print-out has ended before the next program is run.

6.1.16 edit

```
Usage: Edit [-r rows] [-c columns] [-h] fileName
-h      Show this help
```

Edit is a very simple, VT100-compatible full-screen text editor. It is mainly intended to edit short configuration files. It will *not* preserve CR/LF combinations or any binary codes, so never try to modify a binary file with it.

For a full listing of Edit's capabilities, start it, then push Ctrl-P.

When Edit starts, it reads the configuration file S:SYSR/EDIT.INI. By changing the definitions there, you may change the way backspace and delete keys work.

6.1.17 enc*

Audio encoder drivers for various compressed formats. For details, read the separate document *VSOS Audio Subsystem*.

On how to use these libraries, see for instance the source code for *Rec*.

See also:
audiodec (Chapter 6.1.1), dec* (Chapter 6.1.10).

6.1.18 Frags / MEMTRACK

```
Usage: Frags [-v|+v|-h]
-v|+v   Verbose on|off
-h      Show this help
```

List all libraries, and show the amount of static Instruction, X-Data and Y-Data memory they consume as ASCII graphics. With the help of the MEMTRACK.DL3 driver Frags can also show dynamically allocated memory.

You can run Frags from the VSOS Shell command line. However, more detailed output can be shown if you add the following line as the first line of config.sys:

```
MEMTRACK
```

You may also load the memory tracker from the VSOS Shell as follows:

```
S:>driver +memtrack
```

However, in this case the memory tracker cannot give as detailed information as if it was started earlier in the boot process.

The output of Frags looks as follows:

```
S:>frags
Libs: Memtrack Sdsd Auodac auIi2ss auXsyncs auxPlay Run Uartin sHell Frags
I 0000: ################ ################ ################ ################
I 1000: ################ ################ ################ ################
I 2000: ################ ################ ################ ################
I 3000: MMMMMMMMMSSSSSSS SSSSSSSSSSSAAAAA AAAAAAAIIIIIIIIX XXXXXXXXPPPPPRUU
I 4000: UUUUUUUHHHFFFFFFF FFFFFFF......... ................ ................
I 5000: ................ ................ ................ ................
I 6000: ................ ................ ................ ................
```

```
I 7000: ............... ............... ............... ...............#

X 0000: ################ ################ ################ ################
X 1000: ################ ################ ################ ################SS
X 2000: ##IXPP#PPPP#HHHH HHHHHFFFFFF#.... ............... ...............
X 3000: ............... ............... ............... ...............
X 4000: ............... ............... ............... ...............
X 5000: ............... ............... ............... ...............
X 6000: ............... ............... ............... ...............
X 7000: ............... ............... ............... ...............

Y 0000: ################ ################ ################ MMMMMMMFIIIIIIII
Y 1000: UUUUUU.......... ............... ............... .####..........
Y 2000: .......PPPP..... ............... ............... ...............
Y 3000: ............... ............... ............... ...............
Y 4000: ............... ............... ............... ...............
Y 5000: ............... ............... ............... ...............
Y 6000: ............... ............... ............... ...............
Y 7000: ################ ################ ################ ################

Free: I: 14864 words (45%), X: 22788 words (69%), Y: 23648 words (72%)
FLow: I: 11408 words (34%), highestISoFar: 0x5330
```

The first line shows the libraries that reside in memory. Below that, memory maps for Instruction, X Data, and Y Data memories are shown.

Free areas are marked with '.', allocated unknown areas (usually reserved by VSOS) with '#', and areas allocated by libraries with capital letters (e.g. 'M' of Memtrack). Finally, the total free amount of each memory type is shown followed by the lowest free amount of Instruction memory seen after last boot time and highest free Instruction Memory pointer encountered.

See also:
Driver (Chapter 6.1.14).

6.1.19 getcmd

```
Usage: getcmd
```

Library to reads a command line for the shell.

To see how to call this library, see the source code for The Shell.

6.1.20 More

```
Usage: More [-c col] [-r rows] [-x|+x] [-h] fileName
-c col  Force number of columns
```

```
-r rows Force number of rows
-t/+t   Simple "type" mode (no formatting)
-x/+x   Hex mode on / off
-h      Show this help
```

Present an ASCII or binary file one page at a time.

See also:
Type (Chapter 6.1.32).

6.1.21 paramspl

```
Usage: paramspl parameters
```

Library that splits a parameter string to null-terminated strings, taking into account parenthesis and escape characters, then returns the number of parameters it created by splitting.

To see how to use this library, see the source code for Echo.

See also:
Echo (Chapter 6.1.15).

6.1.22 PReg

```
Usage: preg [-v|+v|-b|+b|-f|+f|-sSym|-h] [rgr]|[rg=val|rg|=val|rg&=val|rg^=val]
rgr     number, number-number, name
rg      number, nameprefix
val     value, may be preceded with ~ for bitwise not
        For memory outside of Y space, precede with X: or I:
-l      List all registers (decrease verbosity to not get all bits)
-v/+v   Increase / decrease verbosity
-b/+b   Bit mode on/off
-f/+f   Fast mode on/off (removes disassembly symbols)
-sSym   Find public symbol Sym and print its address ('-' show all)
-h      Show this help

Examples:
  preg i:0x20-0x3f
  preg ana_cf
  preg -b 2g
  preg +v -l
  preg y:0xfec0=0x1010
  preg uart_data=0x40
  preg 0xfca1&=~0x0200
```

```
preg -svo_printf
```

Debug program that prints / sets / modifies contents of registers or memory. When printing or setting registers in the Y memory space, symbolic register names as defined in vsrv.h may be used. It is possible to write only a prefix of a register name. If printing, all registers with the same prefix are printed. If setting, the fitting register with the lowest address is set.

In addition to setting registers / memory locations with operator ('='), preg can modify them with bitwise or ('|='), bitwise and ('&='), or bitwise exclusive or ('^='') operations. If any of these operations are used, the contents of the memory location is read from before written to. Any value may be preceded with a not symbol (tilde, '~').

Preg can also print addresses of symbolic dependencies with the -s option.

6.1.23 RateCount

```
Usage: RateCount [-i|-o|-ddrv|-pfp|-v|+v|-h]
-i Connect to stdaudioin
-o Connect to stdaudioout
-ddrv Connect to audio driver DRV.DL3
-pfp Set output audio file pointer to fp (use with caution!)
-v|+v Verbose on|off
-h Show this help
```

Analyze audio input/output sample rate and sample bursts.

6.1.24 RVParam

```
Usage: RVParam
-h Show this help
-v/+v Controls verbosity
-w/+w Wait in bootloader
-mAD:DR Sets MAC address
```

A driver that allows setting RISC-V parameters before loading with DDRLoad.

See also:
DDRLoad (Chapter 6.1.9).

6.1.25 RVTerm

```
Usage: Term [-v|+v|-a|-b|-s|+s|-h]
```

```
-v|+v Verbose on|off
-sx Set bit speed to x bps
-h Show this help

Console keys:
F4 - Exit
F5 - Toggle Hex Mode
```

A terminal emulation program that connects to RISC-V's 16550 UART0 using VSDSP's UARTRV.

See also:
Term (Chapter 6.1.30).

6.1.26 SetClock

TBD.

6.1.27 Shell

The VSOS Shell is the command line environment for VSOS. The shell never exits.

6.1.28 Sine

```
Usage: Sine freq1 dbl1 dbr1 [freq2 dbl2 dbr2 [...]] [-v|-t|-rrate|-h]
freq dbl dbr    Set frequency to freq Hz, left volume to dbl dB,
                right volume to dbr dB. If dbl or dbr is greater than zero,
                volume is muted for that channel.
-v      Verbose
-t      Generate triangle waveform instead of sine
-rrate  Set sample rate to rate Hz
-x Trash DAC hardware instead of proper sine
-h      Show this help

Examples:
  Sine 1000 0 0
  Sine -r24000 1000 -6 1 2000 1 -6 3000 -6.1 -6.1
```

Generate one or several sine waves.

Examples:

```
S:>Sine 1001.2371 0 0
S:>Sine -r24000 1000 -6 1 2000 1 -6 3000 -6.1 -6.1
```

6.1.29 Tasks

```
Usage: Tasks [-v|+v] [-h]
-v|+v   Verbose on/off
-h      Show this help
```

Show running tasks, their timer queues and interrupts. With the verbose option, also show registers, stacks traces and hardware queues.

An example output for tasks is shown below:

```
S:>Tasks

Task 0x0021, priority 1, in RUNNING, name "MainTask"
 State: 3 (TS_READY)
  Stack: Start 0x0030, size 0x200, in use 0xd9, max used 0x172 (0x8e free)
  Stack Trace: current PC 0x4369, Tasks::main[185]
    Next: PC 0x1e42 @ stack 0x008c, KERNEL
    Next: PC 0x41d0 @ stack 0x0086, shell::main[43]
    Next: PC 0x04c1 @ stack 0x007c, KERNEL
    Next: PC 0x0569 @ stack 0x003e, KERNEL
    Next: PC 0x0087 @ stack 0x0032, KERNEL

Task 0x1c01, priority 10, in waitQueue, name "cyclic"
  State: 4 (TS_WAIT)
  Stack: Start 0x1c10, size 0x100, in use 0x29, max used 0x3a (0xc6 free)
  Stack Trace: current PC 0x918f, IROM
    Next: PC 0x2b79 @ stack 0x1c1e, KERNEL
    Next: PC 0x90b5 @ stack 0x1c11, IROM::exit
  Registers:
    i0:0x1c1f i1:0x0012 i2:0x1beb i3:0x1c08
    i4:0x1c1e i5:0x0000 i6:0x1c2b i7:0xfc08
    a2:0x0000 a1:0x0004 a0:0x8000 b2:0x0000 b1:0x0000 b0:0x0000
    c2:0x0000 c1:0x0000 c0:0x0064 d2:0x0000 d1:0x0005 d0:0x0004
    p1:0x0000 p0:0x0000 ls:0xebab le:0xffff lc:0x0000 mr0:0x0210 lr0:0x9184

Task 0x21a1, priority 2, in waitQueue, name "AUXPLAY"
  State: 4 (TS_WAIT)
  Stack: Start 0x21b0, size 0x100, in use 0x40, max used 0x57 (0xa9 free)
  Stack Trace: current PC 0x918f, IROM
    Next: PC 0x3b82 @ stack 0x21d5, auii2ss::AudioRead[68]
    Next: PC 0x091d @ stack 0x21cc, KERNEL
    Next: PC 0x3dfb @ stack 0x21c4, auxplay::AudioTask[27]
    Next: PC 0x90b5 @ stack 0x21b1, IROM::exit
  Registers:
    i0:0x21d6 i1:0x0012 i2:0x0e30 i3:0x21a8
    i4:0x21d5 i5:0x0000 i6:0x21e2 i7:0xfc08
    a2:0x0000 a1:0x0004 a0:0x8000 b2:0x0000 b1:0x0000 b0:0x0000
    c2:0x0000 c1:0x2150 c0:0x0080 d2:0x0000 d1:0x0001 d0:0x0030
    p1:0x0000 p0:0x0040 ls:0xebab le:0xffff lc:0x0000 mr0:0x0210 lr0:0x9184

Timer queue 0x1c1f for task 0x1c01 ("cyclic")
  Tick count: 0x0064
```

```
Timer queue 0x21d6 for task 0x21a1 ("AUXPLAY")
  Tick count: 0x0001

Interrupts:
  INT  0 INT_DAC     , pri 2, vector 0x37e5= auodac::DacInterrupt
  INT  5 INT_MAC1    , pri 2, vector 0x3a7c= auiadc::Dec6Interrupt
  INT 12 INT_UART_TX , pri 1, vector 0x3d70= uartin::UartTransmitInterrupt
  INT 13 INT_UART_RX , pri 1, vector 0x3da0= uartin::UartReceiveInterrupt
  INT 15 INT_TIMER1  , pri 2, vector 0x2a09= KERNEL
  INT 16 INT_TIMER2  , pri 2, vector 0x2a09= KERNEL

Hardware locks:
  BUFFER  4  locked:YES  in_queue:no  name:HLB_4
  IO     10  locked:YES  in_queue:no  name:HLIO_SD
  PERIP   5  locked:YES  in_queue:no  name:HLP_SD
```

There are three tasks running, the VSOS MainTask, the VSOS cyclic task, and a task for AUXPLAY driver. The cyclic task is running at the highest priority (10).

There are also two timer queues where cyclic and AUXPLAY are currently waiting. cyclic still has 0x64 ticks = 100/1000 seconds before it will next be wokrn up, but AUXPLAY will be woken up at the next 1/1000 s.

Stack status for each task is also shown. If the stack for a task ever gets full, memory will be trashed and the system may crash.

With the `-v` option also stack traces are shown for each task. For instance, you can see that current execution of AUXPLAY is in IROM address 0x918f (actually the Delay() function), called by AudioRead() from library auii2ss, which was called by a VSOS Kernel function (read() in this case), which again was called by the AudioTask() function of library AUXPLAY.

Then, all active interrupts are shown. With the `-v` option the output is as detailed as in the example. There you seen the interrupt numbers, their symbolic names, their priorities (higher is better), jump vectors, and the jump vector's symbolic name if available.

Finally, hardware locks and their queues are shown. If in_queue for any locks is 1, there may be an unrecoverable hardware lock race condition.

6.1.30 Term

```
Usage: Term [-v|+v|-a|-b|-s|+s|-h]
-v|+v Verbose on|off
-a|-b Mux VSDSP|RV to UART
-s|+s Slow TX on|off
-c|+c Disable Ctrl-C reset on/off (default: on)
-d Don't set divider
-h Show this help
```

```
Console keys:
F2 - MUX VSDSP
F3 - MUX RV
F4 - Exit
F5 - Toggle Hex Mode
F6 - Toggle Slow TX mode
```

A terminal emulation program that connects to RISC-V's BOOT UART using VSDSP's UARTMUX.

See also:
Term (Chapter 6.1.25).

6.1.31 trace

```
Usage: trace startAddr [endAddr]   # trace instruction address [or range]
Usage: trace i:startAddr [endAddr] # trace instruction address [or range]
Usage: trace x:startAddr [endAddr] # trace X data mem  address [or range]
Usage: trace y:startAddr [endAddr] # trace Y data mem  address [or range]
```

Trace (a) memory address(es).

Example:

```
S:>trace i:0x4000
uartin::UartPutChar[44]
```

6.1.32 type

```
Usage: type [-c|+c|-h] [file1 [file2 [...]]]
-c      Print file information line
+c      Don't print file information
-h      Show this help
```

Type a file to screen.

See also:
more (Chapter 6.1.20).

6.1.33 UartIn

Interrupt-based UART stdin/stdout driver.

6.1.34 vs3emuc

Not ported yet.

7 Latest Document Version Changes

This chapter describes the latest and most important changes to this document.

Version 0.03, 2025-05-30

First release.

8 Contact Information

VLSI Solution Oy
Entrance G, 2nd floor
Hermiankatu 8
FI-33720 Tampere
FINLAND

URL: http://www.vlsi.fi/
Phone: +358-50-462-3200
Commercial e-mail: sales@vlsi.fi

For technical support or suggestions regarding this document, please participate at
http://www.vsdsp-forum.com/
For confidential technical discussions, contact
support@vlsi.fi