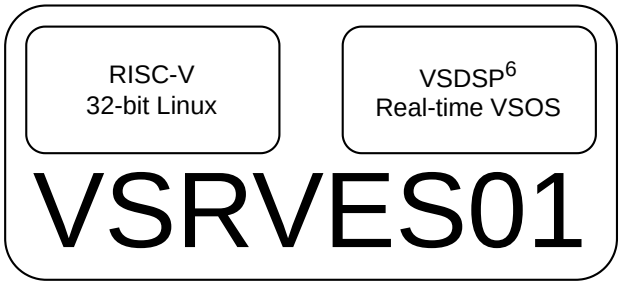

VSRV USER'S GUIDE

How to use and develop with VSRV

Supported chips: VSRVES01

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

Revision History			
Rev.	**Date**	**Author**	**Description**
0.10	2025-06-19	MP	Added documentation on how to compile Linux.
0.03	2025-05-30	HH	Added reference to VSRV VSOS Shell document.
0.02	2025-05-30	HH,MP,HV	Cleanup of existing text.
0.01	2025-05-28	HV,MP,HH	First published preliminary version.

Contents

List of Figures

Listings

1 Introduction

This document is indended to be a tutorial and reference for features in the VSRVES01 prototype chip for details which don't belong to datasheet.

VSRV1 is an RV32IMSU zicsr zifencei RISC-V core implementation designed by VLSI Solution. It is a 32-bit core with a memory management unit and it is capable of running stock Linux.

The VSRVES01 chip has both the RISC-V general purpose prcocessor as well as VLSI Solution's proprietary VSDSP[6] digital signal processor, running VLSI Solution's Real-Time operating system VSOS.

There are many ways to look at the chip. One way is to think of it as an embedded Linux device. Another way is to consider it at as an audio DSP. However as the strengths and weaknesses differ between the processors, the best solution is to take the advantage of the combined system.

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

ADC Analog to digital converter.

B Byte, 8 bits.

b Bit.

DAC Digital to analog converter.

DSP Digital Signal Processor.

I-mem 32-bit VSDSP Instruction Memory.

ISA Instruction Set Architecture.

RAM Random Access Memory.

RISC Reduced Instruction Set Computer.

RISC-V An open standard instruction set architecture (ISA) based on established reduced instruction set computer (RISC) principles.

PCB Printed Circuit Board.

ROM Read Only Memory.

TBD To Be Defined (in the future)

VSIDE VLSI Solution's Integrated Development Environment.

VSOS VLSI Solution's real-time Operating System.

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

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

4 Overview of the VSRV architecture

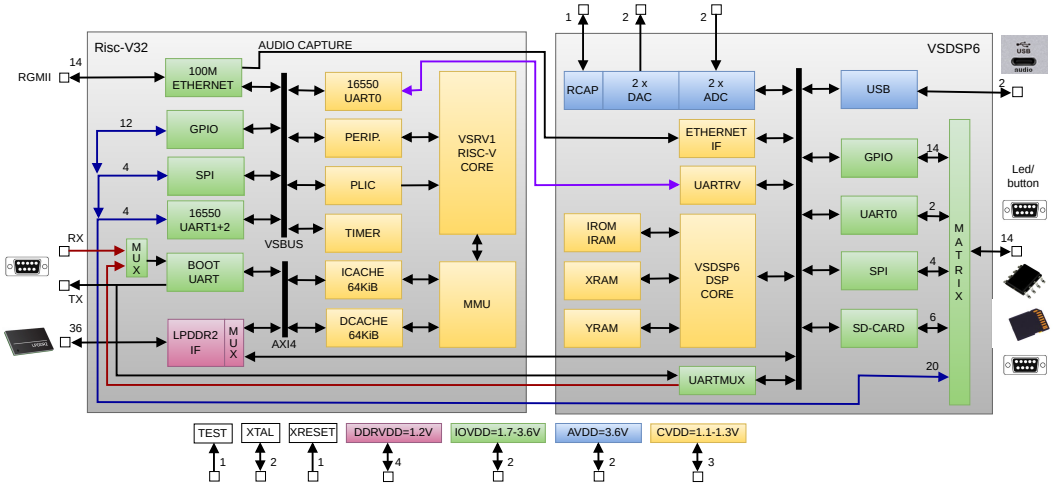

Figure 1: VSRVES01 block diagram

The block diagram of the VSRVES01 is shown in figure 1. The diagram has peripherals colored by the power domains they belong to.

4.1 Clocking

VSRVES01 has a crystal oscillator for connecting a 12.288 MHz crystal. This provides a low jitter and high quality clock for audio and system applications. For performance reasons, there are independent PLLs to provide higher clock speed for VSRV1 and VSDSP. 98 MHz and beyond is usable with the VSRVES01 chip.

4.2 VSDSP^6 core

VSDSP^6 is a 16/40/72-bit digital signal processor. The processor is connected to RAM and ROM 32-bit Instruction Memories (I-mem), RAM X and Y Data Memories (X-mem and Y-mem), as well as many peripherals. A small part of the Y data memory space is used by peripherals.

At maximum bit depth, VSDSP^6 can do the following things each clock cycle:

- A 32x32-bit to 64-bit multiplication and addition to a 72-bit sum, or some other arithmetic or logic operation up to 72 bits, including a barrel shifter.
- Two linear, ring buffer, or bit reverse pointer updates.
- Two 32-bit data memory operations (read and/or write), one for X data memory, one for Y data memory.
- Decrememnt of loop counter and hardware check whether loop has ended.

Having all these features gives VSDSP high signal processing power beyond its MHz figures.

The extremely low latency in serving an interrupt (usually significantly below 10 μs even in a loaded system) allows for implementing a real-time system with audio latencies from input to output in the order of less than 5 milliseconds.

4.3 VSRV1 core

The VSRV1 core is a RISC-V core implementation made by VLSI Solution. The exact flavour of the core is RV32IMSU zicsr zifencei.

VSRV1 has a memory management unit and instruction and data caches, making running Linux feasible.

4.4 VSDSP^6 peripherals

The peripherals of VSDSP^6 are listed below. Please refer to the datasheet for the details.

- RCAP

  The reference voltage generator for analog part. The associated pin expects to have a capacitor to ground so the analog part of the chip has a good reference voltage.

- ADC

  An audio stereo analog to digital converter. The ADC can handle normal line level signals.

- DAC

  DAC converts digital stereo audio samples to analog line level stereo audio signals. The DAC expects to have an output amplifier.

  The sample rate of the DAC can be adjusted with approximately 0.01 Hz steps, which is an absolutely inperceptible step. The sample rate can also be adjusted on the run without any audible glitches. This makes it possible to make minute adjustments to the sample rate when e.g. listening to web radio broadcasts where the receiver has no control over the transmission rate and where clock discrepancies could otherwise in time cause audio starvation or overload, both leading to stuttering.

- GPIO

  All digital pins can be accessed in GPIO mode. The pins are in a matrix and can alternatively be routed to any peripheral functions.

- UART0

UART0 is the main user interface for VSOS running on VSDSP. It has high flexibility in both speed and word length.

- SPI

  SPI is used to connect to an SPI Flash and potentially also to other devices. On startup, VSOS is loaded from SPI Flash. The SPI bus is capable of quite high speed data transfer, in the order to a couple of tens of megabits per second.

- SD Card

  SD Card is another mass media data source for the VSDSP. Current (2025-05-28) version VSOS expects to find the system disk files from SD Card. At a later stage, this will be made optional, the other system disk file option being SPI Flash.

- UARTMUX

  UARTMUX connects to VSRV1 boot UART.

- UARTRV

  UARTRV is connected to VSRV1's 16550 UART0. It acts as an internal communication link between VSRV1 and VSDSP.

- LPDDR2 Access

  VSDSP accees to LPDDR2 is used to load the bootloader and Linux to RISC-V side of the chip. If the RISC-V side is not used, LPDDR2 can also serve as storage space for VSDSP.

- Ethernet audio capture interface

  The Ethernet audio capture interface is designed to forward audio packets to VSDSP, where audio packets can be received and played out from DAC, or otherwise processed.

4.5 VSRV1 peripherals

The peripherals of VSRV1 are listed below.

- Ethernet

  10/100 Mbit/s.

- GPIO

  Optionally routed through the matrix to output pins.

- SPI

  Optionally routed through the matrix to output pins.

- 16550 compatible UART 0

  16500 compatible UART 0 is connected to VSDSP's UARTRV. It acts as an internal communication link between VSRV1 and VSDSP.

- 16550 compatible UART 1 and 2

  Optionally routed through the matrix to output pins.

- Boot UART

  Connected either to pins RV_TX and RV_RX and/or to VSDSP6's UARTMUX. Speed is always RVCLKI/12, where RVCLKI is RISC-V's internal clock.

- LPDDR2 Interface

  LPDDR2 is connected to the Instruction and Data Caches of VSRV1, making it the main RAM memory of the unit. RISC-V cannot be run without LPDDR2.

5 VSRV CAT Board

5.1 Introduction to the CAT Board

So, you have received the VSRV CAT Board and are trying to figure out what to do with it?

The name CAT is a shorthand for Completely Absurd Thing. Or someone just drew a cat on the PCB silkscreen layer.

5.2 CAT Board features

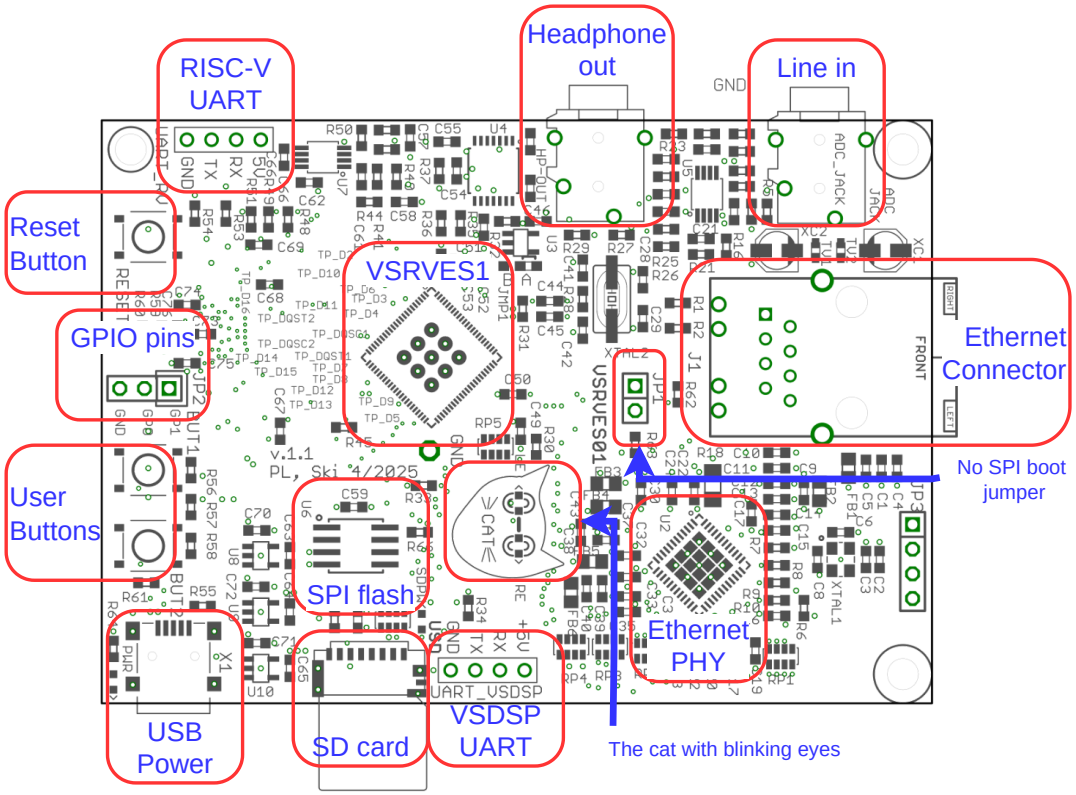

Figure 2: CAT Board block diagram

The CAT Board is a PCB which can be used to run RISC-V Linux and VSDSP VSOS programs. The features of VSRVES01 most likely to be used have been made available with this board.

Figure 2 shows the locations of some features on CAT Board.

Some of the provided features are:

- The CAT
- SD card connector for data and firmware
- SPI flash soldered on board for VSDSP VSOS
- RISC-V and VSDSP UART pin headers
- Ethernet connector
- 3.5 mm jack for line input and headphone output
- 2 user buttons
- Reset button

The LPDDR2 memory chip is located on the bottom side of the board.

5.3 UART communication speeds

CAT Board UART connectors		
UART	**Speed [bps]**	**Features**
(RISC-V) Boot UART	RVCLKI / 12	8 bits, no parity, no flow control
(VSDSP) UART 0	115200	8 bits, no parity, no flow control
Signal level 3.3V, Pins: Ground, Tx, Rx, 5V		

Table 1: Default UART parameters of the CAT Board connectors

The RISC-V Boot UART is connected to RISC-V RX and TX pins as shown in figure 1. It is constantly running at RISC-V clock divided by 12. The easy case is when everything is running with XTALI, 12.288 MHz, and the UART speed is roughly 1 Mbps. However when clock speed around 100 MHz are used, the speed rises closer to 10 Mbps and the signal may be be electrically unusable.

Fortunately there are other means to connect to the RISC-V side. VSOS has a program called “Term” which provides access to the RISC-V Boot UART using the red line in figure 1.

Another option is to set up a DHCP server on your computer and use telnet from pc and connect through ethernet (see chapter 7.3, *Offering DHCP service to CAT Board)*.

5.4 Preventing VSDSP SPI boot

Sometimes, for example when a new operating system is flashed to SPI flash, it is desireable to prevent SPI boot from happening. This can be achieved by shorting the JP1 “No SPI boot” jumper and resetting the chip with the reset button.

Normally, when you reset the board, you will see on the VSDSP UART terminal a boot string like “<C1305Sc2:20:18=10M:24B”, followed by whatever other boot messages there might come. When you short the “No SPI boot” jumper, the message will shorten to “<C1305->”, followed by nothing else.

6 Running the system

Please refer to chapter 7, *Running demo programs on the CAT Board*, for a usage tutorial.

6.1 Preparing the microSD card

You will need a microSD card that has been formatted to the FAT32. ExFAT is not supported.

If you are using Microsoft Windows, you will need an microSD card with a maximum size of 32 GiB. If the card is larger, Windows will not format the card to FAT32.

If using Windows, you can use larger cards, as long as you format them to FAT32. Cards up to 1 GiB have been tested as of writing this (2025-05-30).

Then, copy system root files to this microSD card. Make sure to keep the directory structure intact. The system root files are available at the VSDSP Forum at:
`http://www.vsdsp-forum.com/phpbb/viewtopic.php?t=3242`

6.2 Getting to VSOS shell

There are two documents describing how the VSOS Shell can be used for maximum efficiency. They are:

- VSRV VSOS Shell - Explaining how the shell works and listing its programs and drivers.
- VSRV VSOS Audio - Explaining how the audio subsystem of VSRV VSOS works.

6.3 Starting Linux and accessing it

TBD

7 Running demo programs on the CAT Board

This tutorial will show you through demonstrations of the different tools and drivers. They should be done in order because the later demos build upon the results of the previous ones.

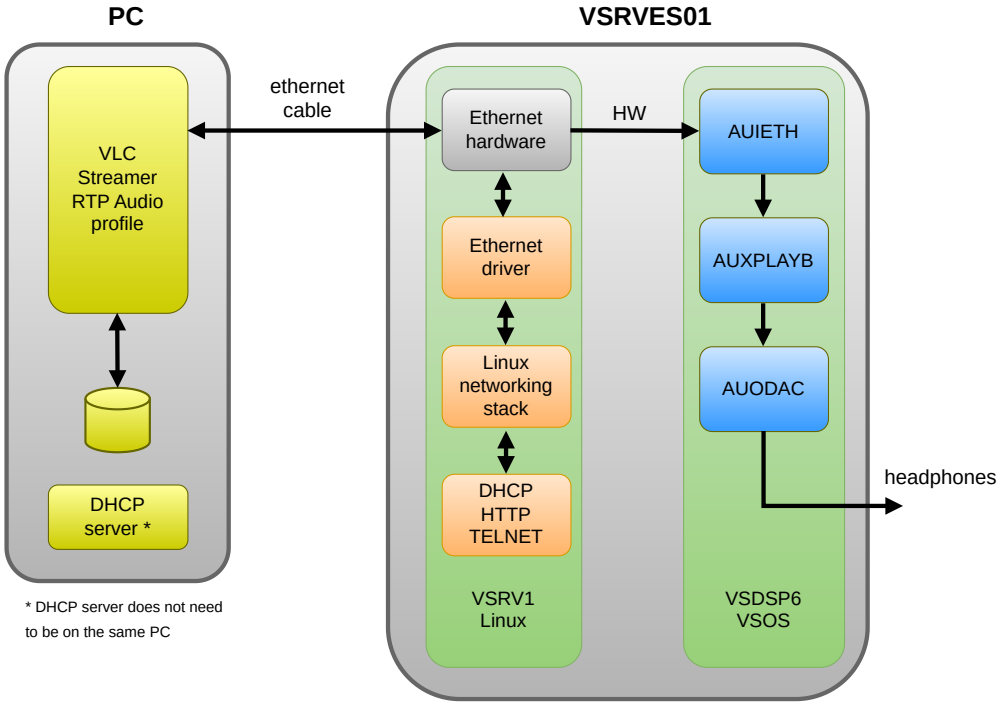

Figure 3: Cat Board VLC audio streaming demo information flow between software and hardware

Figure 3 shows the data and audio paths from the PC running VLC to the earphone output of the CAT Board when running the final demo in Chapter 7.8, *Streaming audio from line input to headphone output*. In the figure VLC, running on a PC, creates an audio stream which is directed by the Ethernet hardware of VSRVES01 to its VSDSP[6]/VSOS side, and then through VSOS drivers AUIETH, AUXPLAYB, and AUODAC to the headphone output.

The following instructions were made with a laptop runnin Kubuntu 24.04.2 LTS, or `Linux retardis 6.8.0-48-generic #48-Ubuntu SMP PREEMPT_DYNAMIC Fri Sep 27 14:04:52 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux`.

To prepare, first see Chapter 5, *VSRV CAT Board*.

After reading the chapter, start up a terminal emulator like microcom at 115200 bps, 8N1:

```
microcom -p /dev/ttyUSB0 -s 115200
```

Install e.g. microcom:

```
sudo apt-get install microcom
```

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. This group is usually called dialout. Verify this by running:
`groups`
Now, you can either run microcom as root with sudo, or, preferably, add yourself to the required dialout group (or whatever gives you access in your Linux/Unix flavour). Run:
`sudo adduser myusername dialout`
Replace `myusername` with your username. Then logout and login from the whole windowing system (or reboot) for the change to take effect.

7.1 Booting the CAT Board

Power up the CAT Board or push the Reset Button. You should now see the following text in quick succession:

```
<C1305Sc2:20:18=10M:24B
<[
Hello.
R'VSOS 3.68 build May 12 2025 10:24:18'
R'VLSI Solution Oy 2012-2025 - www.vlsi.fi'
R'Starting Kernel...'
R'Starting Devices...'
E'SD Ident / Card stuck'
System devices:
- S:  240M SD/SD Card, handled by FAT.
Run S:STARTUP.TXT, config = 0
--  8: BadBit
--  9: Driver +BADBIT
-- 10: BadBit
-- 11: Driver -BADBIT
-- 13: SetClock -l130 98

UART0 115.11 kHz target: 115200 bps, error 0.1%
SPI0 12.29 MHz
SD 49.15 MHz 24.58 MB/s
TIMER0 1000.00 Hz
COREPLL 98.30 MHz
RISCVPLL 98.30 MHz
-- 14: Driver +CATSEYES +l +r
-- 16: Driver +uartin
-- 17: CATSEyes -l +r
-- 19: PReg UARTRV_DIV=0xff14
Y:0xfccb=0xff14 UARTRV_DIV
        15:8   ff   UART_DIV_D1
         7:0   14   UART_DIV_D2
-- 20: PReg UARTMUX_DIV=0xff14
Y:0xfcd3=0xff14 UARTMUX_DIV
```

```
        15:8   ff   UART_DIV_D1
         7:0   14   UART_DIV_D2
-- 22: CATsEyes +l +r
-- 25: Driver +rvparam
-- 27: RvParam -mba:67:11:fd:5f:fa
-- 29: ddrload -brvlbne.bin -B -bcatboard.dtb linux61.vri
Disable Risc-V
DDR parameters:
  Manufact.: 0x8 (Winbond)
  I/O width: 0x1 -> 16 bits
  Density:   0x4 -> 128 MiB
  Type:      0x0 -> S4 SDRAM
  Banks:           8
Load binary file rvlbne.bin @ 0x80000000, 28160 bytes
DDR size 128 MiB
Load binary file _ddrBlob @ 0x80006e00, 56 bytes
Load binary file catboard.dtb @ 0x80006e38, 2465 bytes
Load VSRV image linux61.vri
linux61.vri, 10053940 bytes
Sect  0: addr 80400000, size     168, flags 7 (RWE)
Sect  1: addr 80400168, size     e98, flags f (RWEB)
[... many similar lines omitted ...]
Sect 64: addr 81c7dd20, size    d228, flags 7 (RWE)
0x65 sections, 10053940 bytes in file, 25734984 bytes transferred to DDR
Hand over SD RAM to Risc-V
Enable Risc-V
Execution took 1.065 seconds
-- 30: CATsEyes +l -r
-- 32: Driver +AUIEth s
2a00798e 2a00798e
IB: 0x2000
InitEnd

-- 34: Driver +AUODac s
-- 36: AuOutput -l-12
-- 37: CATsEyes +l +r
-- 39: Driver +AUXPlayB q
-- 44: CATsEyes
-- 45: Driver -CATSEYES
-- 49: Echo Welcome to the shell! Type \"more README.TXT\" on the command line.
Welcome to the shell! Type "more README.TXT" on the command line.
-- 50: !Shell

VSOS SHELL
S:>
```

You are now on the VSOS Shell command line.

7.2 CAT Board VSOS command line

You may feel your way around with for example the following commands:

```
S:>dir
S:>dir -h
S:>dir sysr
S:>type startup.txt
S:>more startup.txt
S:>type readme.txt
```

To get to the Linux side of things, type:

```
S:>term -a
```

If you push enter one extra time, you will be greeted with:

```
rv32 login:
```

That's the Linux login prompt. Try user `root` with password `vsrv`. User `vsrv` with password `vsrv` is also available. However, note that some operations require you to be root, so use root for these initial tests.

To drop from the RISC-V/Linux terminal back to VSDSP/VSOS, push F4. See that your terminal console handles F4 in a VT2xx compatible way, otherwise exiting will not work.

7.2.1 Blinking the cat's eyes

To see what you can do with the cat's eyes, run:
`S:>catseyes -h`

Then you can try the following commands:

```
S:>catseyes -l -r
S:>catseyes +l +r
S:>catseyes /l /r /l /r /r /l /r /l
```

7.3 Offering DHCP service to CAT Board

The CAT Board requires a DHCP server to be able to stream audio through your intranet or provide web services. You need to provide them with a laptop or table computer.

To set up a DHCP server, you can use for example dnsmasq (we will only use its DHCP features, not DNS features). Install it on your computer with:
`sudo apt-get install dnsmasq`

It seems the current Linuxes want to run dnsmasq automatically as a daemon, which we will prevent with:

```
% sudo systemctl stop dnsmasq
% sudo systemctl disable dnsmasq
```

(It may require a reboot before these commands have any effect.)

Now check you ethernet device name with:
`% ip a`

The device name typically starts with the letter "e", for example `eno0` or `enp1s0`. In this example we will use `enp1s0`, but change it to whatever you have.

Now, as root, edit /etc/dnsmasq.conf on your computer. Change the following lines:

```
# disable resolver part, not needed for demo
port=0
no-resolv
no-poll
# adjust line below to reflect ethernet interface on your host which
# will be serving dhcp
interface=enp1s0

bind-interfaces
no-hosts
dhcp-range=192.168.2.120,192.168.2.140,255.255.255.0,5m

# don't send any default route.
dhcp-option=3
```

Now, add an address to your local computer's interface from the same network which is assigned in dhcp-range in the previous configuration file:
`sudo ip a a 192.168.2.42/24 dev enp1s0`
Note: The /24 part is equivalent to netmask 255.255.255.0, just in the different syntax used by "ip".

You are now ready to activate DHCP on your computer:

```
% sudo dnsmasq -d
dnsmasq: started, version 2.90 DNS disabled
dnsmasq: compile time options: IPv6 GNU-getopt DBus no-UBus i18n IDN2 DHCP DHCP
v6 no-Lua TFTP conntrack ipset nftset auth cryptohash DNSSEC loop-detect inotif
y dumpfile
dnsmasq-dhcp: DHCP, IP range 192.168.2.120 -- 192.168.2.140, lease time 12h
dnsmasq-dhcp: DHCP, sockets bound exclusively to interface enp1s0
dnsmasq-dhcp: DHCPDISCOVER(enp1s0) ba:67:11:fd:5f:fa
dnsmasq-dhcp: DHCPOFFER(enp1s0) 192.168.2.139 ba:67:11:fd:5f:fa
dnsmasq-dhcp: DHCPREQUEST(enp1s0) 192.168.2.139 ba:67:11:fd:5f:fa
dnsmasq-dhcp: DHCPACK(enp1s0) 192.168.2.139 ba:67:11:fd:5f:fa
```

In this case, the CAT Board has been assigned address 192.168.2.139. Use the address you now have when doing the excercises for the following chapters.

7.4 Connecting to CAT Board with telnet

Now that you got DHCP running, you can test the connection with:

```
# telnet 192.168.2.139
Trying 192.168.2.139...
Connected to 192.168.2.139
Escape character is '^]'.

rv-139 login:
```

Login as user vsrv with password vsrv. Then you can use `su -` to become root.

Congratulations! You are now capable of logging in to the Cat Board with telnet!

7.5 CAT Board web server

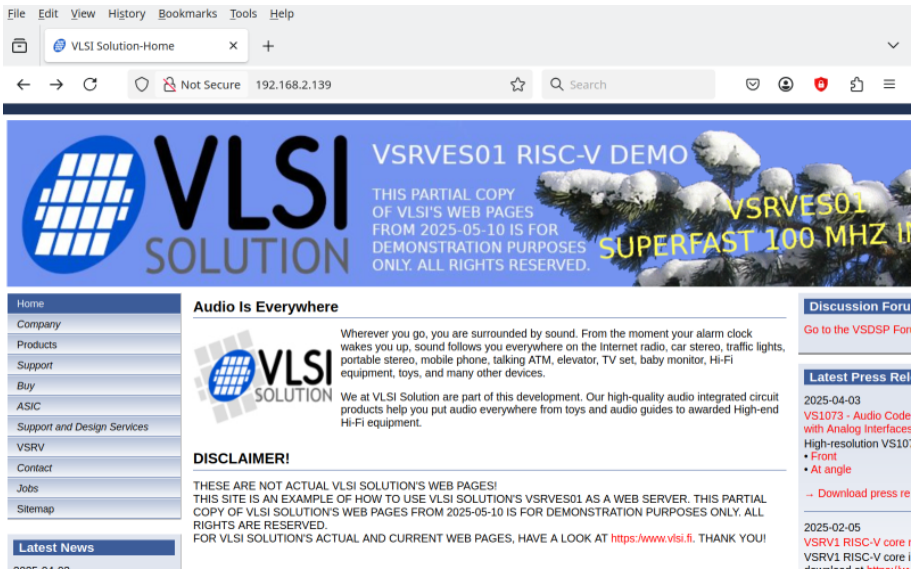

Figure 4: Cat Board web server

Now that you have an active Ethernet connection, start your favourite web browser on your computer, then open the address:
`http://192.168.2.139/`
(Note that the address should read http, not https.)
You should see a cosmetic copy of VLSI Solution's Web Pages from early May, 2025, like the one shown in figure 4.

7.6 Running VLC

After getting the Ethernet connection, install the vlc media player to your computer: `% sudo apt-get install vlc`

Then, run:

```
% vlc -v my_music/* --sout '\#transcode{vcodec=none,acodec=s16l,channels=2,samplerate=44100,scodec=none}:rtp{dst=192.168.2.139,port=5004}'
```

Note: Write the previous command as a single, long line. And be careful with typos!

Now, if you have connected Headphone Out to headphones, you should be hearing your favourite music directory.

7.7 Configuring VLC

There are two issues with VLC's default settings that need to be corrected for the best experience.

7.7.1 Fixing VLC streaming burst

An issue with VLC can be seen as soon as you start playback: the first second or two of each song is cut. This is caused by VLC streaming a large burst of data before going to normal streaming playback. You can see this issue if you copy the program ratecount.dr3, available to download from The VSDSP Forum's RISC-V section, and run it as follows:

```
S:>ratecount -i
Push 'c' to clear sample rate counter, Ctrl-C to quit.
Nominal rate     0.000 Hz, burst     0, 1 sec     0 Hz,    0.990s     0.000 Hz
Nominal rate     0.000 Hz, burst     0, 1 sec     0 Hz,    1.990s     0.000 Hz
Nominal rate     0.000 Hz, burst     0, 1 sec     0 Hz,    2.990s     0.000 Hz
Nominal rate     0.000 Hz, burst     0, 1 sec     0 Hz,    3.990s     0.000 Hz
Nominal rate     0.000 Hz, burst 33895, 1 sec 33895 Hz,    4.990s  6792.585 Hz
Nominal rate     0.000 Hz, burst 76839, 1 sec 45307 Hz,    5.990s 13222.371 Hz
Nominal rate     0.000 Hz, burst 53317, 1 sec 48444 Hz,    6.990s 18261.230 Hz
Nominal rate     0.000 Hz, burst 52551, 1 sec 50250 Hz,    7.990s 22264.831 Hz
Nominal rate     0.000 Hz, burst 51854, 1 sec 51854 Hz,    8.990s 25556.174 Hz
```

At 5.990 seconds you can see a burst of 76839 stereo samples, which is too much for our audio buffers to handle.

To fix this, open the VLC menu Tools -> Preferences -> Interface. The menu looks as shown in figure 5.

On the bottom left, change "Show settings" from "Simple" to "All", so you get a menu structure as shown in figure 6.

Now, change "Input / Codecs" -> "File caching (ms)" from 1000 to 100.

Then close VLC and restart it for the change to take effect.

Audio stuttering at the beginning of each song should now be gone, and the peak burst rate should be way lower whan starting playback of a new song, as shown below:

```
S:>ratecount -i
Nominal rate     0.000 Hz, burst     0, 1 sec     0 Hz, 1323.990s  4894.669 Hz
Nominal rate     0.000 Hz, burst     0, 1 sec     0 Hz, 1324.990s  4890.975 Hz
```

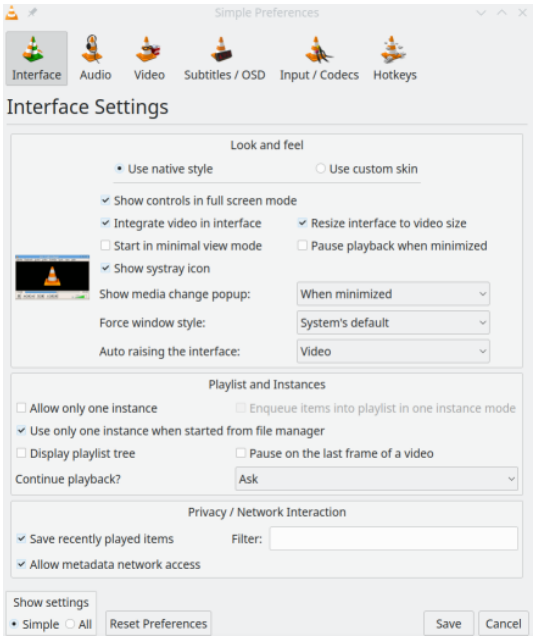

Figure 5: VLC Interface Settings, Show settings = Simple

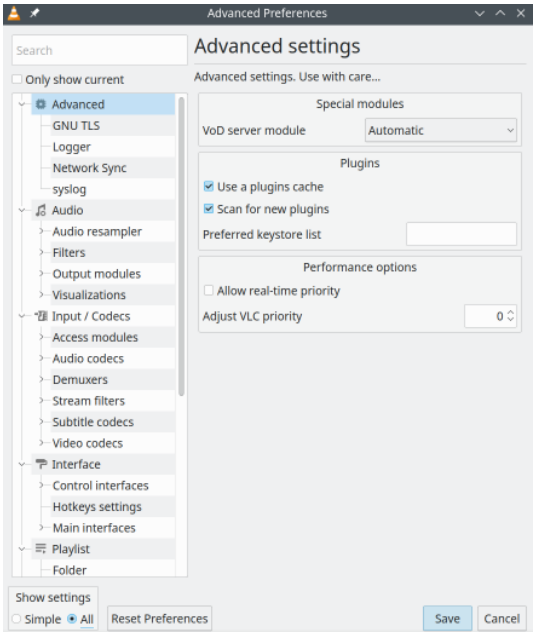

Figure 6: VLC Interface Settings, Show settings = All

```
Nominal rate      0.000 Hz, burst 40096, 1 sec 40096 Hz, 1325.990s  4917.525 Hz
Nominal rate      0.000 Hz, burst 57638, 1 sec 45108 Hz, 1326.990s  4947.812 Hz
Nominal rate      0.000 Hz, burst 50120, 1 sec 50120 Hz, 1327.990s  4981.827 Hz
```

In this example, the burst at the beginning of a song has become 20000 samples smaller, making it managable for our audio path.

7.7.2 VLC resampling

In our example, we have set our sample rate at a constant 48 kHz. However, all audio is not at this sample rate and in this case it falls upon VLC to do sample rate conversion from e.g. 44.1 kHz to 48 kHz.

Unfortunately, the protocol used doesn't allow for us to use the excellent sample rate converter of VSRV, and VLC's default sample rate converter is not very good, leading to lots of aliasing distortion, which can be heard as harshness and unwanted elements in audio. Fortunately, VLC has several sample rate converter options, some of them actually good.

To change the active sample rate converter in VLC, select menu Tools -> Preferences -> Interface, then activate extended settings menus as shown in figures 5 and 6.

The default, which appears to be libsamplerate, is not working properly, leading into aliasing distortion. So, let's get rid of it.

Select Audio -> "Audio resampler". There, set "Audio resampler" to soxr. Then select Audio -> "Audio resampler" -> "SoX Resampler". There, set "Resampling quality" to "Very high quality".

If soxr isn't available, you can try the Speex resampler by selecting Audio -> "Audio resampler", and set "Audio resampler" to "Speex resampler".

Now close VLC and restart it for the change to take effect.

7.8 Streaming audio from line input to headphone output

To stream audio from line input to headphone output, you need to enter the following instructions on the VSOS terminal command line. Do not type the portions after the # characters, they are comments telling you what the instructions do.

```
S:>driver -aux      # Remove auxplay driver from memory
S:>driver -aui      # Remove any audio input driver from memory
S:>driver +auiadc s # Add analog audio input driver, 's' = connect to stdaudioin
S:>driver +auxplay  # Add stdaudioin to stdaudioout player driver
S:>frags            # See all the drivers in memory and memory map
S:>auinput          # See what's connected to stdaudioin
S:>auoutput         # See what's connected to stdaudioout
S:>auoutput -l-20   # Change output volume to -20 dB of maximum
S:>auoutput -h      # See all the other exciting options of auoutput
```

7.9 Booting Linux through VSDSP

On the current system (2025-05-27), Linux can be rebooted with the following instruction:

```
S:>ddrload -brvlbne.bin -B -bcatboard.dtb linux61.vri
```

If you type very quickly after this: `term -a`, you will be able to see Linux boot messages. Alternatively, if you always want to see Linux boot on the VSDSP terminal, you may add (or uncomment) the line `term -a` in microSD card's file startup.txt, just before the `!Shell` command.

For more information on how to use the VSOS Shell, have a look at the document *VSRV VSOS Shell*, available at:
`https://www.vlsi.fi/fileadmin/products/vsrv/vsrv_vsos_shell.pdf`

8 Installing and updating VSOS

8.1 VSOS in short

VLSI Solution has developed a pre-emptive real-time multitasking operating system specificially designed for the VSDSP.

VSOS supports multiple devices, and FAT12/16/32 filesystems. Another important feature are dynamically loadable and unloadable executables.

The C language interface for VSOS has similar concepts for standard audio streams as normal programs have for standard input and output. Just like standard C provides stdin and stdout, VSOS also provides stdaudioin and stdaudioout file handles which make it trivial to write audio applications with a single input and output. For more diverse audio systems, the application needs to open extra audio handles by itself.

The development history of VSOS is over 10 years long. It appeared first in the VS1005 family, then VS1010 ROM with a different feature set, and VSRVES01 has again had a big set of changes. Some features are significantly different from VS1005. However, most programs can be easily ported from VS1005 to VSRVES01.

8.2 Building VSOS

Current version of VSOS isn't available as a source code distribution. It is under active development and will be released after features stop changing, and when VLSI has the toolchain ready for release, probably before end of Q3/2025.

The release of VSOS source code, when made, will be complete and free to use and modify, just as has been the case with the VS1005 family.

8.3 Flashing VSOS image to SPI flash

TBD

8.4 Adding required programs to SD card

The VSOS system root files are available at the VSDSP Forum:
`http://www.vsdsp-forum.com/phpbb/viewtopic.php?t=3242`

9 Developing VSDSP programs

TBD when VSIDE update for VSRV is released before end of Q3/2025.

9.1 Preparing VSIDE to work with VSRV

TBD

9.2 Developing a program

TBD

10 Building Toolchain for RISC-V side of VSRV

First you need to fetch repository, official RISC-V toolchain is just fine for that purpose. Bear in mind that this step requires roughly 20GB of drive space, most of it can be reclaimed after building if needed but doing so will make any later rebuild longer.

```
git checkout -depth 1 https://github.com/riscv/riscv-gnu-toolchain
```

Then go to that directory, configure and build it. Of course `/local/new-toolchain` path has to be adjusted to suit your system needs. This step will take a bit of time, depends on machine may be also few hours.

Important parts in following configure are obviously architecture setting part, since VSRV is 32bit RiscV with two extensions `ma`, because `i` in fact selects full version of RiscV, as opposed to 'microcontroler' flavour with reduced number of registers denoted by `e` letter instead of `i`.

Extension `m` is a Standard Extension for Integer Multiplication and Division, and `a` is a Standard Extension for Atomic Instructions.

Then after underscore we have `zicsr` which denotes Control Status Register (CSR) Instructions, necessary for accessing hart's register that for example control exceptions and interrupts masking and delegation.

Now we select code model (cmodel) to be medlow, which is default for gcc so technically we could not specify it but if upcoming gcc release changes that having it specified ensures we will get compatibile binaries.

Last parameter selects floating point implementation, in our case there is no hardware support for FP therefore we select 32bit soft-float denoted by ilp32.

```
cd riscv-gnu-toolchain

./configure \
--prefix=/local/new-toolchain \
        --with-arch=rv32ima_zicsr \
        --enable-linux \
        --with-cmodel=medlow \
        --with-abi=ilp32

make -j `nproc` linux
```

So next step is to add new toolchain to PATH, like

```
export PATH=$PATH:/local/new-toolchain/bin
```

You of course above can be added to profile script for shell of your choice. Also please notice that path added to PATH variable has extra /bin on end of it in comparision to one

which we passed as `prefix` to configuration, it is because prefix expects top level directory where all toolchain files will reside, and compiler binaries will land in bin directory relative to that path.

Now, we have a compiler, lets verify we set up all properly and it can compile very simple program. First we need test code like

```
#include <stdio.h>

int main() {
    printf("Hello RiscV\n");
    return 0;
}
```

Then we can compile it, twice to make sure both static and dynamic linking works.

```
riscv32-unknown-linux-gnu-gcc -o hello.dyn hello.c

riscv32-unknown-linux-gnu-gcc -o hello hello.c -static
```

Now we can check if they are as we want, dynamically linked and statically linked binaries.

```
file hello.dyn
```

hello.dyn: ELF 32-bit LSB executable, UCB RISC-V, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux-riscv32-ilp32.so.1, for GNU/Linux 5.4.0, not stripped

`file hello` hello: ELF 32-bit LSB executable, UCB RISC-V, version 1 (GNU/Linux), statically linked, for GNU/Linux 5.4.0, not stripped

11 Building Linux for RISC-V side of VSRV

Following sections assume little bit of Linux experience on reader side, order in which they appear is dictated by order necessary to do a full rebuild. However, it is perfectly possible to execute them in different sequence or to skip some of them if task requires only for example getting some extra software to be compiled and added to existing image.

Host system needs about 2GB of space to compile the kernel itself. Also we assume that toolchain mentioned in Chapter 10 is reachable by $PATH.

11.1 Getting sources

11.1.1 Linux Kernel

We can get tarballs from https://kernel.org/ or fetch fresh sources directly from git at https://github.com/torvalds/linux.

For simplicty and reproducible results this guide assumes tarball. We also pick latest stable kernel from 6.1 line at time of writing of this guide (2025-06-19).

```
$ xz -d < linux-6.1.141.tar.xz | tar xv
```

There is only one manual change in the kernel that is currently needed. Edit `drivers/clocksource/timer-riscv.c` around line 88 and change the value 100 in clockevents_config_and_register to about 1000.

Now it is time to copy the template kernel config offered by VLSI Solution to the newly unpacked kernel sources.

```
$ cp config-6.1.txt linux-6.1.141/.config
```

11.1.2 VLSI Drivers

TBD

11.2 Configuring and building Linux Kernel

For first build it may be safe to not modify the values provided by the template config and go with build as we need compiled kernel to build drivers against it.

So now to the build, there may be a few questions if your kernel as in the following example is more recent than one used to prepare config file (6.1.0), but it is usually safe

to assume that default values offered for those are safe. We can alter that anytime later too.

In this step we also install modules, so we can update the initial ramdisk as well.

```
$ cd ~/linux-6.1.141
$ touch catboard.cpio
$ CROSS_COMPILE=riscv32-unknown-linux-gnu- ARCH=riscv \
    make modules Image
### now install them into some temporary directory
$ CROSS_COMPILE=riscv32-unknown-linux-gnu- ARCH=riscv \
    make INSTALL_MOD_PATH=/tmp/rv-krnl modules_install
```

Now we have all necessary bits ready to compile out-of-tree drivers against this kernel.

```
$ cd ~/vlsi-lnx-drv
$ CROSS_COMPILE=riscv32-unknown-linux-gnu- \
    KSRC=/path/to/linux-6.1.141 make
```

The step above should produce the driver built against kernel specified by the KSRC variable.

Now we need to put cpio archive with CAT Board setup, finish building and modules installation, then rebuild the cpio archive and finally relink again the kernel.

```
$ cp catboard.cpio linux-6.1.141/catboard.cpio
```

If there is a need, we can run again menuconfig as in the example below, alter what modules are built, then install modules again. It is safe to skip this step if not needed.

```
$ CROSS_COMPILE=riscv32-unknown-linux-gnu- ARCH=riscv \
    INSTALL_MOD_PATH=/tmp/rv-krnl \
    make menuconfig Image modules modules_install
```

11.3 Initial Ramdisk

Kernel needs some programs to execute, for VSRVES01 it was chosen to utilize only RAM filesystem which is initialized by the kernel itself from a cpio archive provided during kernel booting. It is not most convenient for development, because each change in initial RAM disk requires to relink kernel, but it allows for a very simple and fast booting process.

Now let's assume we have a file 'foobar' we want to add to the root directory. But before doing anything else, let's ensure we have copy of the current cpio file before we modify it.

```
$ cp catboard.cpio catboard_old.cpio
```

Now we need to do a few things, first is to put updated modules for new kernel version as original cpio contains them for old kernel, also we need to add our hello program we produced at toolchain preparation.

But first things first, lets check what modules are inside initrd so we update the right ones.

```
$ cpio -t -F catboard.cpio|grep "lib/modules/"
```

So now we need mii.ko which would be part of linux kernel, we can find it with

```
$ find /tmp/rv-krnl -type f -iname "mii.ko"
```

and other one will be the driver we compiled just before.

We can take the existing archive and append files to it for a test as it is the simplest way to just update file or two. Of course it will make the file grow bigger each time we do that, so it might be handy to start from a fresh original copy if that is a problem.

```
$ mkdir -p initrd-addon/root initrd-addon/lib/modules
$ cp ~/hello initrd-addon/root/hello
$ cp ~/vlsi-lnx-drv/vlsi-mac.ko initrd-addon/lib/modules/
### grab updated modules from temporary install directory
$ cp /tmp/rv-krnl/lib/modules/6.1.141/kernel/drivers/net/mii.ko \
    initrd-addon/lib/modules/
### at this step you can copy any other files or modules into initrd-addon
### with the same directory structure as you want them to be at on target.
$ (cd initrd-addon; find -type f | cpio -o -H newc -O ../catboard.cpio -vA)
```

Now we have an updated catboard.cpio, adding our extra file plus newly rebuild modules for the new kernel. Let's check if it does contain the extra files.

```
$ cpio -t -F catboard.cpio|tail
```

Now we can execute again the command to build kernel as described before, and we only need to run 'Image' target.

11.4 Preparing a VRI image

As we finished with kernel compilation we need to convert it to VLSI Solution's proprietary VRI format to utilize fast loading provided by DDRLoad.

```
$ elf2vri +v -a 0x80400000 -B arch/riscv/boot/Image - newlinux.vri
```

And of course above command needs to be executed in linux-6.1.141 directory, or modified to reflect actual path to Image. Next step is to copy newlinux.vri to sdcard and we can test booting.

Start address (0x80400000) is where loader code contained in rvlbne.bin expect kernel image to land. Currently there is no way to adjust that short of editing loader code and rebuilding. However, this may change.

Now we can test the newly built image by typing or copying following command to the VSOS Shell prompt.

```
S:>ddrload -brvlbne.bin -B -bcatboard.dtb newlinux.vri
```

In the above command first parameter causes riscv bootloader to be loaded at default address which is 0x80000000, then -B makes ddrload to append just after it its parameter data which carries mac address, clock and memory configuration, then it appends dtb file, finally followed by vri which will be loaded at the address we defined on its creation.

12 The VRI Image File Format

The VRI (Vlsi Risc-v Image) image file format is optimized for making it possible to load files very quickly from VSDSP/VSOS to DDR memory.

The file begins with a 4-byte magic header “VRI1”, followed by zero or more sections:

```
"VRI1"
Section0
...
SectionN-1
```

Sections can be of two types, either a literal run section, or a RLE zero run section:

```
Section, literal run:
    56 78 12 34  ADDR: Address (0x12345678), must be divisable by 4
    56 78 12 34  SIZE: Literal run size in bytes, must be divisable by 4
    00 0? 00 00  FLAGS: FL_READ 1, FL_WRITE 2, FL_EXECUTE 4, no FL_BSS 8
    00 00 00 00  EXTENSION: 0
  n*56 78 12 34  Literal data, n=SIZE/4, file bytes: 0x78 0x56 0x34 0x12

Section, RLE zero run:
    56 78 12 34  ADDR: Address (0x12345678), must be divisable by 4
    56 78 12 34  SIZE: RLE zero run size in bytes, must be divisable by 4
    00 0? 00 00  FLAGS: FL_READ 1, FL_WRITE 2, FL_EXECUTE 4, set FL_BSS 8
    00 00 00 00  EXTENSION: 0
```

Below is an example of a text file consisting of 13 characters “Hello, world!” followed by 551 zeroes, and how it will look after encoding it into a VRI file.

```
% hd helo.txt
00000000  48 65 6c 6c 6f 2c 20 77  6f 72 6c 64 21 00 00 00  |Hello, world!...|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000234
% elf2vri +v -a 0x80400000 -B helo.txt - helo.vri
RLE/BSS/strip compressed 0 bytes to 4 (0.0% compression)
RAW file at 80400000, helo.txt
    RLE: a=80400000, lit     10, bss    224
RAW compressed 564 bytes to 48 (91.5% compression)
% hd helo.vri
                     ---ADDR----  ---SIZE---- ---FLAGS---
00000000  56 52 49 31*00 00 80 40  00 10 00 00 00 07 00 00  |VRI1...@........|
          -EXTENSION- -LITERAL-RUN-DATA-LITERAL-RUN-DATA-L
00000010  00 00 00 00 65 48 6c 6c  2c 6f 77 20 72 6f 64 6c  |....eHll,ow rodl|
          ITERAL-RUN- ---ADDR----  ---SIZE---- ---FLAGS---
00000020  00 21 00 00*00 10 80 40  02 24 00 00 00 0f 00 00  |.!.....@.$......|
          -EXTENSION-
00000030  00 00 00 00                                       |....|
00000034
```

Sections are independent of each other so new sections can be appended to an existing image with no issues.

All 32-bit values are encoded in the mixed-endian way shown above. Note that for literal runs, the byte order is changed. So, a text file reading “HELO” would become “EHOL” in the VRI file.

Literal and RLE zero runs are recognized by the FL_BSS bit: if FL_BSS is set the section is of type RLE zero run. If FL_BSS is clear then the section is a literal run.

The VRI format may be freely used for anything, but please don't break it!

13 Latest Document Version Changes

This chapter describes the latest changes to this document.

Version 0.10, 2025-06-19

- Added necessary information to compile the Linux toolchain to Chapter 10. *Building Toolchain for RISC-V side of VSRV*.
- Added information on how to compile your own Linux to Chapter 11, *Building Linux for RISC-V side of VSRV*.
- Added new Chapter 12, *The VRI Image File Format*.
- Added reference to all-new *VSRV VSOS Audio* document to Chapter 6.2, *Getting to VSOS shell*.

Version 0.03, 2025-05-30

Added reference to all-new *VSRV VSOS Shell* document to Chapter 6.2, *Getting to VSOS shell*.

Version 0.02, 2025-05-30

Cleanup of existing text.

Version 0.01, 2025-05-28

First published preliminary version.

14 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