Lab Report

Starfive's VisionFive 2 and HPC

A Setup Guide

Jonas Hafermas

MatNr: 21862404

Supervisor: Julian Rüger

Georg-August-Universität Göttingen
Institute of Computer Science, GWDG

August 14, 2024

Abstract

In this lab report, we set up the VisionFive 2 Single Board Computer (SBC) to prepare it for further investigation. The firmware is updated, the operating system is configured, a custom kernel is compiled and booted, challenges will be discussed and a brief outlook for further work is provided.

Contents

List of Figures

List of Abbreviations

HPC	High-Performance Computing
RISC	Reduced Instruction Set Computer
ISA	Instruction Set Architecture
SBC	Single Board Computer
SoC	System on Chip
TFTP	Trivial File Transfer Protocol
QSPI	Quad Serial Peripheral Interface
openSBI	(RISC-V) Open Source Supervisor Binary Interface
SD	Secure Digital (Card)
eMMC	Embedded Multimedia Card
NVMe	Non-Volatile Memory - Express (Protocol)
VDI	Virtual Disk Image

1 Introduction

While the x86-architecture has for a long time played the most important role in the design of High-Performance Computing (HPC) systems, alternatives are gradually beginning to surface. One such example is Reduced Instruction Set Computer (RISC)-V, an open-source Instruction Set Architecture (ISA) initially developed at the University of California, Berkeley and implementing the *reduced instruction set* design principle, which is especially interesting due to its high architectural flexibility and low power consumption, both of which are of great importance in HPC applications. Further information for the curious can be found on the official website. Until a few years ago, the lack of HPC-viable RISC-V hardware limited more technical research attempts to a handful of boards with few, low-powered cores. One SBC that shows potential for testing toy HPC workloads is the VisionFive 2 by Chinese company Starfive which is built around their improved JH7110 64-bit quad-core System on Chip (SoC). In this report, guidance will be provided on how to prepare the VisionFive 2 for further investigation with regards to its HPC viability. This includes updating the firmware, installing a current operating system and experimenting with a custom Linux kernel to hopefully improve functionality. Finally, the first impressions of working with the VisionFive 2 will be discussed and an outlook at further research will be provided.

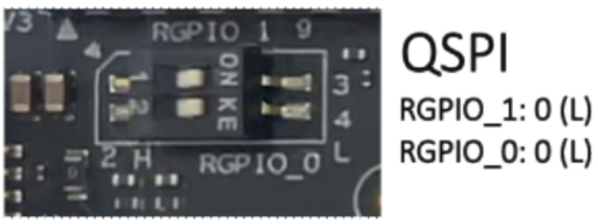
2 Updating the Firmware

Note: All images in this section were taken from Starfives official quickstart guide, which also holds more information as well as pointers towards advanced topics not covered in this document.

Our board revision is v1.3B, which shows no discernible differences to v1.2A except for the second LAN port now too operating at 1 GBit/s[1]. The units came shipped with 8 GB of RAM installed and have not been tampered with. Unfortunately, the factory default settings only support a minimal Debian 12 snapshot based on the obsolete Linux kernel version 5.15.0 with limited functionality, thus requiring an update of the firmware and the bootloader before being able to load more recent images. The manufacturer suggests using a Trivial File Transfer Protocol (TFTP) server to upgrade the firmware which is comparatively cumbersome to set up, so another, more practical way to transfer the files using the `YMODEM` protocol directly from the host system without having to use TFTP at all will be lined out[2].

First, we want to install `lrzsz` and `minicom` using the built-in package manager of our respective linux distribution. `lrzsz` is needed to enable `YMODEM` functionality for `minicom`, a friendly serial communication program which we will use to indeed communicate with the SBC over a serial connection we will set up shortly. In `minicom` (initial launch may need `-s` option, see manpage), we set the baud rate to 115200 bps and the bit parity to 8N1. Furthermore, we need to determine the serial port we wish to use for communication (for example `/dev/ttyUSB0`).

Next, we prepare the files we want to flash. Usually, when getting a release from Starfives Debian webpage or the release repository, the files in question will be called `u-boot-spl.bin.normal.out` and `visionfive2_fw_payload.img`. It is generally advisable to use the newest version, though sometimes during testing, the firmware version had to exactly match the Debian release version it belonged to so conflicts cannot be ruled out entirely. After the files have been procured and saved to an easy to reach location, we boot the SBC in QSPI flash mode without an SD card inserted.



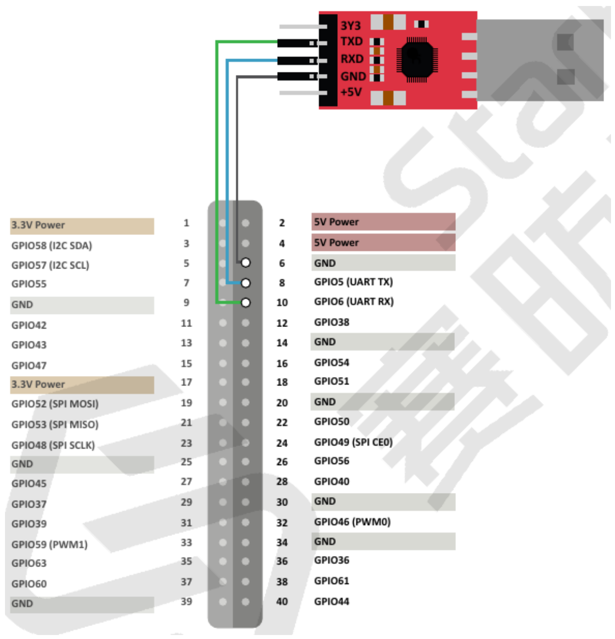
Source: `https://doc-en.rvspace.org/VisionFive2/Quick_Start_Guide/`

Figure 1: QSPI Flash Boot Mode Setting

[1]`https://doc-en.rvspace.org/VisionFive2/Quick_Start_Guide/VisionFive_2/specification_pb.html`

[2]`https://forum.rvspace.org/t/guide-upgrading-firmware-with-only-a-serial-uart-connection-via-u-boot/1589`

We need to connect the serial adapter to the VisionFive 2 as is shown below. If chaos ensues, try switching the `TX` and `RX` cables[3].



Source: `https://doc-en.rvspace.org/VisionFive2/Quick_Start_Guide/`

Figure 2: GPIO Pinout

Power on the SBC and launch minicom. If everything is set up correctly, we should see (RISC-V) Open Source Supervisor Binary Interface (openSBI) starting up and the u-boot autoboot being interrupted automatically (since no boot drive is present).

```
Hit any key to stop autoboot: 0
StarFive #
```

Now that we're in u-boot, we can start the flashing process. These first steps are identical for both files. We use the `loady` command to select one of the two files via the minicom prompt.

```
StarFive # loady
## Ready for binary (ymodem) download to 0xA0000000 at 115200 bps...
CCxyzModem - CRC mode, 1(SOH)/128(STX)/0(CAN) packets, 7 retries
## Total Size      = 0x0001fe80 = 130688 Bytes
StarFive #
```

Since the firmware is sensible data, we want to ensure its integrity. We do this by running `crc32` on the file and comparing it against a checksum calculated locally (for example using `cksum`).

```
StarFive # crc32 $loadaddr $filesize
crc32 for a0000000 ... a001fe7f ==> ca14da8e
```

If the sums match, we most likely have an uncorrupted file and can proceed.

[3] `https://jamesachambers.com/starfive-visionfive-2-firmware-update-guide/`

First, we have to call and detect the SPI Flash device.

```
StarFive # sf probe
SF: Detected gd25lq128 with page size 256 Bytes, erase size 4 KiB, total 16 MiB
```

Then, we can finally start flashing. It is important to know that the two files **require different loading address offsets**. For `u-boot-spl.bin.normal.out`, the offset is **zero**.

```
StarFive # sf update $loadaddr 0x0 $filesize
device 0 offset 0x0, size 0x1fe80
130688 bytes written, 0 bytes skipped in 0.600s, speed 221563 B/s
```

For `visionfive2_fw_payload.img`, the right offset is **0x100000**.

```
StarFive # sf update $loadaddr 0x100000 $filesize
device 0 offset 0x100000, size 0x2aae85
867973 bytes written, 1929216 bytes skipped in 5.750s, speed 497796 B/s
```

We perform one of the two operations above depending on the file we have loaded before. After the process has finished without any apparent errors, we repeat the steps with the other file. Once both files have been flashed to memory successfully, we can reboot the system by either using the `reset` command or by pressing the reset button on the VisionFive 2. If we end up in the serial console again, then congratulations, the firmware has been flashed!

3 Setting up Debian

Starfive is hosting minimal Debian images in the cloud; at the time of writing, Debian images using kernel version `5.15.0` as well as `6.6.20` were available. In addition, several storage device types are supported, most prominently Secure Digital (Card) (SD) as well as Embedded Multimedia Card (eMMC), but also Non-Volatile Memory - Express (Protocol) (NVMe). We went with the newer kernel version which is actually not that far behind the most recent LTS kernel (6.6.41 at the time of writing) and should thus be decently usable out-of-the-box. Our proof-of-concept system used an SD installation for ease of access. balenaEtcher was used to flash the SD image for the same reason.

Upon startup, the system is reasonably responsive and even supports high display refresh rates and ultrawide configurations if available. However, light graphics tests like launching Google Maps and searching through the local area's cartography showed disappointing 3D performance and no load spikes on the GPU, indicating issues with rendering task delegation which might be ironed out with future patches. The Debian version of the image is described as *bookworm sid*, indicating an unstable snaphot pre-Debian 12 (which is codenamed *bookworm*). We tried to upgrade the Debian version by selecting the newest snapshot with decent software availability from the `debian-ports` snapshot archive, the newest meaningful upgrade being from July 2023. Following standard Debian upgrade procedures yielded no issues and the system was successfully upgraded to Debian 12 unstable, codenamed *trixie sid* (*trixie* being the codename for Debian 13).

```
user@starfive:~$ cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux trixie/sid"
NAME="Debian GNU/Linux"
VERSION_CODENAME=trixie
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
```

It is important, however, to reinstall or hold back custom packages provided by Starfive since they would be overwritten by the upgrade.

Seeing that Debian runs reasonably well using the precompiled images, we went ahead and attempted to build a newer Debian image from scratch. To ensure proper installation of proprietary/custom software, this paragraph follows StarFive's own guide on building Debian images for the VisionFive 2 from source. We undertook this task using a VirtualBox Debian 12.6 environment running kernel version `6.1.0`. Conveniently packaged Virtual Disk Image (VDI)s are available from sources like `osboxes.org`. Initial test results show that much of the kernel configuration and proprietary software installation can be automated with a simple bash script (appended as `build_deb.sh`) while Debian configuration can cause more or less complex software dependency issues when using a newer Debian snapshot, thus requiring individual handling. Two things should be noted throughout the installation process: When choosing a development branch to fetch the kernel from, note that the standard choice (`JH7110_VisionFive2_devel`) uses version 5.15.0 instead of 6.6.20. Instead, use `JH7110_VisionFive2_6.6.y_devel` to get the newer kernel version. Secondly, by default, the build script sets an older Debian snapshot as package source, but recommends a newer version as well as allowing the user to input a custom URL.

On booting up the self-built image, in its current version the root partition does not take up all of the unallocated space (only around 3 GB) and should thus be enlarged using `parted` or adjacent software (do not forget to run `resize2fs` on the partition afterwards). Since the custom images software stack is decidedly minimal, we install some prerequisites for building a kernel image (*see section 4*) as well as additional custom software provided by Starfive via a small script.

```bash
#!/bin/bash

# just a simple script fetching the latest software for the VisionFive 2 SBC
# provided by Starfive
# check if the linux image decompressed correctly (no unallocated space),
# otherwise storage will not suffice!

# get prerequisites for setup and custom kernel installation
sudo apt-get install -y bc binutils bison curl dwarves flex gcc git gnupg2 gzip
libelf-dev libncurses5-dev libssl-dev make openssl pahole perl-base rsync tar
xz-utils

# get redirect and download latest install script
RE_URL=$(curl -sL -o /dev/null -w "%{url_effective}" https://github.com/starfive-
tech/Debian/releases/latest | grep -oE '[^/]+$')

curl -L -o ./install_package_and_dependencies.sh https://github.com/starfive-
tech/Debian/releases/download/${RE_URL}/install_package_and_dependencies.sh

# let Starfive work their magic
sudo bash install_package_and_dependencies.sh
```

A final word on alternative image files: Pre-compiled eMMC images work fine when run in QSPI boot mode (*see section 1*), however the dedicated eMMC option sometimes fails to boot. Starfive is aware of the issue but as of July 2024, it has not beeen fixed. However, simply booting in QSPI mode seems to mitigate the issue. eMMC images can be built from source without much apparent hassle, as is described in the build guide. It was out-of-scope for this report, however.

4 Experimental Linux Kernels

After confirming system operability both on SD and eMMC devices, we finally attempted to build a newer kernel from scratch and getting it to run on the SBC. Starfive has completed patchwork and is now awaiting upstream integration approval[4] for linux kernel version 6.9, with prepatched linux kernels already floating around the internet. We will use yuzibo's prepatched vf2 kernel (version 6.9.8) for this experiment. There are a lot of guides on how to build the Linux kernel from source, but this section took inspiration from a nicely written article on https://itsfoss.com/compile-linux-kernel/. Following up on section , no extra software should be needed at this point. We first want to fetch our kernel files by cloning yuzibo's repository. We also want to make sure that we are on the right branch for version 6.9.8.

```
user@starfive:~$ git clone https://github.com/yuzibo/vf2-linux.git && git checkout
vf2-v6.9.8-dev
```

Afterwards, we follow the instructions given in the article above (skipping the checksum verification since we cloned the repository directly) until the actual kernel installation, where we have to fix an error in our current Debian image version, more precisely an issue with escaped variables within `zz-initramfs-mod` which blocked proper initrd image setup. We can quickly and elegantly correct this using `sed`.

```
user@starfive:~/vf2-linux$ sudo sed 's/\\\(\$\w\+\)/\1/g' /etc/kernel/postinst.d/
zz-initramfs-mod
```

Lastly, after installation has finished successfully, we have to update the u-boot bootloader to properly accomodate the new kernel at startup.

```
user@starfive:~/vf2-linux$ sudo u-boot-update
P: Checking for EXTLINUX directory... found.
P: Writing config for vmlinuz-6.9.8-vf2-custom...
P: Writing config for vmlinuz-6.6.20-starfive...
P: Updating /boot/extlinux/extlinux.conf...
```

After rebooting, we can see that indeed, we are now running an updated and patched kernel!

```
user@starfive:~$ uname -r
6.9.8-vf2-custom
```

Further testing was outside the scope of this report, but one possible approach could be to install a benchmark tool like the phoronix test suite for system evaluation and performance testing (for example using `gromacs`).

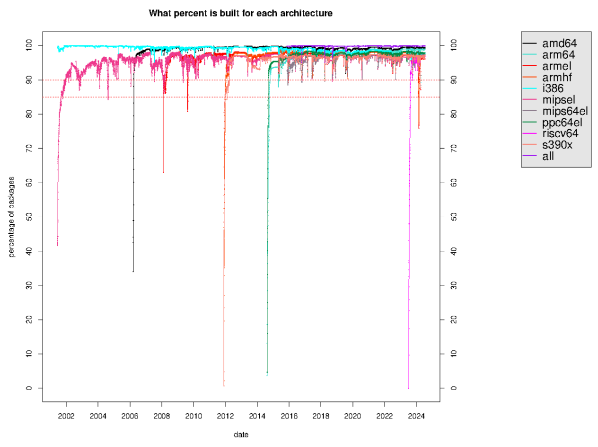
[4] `https://rvspace.org/en/project/JH7110_Upstream_Plan`

5 Challenges and Outlook

Just because this report is nice and short does of course not mean that challenges were not encountered.

Firstly, the process of updating the firmware (which is very much mandatory due to the limited functionality of Debian images available pre-flash) is an arduous and unnecessarily complex task which elevates the barrier of entry for users without experience in embedded systems development (which also includes the author). The user-devised solution presented within this report mitigates some of the issues arising from following the official procedure, but flashing shouldn't have to be done via serial console and instead come configured with the minimal linux distribution provided alongside the hardware, as is the case with for example the Raspberry Pi series of ARM-based SBCs.
Furthermore, the kernels provided by Starfive rely heavily on built-in functionality, with few modules loaded at runtime, which is desirable for embedded systems but generally unfavourable for HPC configurations. Of course, the VisionFive 2 is not explicitly advertised as a system for HPC deployment but neither should it be considered a barebones SoC powering a coffee machine. Often cited as a direct competitor to Raspberry Pi SBCs[5], it seems to lack the refined software ecosystem available on ARM systems as of right now.

Because in principle, the VisionFive 2 arrives at a time when great strides are being made within the RISC-V community. This is supported by `riscv64` becoming an official Debian architecture[6] with out-of-the-box support set to ship with Debian 13[7].



Source: `https://buildd.debian.org/stats/`

Figure 3: Percentage of source packages built for each official Debian architecture

In their recent paper, Nick Brown et al outline that the performance improvements over several generations of RISC-V hardware (culminating in the VisionFive 2) are considerable and that large-scale integration of RISC-V CPUs is already underway [Bro24], with another relevant example being the server solutions provided by Esperanto AI.

[5]`https://www.heise.de/tests/Einplatinencomputer-im-Test-StarFive-VisionFive-2-mit-RISC-V-Chip-7473680.html?seite=all`
[6]`https://lists.debian.org/debian-riscv/2023/07/msg00053.html`
[7]`https://lists.debian.org/debian-devel-announce/2023/06/msg00001.html`

Those scalable systems in particular should offer plenty of new research opportunities beyond what is achievable with SBCs, particularly into the various options an open instruction set could bring to the table for synchronisation and parallelisation efforts as well as customised instruction pipelining and increased hardware adaptability for HPC tasks overall.

References

[Bro24] Nick Brown. *RISC-V for HPC: Where we are and where we need to go*. 2024. arXiv: 2406.12398 [id='cs.DC' full_name='Distributed, Parallel, and Cluster Computing' is_active=True alt_name=None in_archive='cs' is_general=False description='Covers fault-tolerance, distributed algorithms, stability, parallel computation, and cluster computing. Roughly includes material in ACM Subject Classes C.1.2, C.1.4, C.2.4, D.1.3, D.4.5, D.4.7, E.1.'].