decode

Integration and deployment of the DECODE OS and HUB platform

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 732546

Project no. 732546

DECODE

DEcentralised Citizens Owned Data Ecosystem

D4.6 Integration and deployment of the DECODE OS and HUB platform

Version Number: V1.0

Lead beneficiary: Dyne.org

Due Date: June 30th

Author(s): Denis Roio, Ivan Jelin (Dyne.org)

Editors and reviewers: Guy Samuel, Jordi Coscolla (Thoughtworks), Alberto Sonnino (UCL)

Dissemination level:		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Approved by: Francesca Bria (Chief Technology and Digital Innovation Officer, Barcelona City Hall)
Date: 02/07/2018

This report is currently awaiting approval from the EC and cannot be not considered to be a final version.

Table of Contents

1 Introduction

This brief technical deliverable illustrates aspects related to the integration and deployment of the DECODE OS (D4.4) with the DECODE HUB hardware platform according to benchmarks (D4.5) and the indicated reference prototype (D4.8).

This deliverable configures itself also as an operational manual about the following topics:

1. Specifications of the HUB hardware adopted
2. Procedures to build the OS for the HUB
3. Instructions to deploy the HUB running the OS
4. Overview of networking access and operations

This document provides pilot partners in DECODE the basic knowledge to run a DECODE HUB node on their premises, assessing the impact of such a deployment within their infrastructure. As the piloting phase unfolds towards D4.10 “DECODE HUB devices piloted in trials” this document will be extended considering the security and liability implications emerged.

The latest version of this deliverable is made available to the public as part of the documentation of DECODE OS and attainable from its website on.

2 Hardware specification

According to findings outlined in D4.8 by project partner Arduino, we have selected an embedded low-consumption board for its specific characteristics conforming our requirements. This choice has been made despite certain shortcomings on performance shown in D4.5 benchmarks that may be regarded as negligible in this piloting phase.

In brief, these characteristics are: - Availability of a SATA interface for storage solutions beyond SDCards - Free from any proprietary firmware blob for boot (100% free eligible) - Capable of running mainline Linux kernel without patches - Assembled in Europe and certified as Open Hardware.

DECODE HUB nodes (elsewhere indicated as "nano-nodes") are then made using the Olimex Lime2 hardware boards (based on a ARM20 processor) and connected using a gigabit Ethernet port. A LiPo battery provides stability in case of short power cuts and a noise immune power supply provides the necessary 5v energy to function.

The boards are assembled in Europe and are certified Open Hardware: they do not require any proprietary blob to run. It is therefore possible to assume that any software function running on the boards are not conditioned by the hardware manufacturer, clearing issues of liability and integrity of computations.

2.1 Manufacturer's documentation

The manufacturer provides extensive documentation about these boards:

- The Olimex Lime2 user manual
- The full A20 hardware datasheet

For a quick overview of the hardware here is included the block diagram found in the A20 datasheet (AllWinner chip):

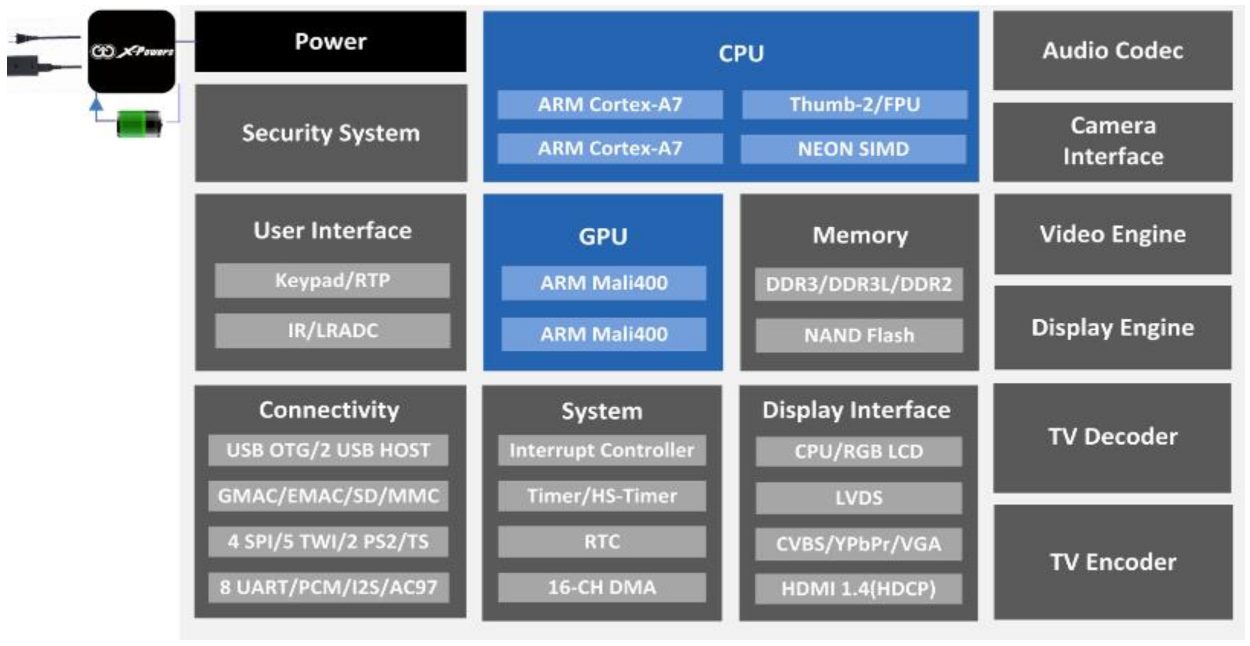

Figure 1

An illustrated image of the Lime2 board assemblage with indicated components is also reported here from its user manual. Here the front:

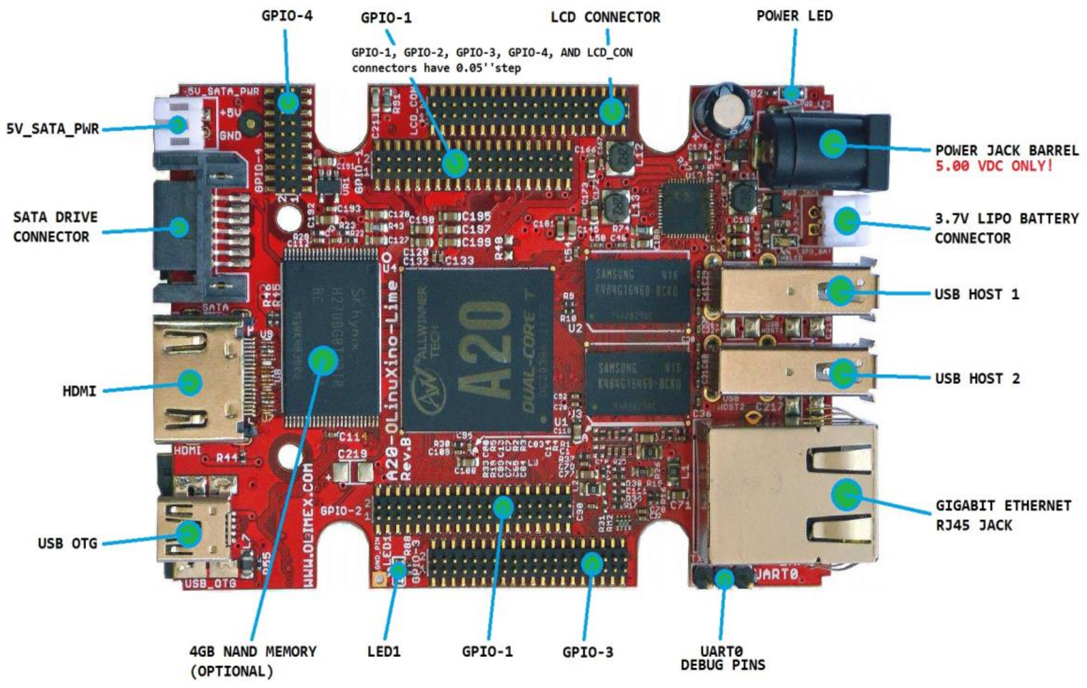
GPIO-4
GPIO-1
LCD CONNECTOR
POWER LED
GPIO-1, GPIO-2, GPIO-3, GPIO-4, AND LCD_CON connectors have 0.05''step
5V_SATA_PWR
SATA DRIVE CONNECTOR
HDMI
USB OTG
POWER JACK BARREL 5.00 VDC ONLY!
3.7V LIPO BATTERY CONNECTOR
USB HOST 1
USB HOST 2
GIGABIT ETHERNET RJ45 JACK
4GB NAND MEMORY (OPTIONAL)
LED1
GPIO-1
GPIO-3
UART0 DEBUG PINS

Figure 2

And here below the rear:

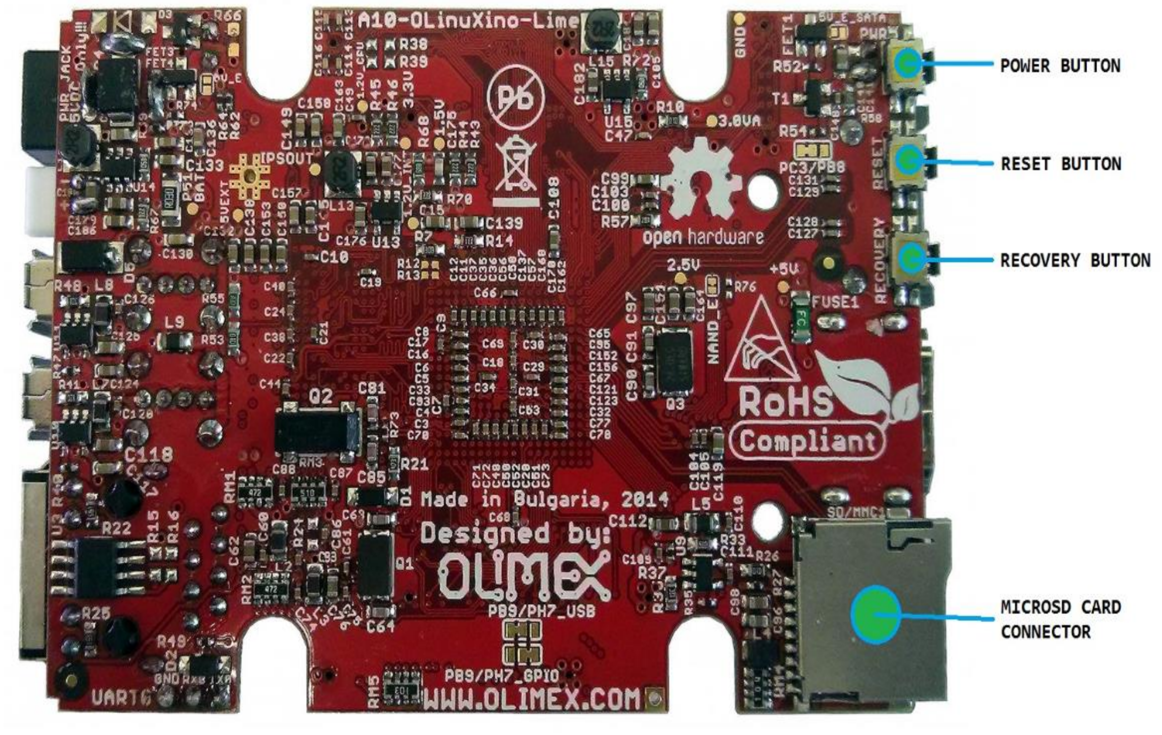

Figure 3

In the following chapter is provided information on how to build the DECODE OS system to operate on this board.

3 DECODE OS builds and hardware integration

The DECODE operating system is built in a number of stages: this process is detailed in D4.4 and will be briefly documented here.
This section outlines how to build DECODE OS for our DECODE HUB hardware target architecture, which is the hard-float ARM (armhf). For the purpose we adopted the Devuan Ascii (2.0.0) armhf port.

Each build stages illustrated below is independent from each other and can be triggered on its own; however the final stage requires the results from the previous ones to be able to integrate them together.

3.1 Userspace

The first stage built is the actual userspace, where the system is bootstraped from scratch, and packages and configurations are added on top of it. This is the step most extensively described in D4.4. The tools we use to build our userspace are made available in the git repository. These tools are a higher-level abstraction on top of Devuan's Arm-SDK, which is the way ARM-based images are created in Devuan. The port of Devuan to ARM has gathered an active community that can be also reached on the Freenode IRC channel #devuan-arm.
The process of building via the Devuan SDK is abstracted using a concept called *blends*: it offers us a more high-level approach to configuring a base system after it has been bootstrapped. Using this *blend*, we are able to easily and reproducibly install various software - from the init system (OpenRC), to the very final userspace tools needed for DECODE like Chainspace and the JVM, tor-dam and Golang for more applications.

Along with installing software, the blends allow us to provide file overlays, which enable us to have a directory structure that resembles the bootstrapped system, and use the overlay to reconfigure any existing files. Finally, it also gives us a space where we can mark certain binaries with needed flags that are used in the kernelspace.

3.2 Kernelspace

The ARM kernelspace consists of mainline Linux (official kernel.org) releases (version 4.9.y) which in the special case of DECODE OS requirements for integrity and security of the system we patch with the unofficial grsecurity patch that is maintained for this very kernel version. It is maintained at and (unlike x86) does not suffer from the recent Meltdown and Spectre attacks (CVE-2017-5715, CVE-2017-5753, and CVE-2017-5754) while this does not apply for *all* ARM processors.
The tools used for building this stage are packed into a single git repository which is a suite of makefiles that contain deterministic rules for building the Linux kernel, the *initramfs*, and finally packs the results into a tarball, and a ready-to-flash raw image - both of which can be used and put onto a microSD card to be used with the Olimex OLinuXino LIME2. This suite can be found at the DECODE project's Github:

Roundshot is in fact a new software product part of this DECODE deliverable, to provide a seamless and minimalist way to distribute signed updates of the DECODE OS to all running nodes, with the only need to reboot them in order to activate the updates. Roundshot is engineered with minimalism in mind and consists only of a set of declarative scripts (GNU Makefile) to pack a new system ramdisk that will operate the update.

3.2.1 Cross-toolchain and initramfs

When ran, Roundshot will first build a musl-based GCC toolchain which can crosscompile and statically link armhf binaries for us. Static compilation is adopted in many critical parts of DECODE OS infrastructure also to keep integrity with the privilege escalation model "sup" that is described in DECODE's whitepaper.
The adoption of musl-libc is useful for having a static initramfs, and smaller binaries than what the GNU libc provides.

The ramdisk (initramfs) consists of busybox, btrfs tools, and gnupg, which provide tools necessary for fetching our over-the-air updates to the DECODE operating system. It also contains the logic of setting up the system to boot into the actual userspace that was built in another stage.

3.2.2 The Linux kernel

After finishing the initramfs, the makefile suite will continue with fetching Linux sources and the patches, applying them, and compiling the Linux kernel and the necessary device tree that ARM boards need to define their hardware. These device trees are used by the u-boot bootloader, which is explained further on.

3.2.3 The squashfs

The squashfs is the fourth phase of Roundshot, which takes the userspace we have bootstrapped and configured and compresses it into a read-only squashfs file. This very file can be rebuilt and is used when updates are shipped. If an update is available, this file is replaced in-place on the machine which has downloaded the update after rebooting.

On the actual hardware, this squashfs is configured and set up through the logic found in our initramfs. The squashfs also contains the kernel modules that were built in the previous phase. The modules are not needed inside the initramfs.

3.2.4 The Bootloader

The Olimex Lime2 - much like all other ARM SoCs - uses the u-boot bootloader. Luckily, in the case of Lime2, we don't require any proprietary blobs or firmware and we can utilize explicitly libre software here. u-boot is built using the musl-libc cross-toolchain as well, and is flashed on the microSD card. The device tree file that we compiled with Linux sources is read by u-boot at the boot stage, along with boot.scr which is a u-boot boot script that contains the necessary information of what partitions/devices the dtb, the kernel, and the initramfs can be found on.

Finally, when all of this is finished, the results are packed and compressed into a tar.gz file, and a raw .img file. These files are used for initial distribution and are to be flashed on microSD cards.

4 Decode OS hardware deployment

This section gives a quick and visual explanation on how to deploy the provided DECODE OS image on an Olimex Lime2 board and power it up. After this the DECODE HUB will be online and automatically connect to all other DECODE nodes.
Upon creating the userspace and kernelspace, we deploy our results on the Olimex OLinuXino LIME2 ARM SoC.

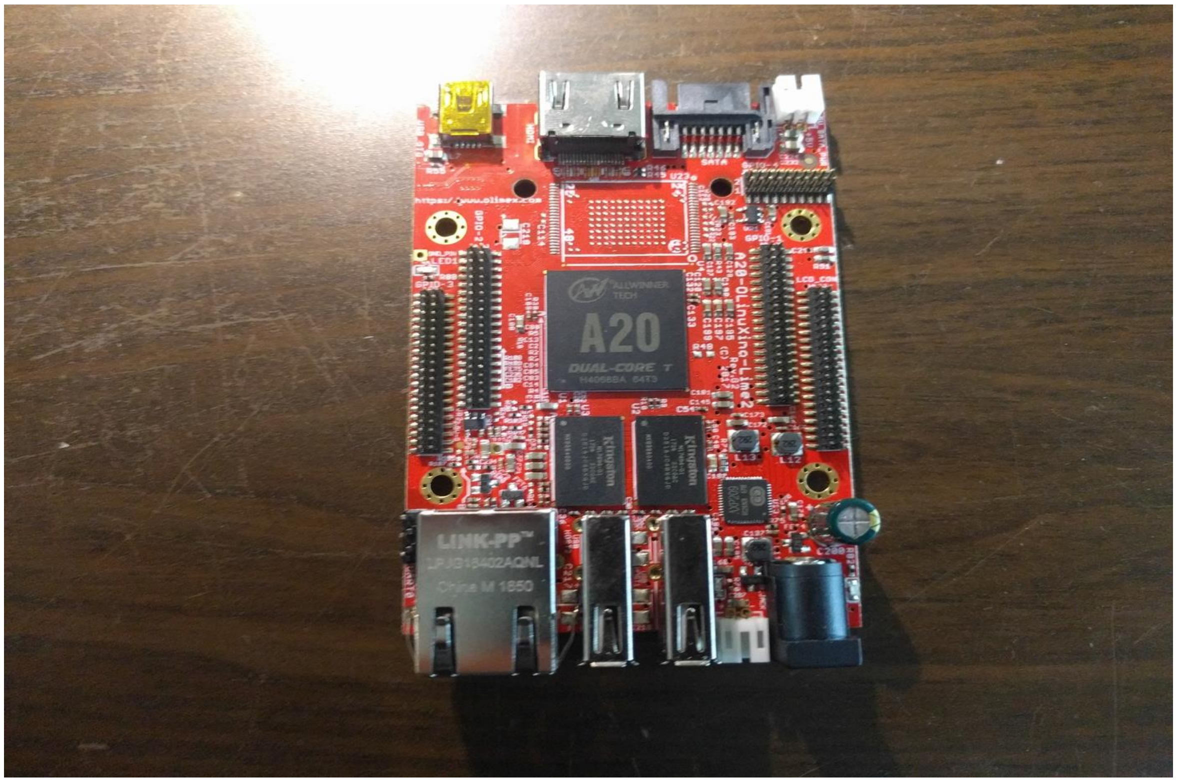

Figure 4

Following up, we connect the SATA hard drive with the data and power cables that are custom-built by Olimex.

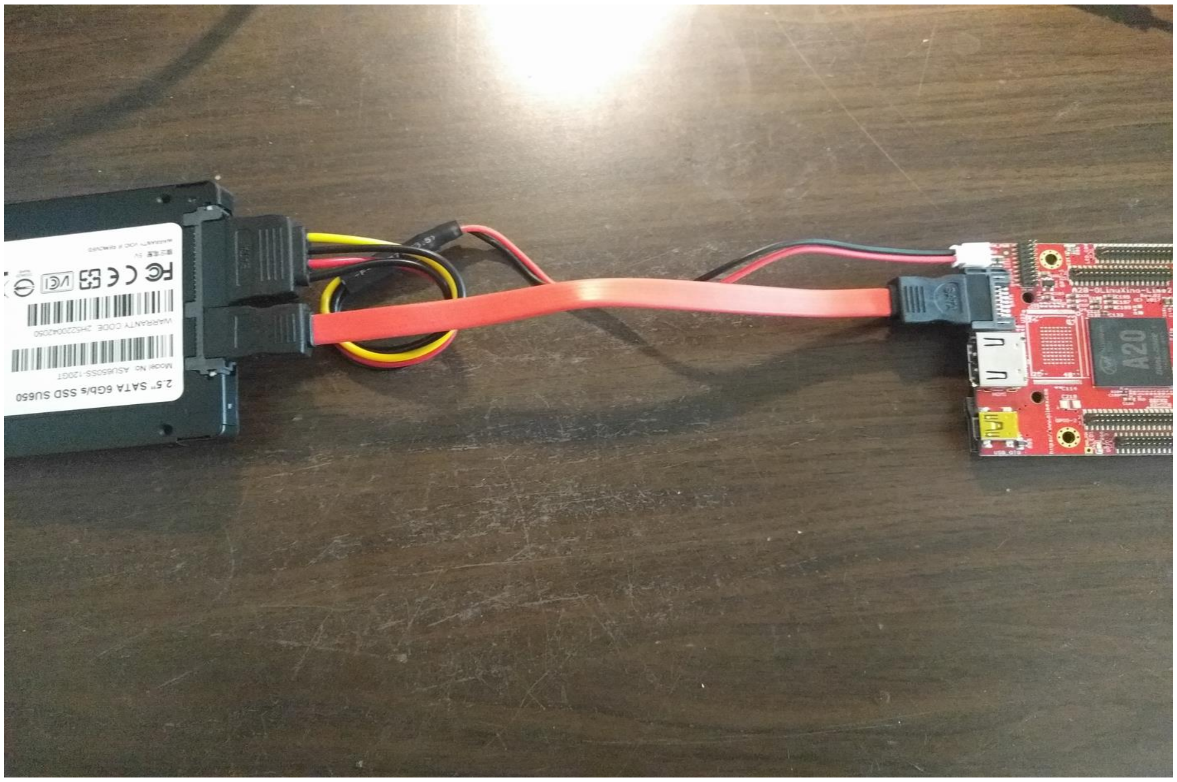

Figure 5

We flash the image on a microSD card and put it in place.

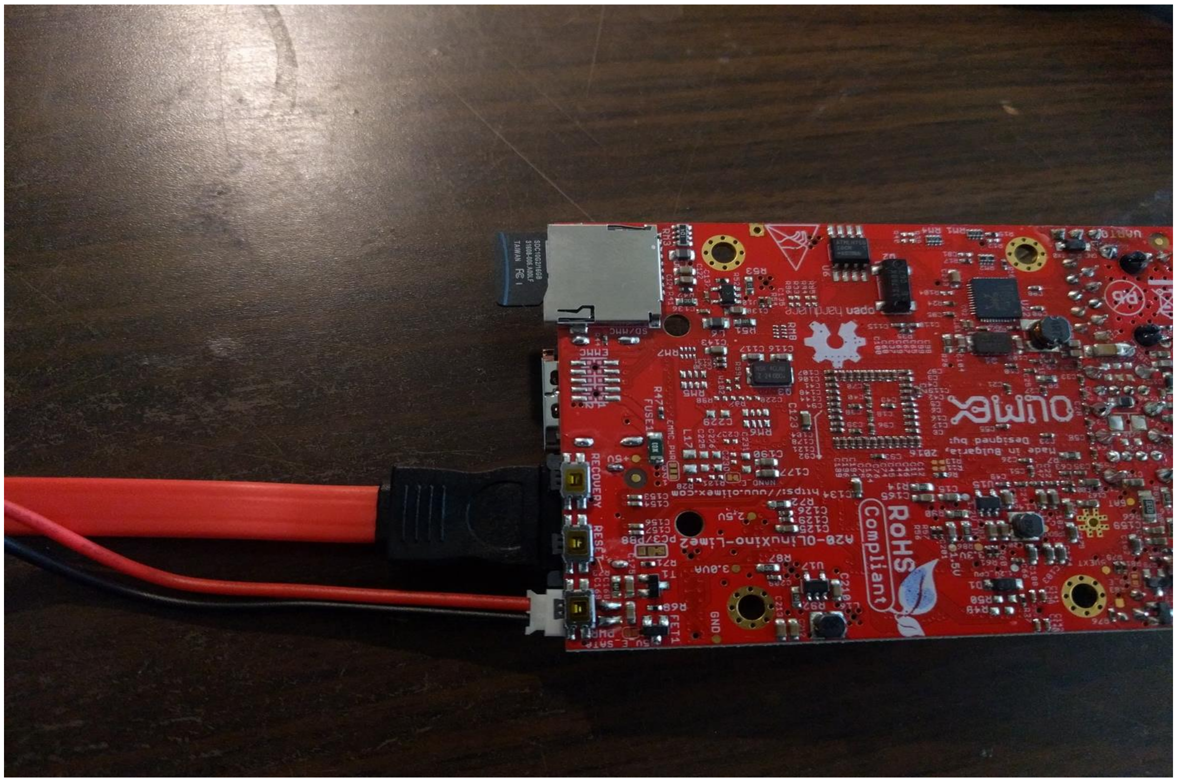

Figure 6

The initramfs requires a network connection to be able to check for and eventually download available updates to the system. It assumes that the router it's connecting to offers DHCP.

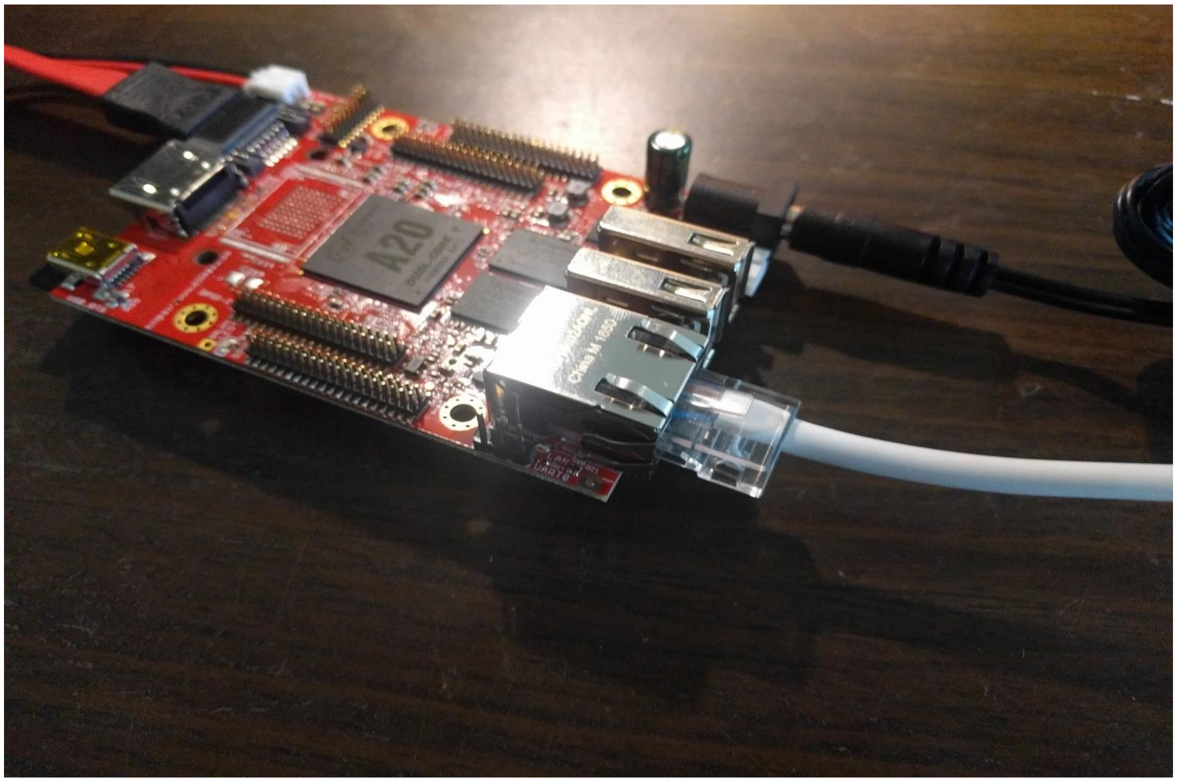

Figure 7

Finally, by connecting the battery and the power supply, the board will start booting and if everything is done properly, in a couple of minutes it will boot into DECODE OS.

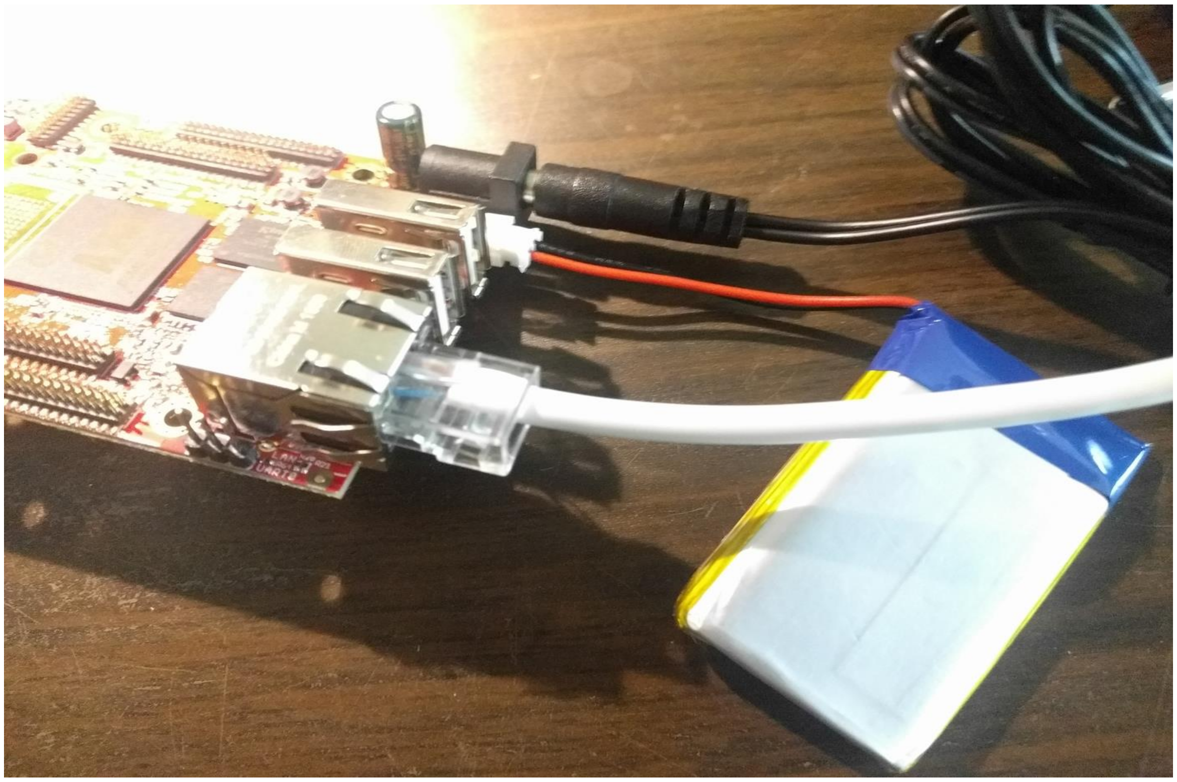

Figure 8

5 Network activity

This section offers a brief overview of all the visible network activity that a DECODE NODE produces when operated.

5.1 Local network

On bootup, DECODE nodes will request DHCP over Ethernet.
In case of lack of DHCP the box will boot after a delay, but will not be online and will not join the DECODE network. The board can eventually be configured by hand by connecting a keyboard and an HDMI monitor.

5.2 Incoming traffic

DECODE nodes do not require any input ports to be open of forwarded, nor any UPNP capability on the LAN gateway. Nodes route all their traffic via the Tor network and offer the possibility to application space utilities to receive any communication via Socks5 protocol router by Tor. This also means that there are no common firewall configurations that can stop the DECODE node from connecting: once it has DHCP and at least web access it will manage to join the Tor network and get in touch with all the DECODE network.

5.3 Outgoing traffic

Once received a local IP via DHCP, DECODE nodes build outgoing connections to:

1. pool.ntp.org (port 123) to get current time
2. dam.decodeproject.eu (port 80) to see if there are available system updates
3. if there are updates, dam.decodeproject.eu:80 is accessed again to download the update

These is the only deterministic traffic produced by a DECODE node upon boot and can be considered a base for fingerprinting nodes running on the local network, recognizing them from this initial behavior. All subsequent connections are routed through Tor and therefore opaque even to the local network administrators.

5.4 Continued operation

Once this first configuration phase (initramfs phase) is done, we continue booting into the actual system. The system will again request NTP on outbound port 123 and continue by starting all the remaining services in userspace. All services are tunnelled through Tor socks and therefore do not require any inbound or outbound connections outside of Tor.

Tor’s network requirements can be found in the Tor Project FAQ, most notably:

- Tor Outbound Ports documentation
- Tor Firewall Ports documentation

DECODE nodes operate only as hidden services inside Tor and connect only to other DECODE nodes: they do not act as entry, relay or exit nodes for the Tor network.

6 Conclusion

This brief deliverable consists of accompanying documentation to the following software implementations realised for the DECODE project:

- DECODE OS Devuan blend (ARM and VM targets)
- Tor Dam software application (Go language)
- Roundshot ramdisk (GNU Makefile)

The implementation of the OS with the HUB hardware platform has been a straightforward process and it did not induce any compromise on the freedom of software and hardware deployed. The DECODE HUB when equipped is usable as a reliable platform for P2P networks that abide to the principles of privacy by design enounced in D4.1.

The work outlined in this deliverable will proceed further to test the behaviour of DECODE HUBs in various situations, functioning as edge routers to the DECODE network, with their deployment on premises of various DECODE pilot partners.

The upcoming deliverable D4.10 "DECODE HUB devices piloted in trials" will report field tests conducted on pilots, while the D4.16 "Report on DECODE architecture stability and usability" shall outline the stability and usability of this component used in conjunction with DECODE application space tooling as well as standalone base for more modular P2P applications in need to deploy cloud-shaped micro-service swarms whose connections can be private, pseudonimic and encrypted.