Raspberry Pi Setup

- Install Raspbian Lite. We currently use Debian 10 (Buster).
- Change password by using

```
passwd pi
```

- Activate SSH

```
sudo systemctl enable ssh
sudo systemctl start ssh
```

- Install and configure NTP
- Install tmux

```
 sudo apt install tmux
```

- "Install" the RTI Connext libraries under `/usr/local/lib/` by copying the arm-folder from the Main PC under `opt/rti_connext_dds-6.0.0/lib/armv6vfphLinux3.xgcc4.7.2/`. This can of course only be done after the ARM libraries have been installed on the Main Computer.

It should look like this afterward:

```
pi@raspberrypi-06:~ $ find /usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddscppd.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddscd.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddstransporttcpzd.a
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddscpp2z.a
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/librtidlcppz.a
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/librtimonitoringd.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddstransporttcp.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddscpp2.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddscpp2d.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/librtimonitoringz.a
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddstransporttcpd.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddscz.a
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddscppzd.a
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddsc.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddstransporttcpz.a
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/librtidlcpp.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddscpp.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddscore.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/librtimonitoringzd.a
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddscorez.a
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddscppz.a
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddscored.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/librtidlcppd.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddsjava.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddscpp2zd.a
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddsczd.a
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddscorezd.a
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/libnddsjavad.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/librtimonitoring.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/librtidlcz.a
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/librtidlcppzd.a
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/librtidlc.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/librtidlcd.so
/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2/librtidlczd.a
```

'Register' the Libraries

Make the libraries available. Insert `/usr/local/lib/rti_connext_dds-6.0.0/armv6vfphLinux3.xgcc4.7.2` into `/etc/ld.so.conf`. Then run

```
sudo ldconfig
```

Configure Bootup Behaviour

Copy the file `git:software/vehicle_raspberry_firmware/bootloader_raspberry.bash` to `/root/bootloader_raspberry.bash` on the Raspberry.

Enable autostart, using `sudo nano /etc/rc.local` and insert

tmux new-session -d -s "bootloader_raspberry" "bash /root/bootloader_raspberry.bash"

NTP configuration

See the NUC NTP-Configuration.

ID-Related Settings

Edit the `/etc/hostname` file to give the Raspberry a unique name. The name should correspond to the vehicle ID, for example `raspberrypi-06` or `raspberrypi-42`.

Create a DHCP reservation for the Raspberry on the router. The vehicle ID is derived from the IP. The IP is `192.168.1.1XX` where `XX` are the digits of the vehicle ID.

Read-Only Filesystem

One crucial aspect regarding the Raspberry Pi is, that the vehicle will be turned off by using the switch on the bottom, mostly. That will directly cut off the power supply of the RPI. In order to avoid damage to the OS, it is important to configure the filesystem to be read-only. For that, follow the instructions on https://kofler.info/raspbian-lite-fuer-den-read-only-betrieb/ . (As alternative https://www.mehr4u.de/raspberry-pi-mit-readonly-filesystem.html may also be used. Vehicle 20 is created by following the first link but using `fastboot noswap ro` instead of `fastboot noswap`.)

In order to make NTP working properly with a read-only filesystem, it is also necessary to copy the following file

```
cp /lib/systemd/system/ntp.service /etc/systemd/system/ntp.service
```

and to comment out `PrivateTmp=true` in

`/etc/systemd/system/ntp.service`

If you need *writing access* to the file system, you can use (dependent on the folder you need access to)

```
sudo mount -o remount,rw /
sudo mount -o remount,rw /boot
```

to remount the system.

Applying RT-Patch

Because the vehicle needs to be real-time capable, a real-time patch (RT-Patch) should be applied to the underlying Linux Kernel.

Follow these instructions to build the kernel. It is recommended to use the section *Cross-Compiling* instead of *Local Building* to speed up the process (e.g. ~15 minutes vs. ~2 hours). Additionally, there has to be done several annotations:

- **Get Sources:**
  - Make sure that there exists a real-time patch for the current kernel version. You can see this version by looking at the first lines of the makefile residing in the cloned directory, e.g., by using
    `head Makefile -n 4`
    The currently existing real-time patches can be found here.
- **Before Build sources**
  - Download the RT-Patch matching your kernel version and patch the kernel according to these instructions. (Execute the patch-command in the root directory of the cloned git repository.)
- **Build sources**
  - Run the commands which depend on the RPi version.

- Run `make menuconfig`, navigate to `General Setup > Preemption Model`, choose `Fully Preemptible Kernel (RT)` and save.
- Run the lastly given make command.
  - Make sure to use the parameter `-j n` to configure how many threads are used for this command in order to speed up this process as explained.
  - It might happen that you are **asked to set up** the .config file again which you already configured in the last step. (The console will show something like `*Restart config...` and asks you questions.) Then press `strg+c` and replace the command by `make ARCH=arm menuconfig CROSS_COMPILE=arm-linux-gnueabihf- zImage modules dtbs`
    Now, the menu will open again, but you can quit that by only **?saving?** and leaving the menu. Afterwards, the process should start building as usual.

Troubleshooting

It might happen that the system clock is not automatically synchronizing via NTP if it differs too much from the real-time. Firstly, take care that your timezone is set correctly:

```
sudo raspi-config
```

In the opening UI go to: 4 Localisation Options I2 Change Time Zone None of the above UTC

Furthermore, you can force NTP to set the time once. Therefore, stop the service, force it, and restart it again :

```
sudo /etc/init.d/ntp stop
sudo ntpd -qg
sudo /etc/init.d/ntp start
```

Afterwards, NTP should synchronize automatically.

Repeating the process for several vehicles

In order to create identical vehicles it might be useful to clone the SD-cards by using Clonezilla. Afterwards, only the ID-Related Settings have to be adapted at the cloned SD-card.

Cross Compilation and Software Distribution

The vehicle Raspberry software is cross-compiled from the master PC. The build script `git:software/vehicle_raspberry_firmware/build.bash` creates an archive of the compiled software and other required files and publishes it via Apache. The `bootloader_raspberry.bash` autostart script on the Raspberry downloads and runs this software.

ARP Table Generation

Because unsuccessful ARP requests sometimes block communication a static ARP table is sometimes useful. To generate one one needs to switch on all participants, ping each one from the main PC and then output the arp table on the main pc with e.g. https://man7.org/linux/man-pages/man8/ip-neighbour.8.html (newer) or `arp -a` (old).
The info can then be used to feed a script that sets up the arp entries statically for the participants. Currently this is done in https://git.rwth-aachen.de/CPM/Project/Lab/software/-/blob/feature/fastdds/mid_level_controller/package/start.bash and the static arp table is part of the package.