ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Mose Müller

An Embedded Linux Solution to Monitor Laboratory Environmental Parameters

Semester Thesis

Institute for Quantum Electronics, Swiss Federal Institute of Technology (ETH), 8093 Zurich, Switzerland

Supervision

Oliver Wipfli
Prof. Dr. J. P. Home

January 17, 2022

Contents

Abstract

This project is a buildroot implementation of an embedded Linux system on a Banana Pi P2 Zero (BPI). It is designed to read out sensor data in the lab for the following sensors:

1. Adafruit SHT31-D: temperature and humidity sensor
2. Adafruit ADS1115: analogue-to-digital converter (returns single-ended values in mV)
3. Adafruit MCP9808: temperature sensor
4. Adafruit BMP180: pressure and temperature sensor
5. Adafruit LMS303DLHC: 3-axis magnetometer/accelerometer

The code is stored on a gitlab repository of the TIQI group. The buildroot project can be built and the bootable image be inserted into a Banana Pi P2 Zero (BPI) upon flashing it onto an SD card. Connection to the BPI is possible through both SSH and serial UART. Readout of the sensors is demonstrated using the sysfs interface. Sensor support of both Hardware Monitoring and Industrial I/O subsystem is implemented using Linux kernel drivers and devicetree overlays of the listed I2C sensors.

Chapter 1

Usage

1.1 How to Build

To download and build the buildroot project, execute the following commands from your local computer (on a Linux machine)

```
$ git clone --recursive https://gitlab.phys.ethz.ch/tiqi-projects/banana-pi-buildroot.git
$ cd banana-pi-buildroot
$ git submodule update --init --remote
$ make
```

Once the build process is finished you will have an image called "sdcard.img" in the `build/images/` directory.

1.2 Flashing the Image onto an SD Card

You can flash the bootable image `build/images/sdcard.img` onto an SD card using `dd` from the terminal

```
$ sudo dd if=build/images/sdcard.img of=/dev/sdX bs=1M status=progress
$ sync
```

You should **replace `X` with the actual drive letter** (`a`, `b`, `c`, ...) that you can identify using

```
$ sudo lsblk    # lists information about block devices
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sdb           8:16   1  14.8G  0 disk
zram0       252:0    0     8G  0 disk [SWAP]
nvme0n1     259:0    0 953.9G  0 disk
├─nvme0n1p1 259:1    0   600M  0 part /boot/efi
├─nvme0n1p2 259:2    0     1G  0 part /boot
└─nvme0n1p3 259:3    0 952.3G  0 part /home
```

Here, the SD card is located at `/dev/sdb`.
You can now insert the micro SDcard in your Bananapi P2 Zero and power it up. For information on how to connect to the BPI, see chapter 2.

1.3 Reading out Sensors

The sensors this system is configured for are supported by either the Hardware Monitoring (HWMON) or the Industrial I/O (IIO) subsystem. We will shortly cover what these subsystems' purposes are and how you can interact with the drivers and sensor data using the so-called *sysfs interface*.

1.3.1 Industrial I/O

IIO's main purpose is to provide support for devices that in some sense perform either analog-to-digital conversion (ADC) or digital-to-analog conversion (DAC) (see [1]).

You can interact with an IIO driver through the *IIO device sysfs interface*. The attached devices can be found under `/sys/bus/iio/devices/`:

```
root@TIQI:~$ ls /sys/bus/iio/devices/
iio:device0
```

The attributes of the devices are sysfs files which expose chip information and also allow to set various configuration parameters. For a device with index `X`, attributes can be found under `/sys/bus/iio/devices/iio:deviceX/`. For example, the output for a LSM303DLHC sensor looks like:

```
root@TIQI:~$ ls -la /sys/bus/iio/devices/iio:device0/
total 0
drwxr-xr-x    6 root     root             0 Jan  1 05:18 .
drwxr-xr-x    4 root     root             0 Jan  1 05:18 ..
drwxr-xr-x    2 root     root             0 Jan  1 05:18 buffer
-rw-r--r--    1 root     root          4096 Jan  1 05:18 current_timestamp_clock
-r--r--r--    1 root     root          4096 Jan  1 05:18 dev
-r--r--r--    1 root     root          4096 Jan  1 05:18 in_accel_scale_available
-rw-r--r--    1 root     root          4096 Jan  1 05:18 in_accel_x_raw
-rw-r--r--    1 root     root          4096 Jan  1 05:18 in_accel_x_scale
-rw-r--r--    1 root     root          4096 Jan  1 05:18 in_accel_y_raw
-rw-r--r--    1 root     root          4096 Jan  1 05:18 in_accel_y_scale
-rw-r--r--    1 root     root          4096 Jan  1 05:18 in_accel_z_raw
-rw-r--r--    1 root     root          4096 Jan  1 05:18 in_accel_z_scale
-r--r--r--    1 root     root          4096 Jan  1 05:18 name
lrwxrwxrwx    1 root     root             0 Jan  1 05:18 of_node ->
        ../../../../../../../firmware/devicetree/base/soc/i2c@1c2ac00/lsm303dlhc@19
drwxr-xr-x    2 root     root             0 Jan  1 05:18 power
-rw-r--r--    1 root     root          4096 Jan  1 05:18 sampling_frequency
-r--r--r--    1 root     root          4096 Jan  1 05:18 sampling_frequency_available
drwxr-xr-x    2 root     root             0 Jan  1 05:18 scan_elements
lrwxrwxrwx    1 root     root             0 Jan  1 05:18 subsystem ->
        ../../../../../../../bus/iio
drwxr-xr-x    2 root     root             0 Jan  1 05:18 trigger
-rw-r--r--    1 root     root          4096 Jan  1 05:18 uevent
```

You can use `cat` to print out the values of the attributes and `echo` to set the writable attributes. For more information about naming and data format standards for sysfs files see the Linux kernel sysfs-interface page [2].

To confirm the name of the chip and the name of the sensor given in the devicetree overlay, you can cat the `name` files:

```
root@TIQI:~$ cat /sys/bus/iio/devices/iio:device0/name
lsm303dlhc_accel
root@TIQI:~$ cat /sys/bus/iio/devices/iio:device0/of_node/name
lsm303dlhc
```

As you can see, the chip name is *lsm303dlhc_ accel* (automatically assigned) while the node in the devicetree overlay is *lsm303dlhc*.

1.3.2 Hardware Monitoring

Hwmon is directed at low sample rate sensors used to monitor and control the system itself, like fan speed control or temperature measurement (see [2]).

As with the IIO drivers, you can access the sysfs files directly to read out the attributes of HWMON devices. Thus, you can write an application program that scans for entries and accesses this data in a simple and consistent way.
Each chip gets its directory in the sysfs `/sys/devices` tree. To find all sensor chips, it is easier to follow the device symlinks from `/sys/class/hwmon/hwmon*`:

```
root@TIQI:~$ ls /sys/class/hwmon/
hwmon0  hwmon1
```

The attributes of the devices are sysfs files which expose chip information and also allow to set various configuration parameters. For a MCP9808 sensor you find the contents:

```
root@TIQI:~$ ls -la /sys/class/hwmon/hwmon0/
total 0
drwxr-xr-x    3 root     root             0 Jan  1 05:17 .
drwxr-xr-x    3 root     root             0 Jan  1 05:17 ..
lrwxrwxrwx    1 root     root             0 Jan  1 05:17 device -> ../../../0-0018
-r--r--r--    1 root     root          4096 Jan  1 05:17 name
lrwxrwxrwx    1 root     root             0 Jan  1 05:17 of_node ->
        ../../../../../../../../firmware/devicetree/base/soc/i2c@1c2ac00/mcp9808@18
drwxr-xr-x    2 root     root             0 Jan  1 05:17 power
lrwxrwxrwx    1 root     root             0 Jan  1 05:17 subsystem ->
        ../../../../../../../../class/hwmon
-rw-r--r--    1 root     root          4096 Jan  1 05:17 temp1_crit
-r--r--r--    1 root     root          4096 Jan  1 05:17 temp1_crit_alarm
-rw-r--r--    1 root     root          4096 Jan  1 05:17 temp1_crit_hyst
-r--r--r--    1 root     root          4096 Jan  1 05:17 temp1_input
-rw-r--r--    1 root     root          4096 Jan  1 05:17 temp1_max
-r--r--r--    1 root     root          4096 Jan  1 05:17 temp1_max_alarm
-r--r--r--    1 root     root          4096 Jan  1 05:17 temp1_max_hyst
-rw-r--r--    1 root     root          4096 Jan  1 05:17 temp1_min
-r--r--r--    1 root     root          4096 Jan  1 05:17 temp1_min_alarm
-rw-r--r--    1 root     root          4096 Jan  1 05:17 uevent
```

You can use `cat` to print out the values of the attributes and `echo` to set the writable attributes. For more information about naming and data format standards for sysfs files see the Linux kernel sysfs-interface page [2].

To confirm the name of the chip and the name of the sensor given in the devicetree overlay, you can cat the `name` files:

```
root@TIQI:~$ cat /sys/class/hwmon/hwmon0/name
```

```
jc42
root@TIQI:~$ cat /sys/class/hwmon/hwmon0/of_node/name
mcp9808
```

As you can see, the chip name is *jc42* (automatically assigned) while the node in the devicetree overlay is *mcp9808*.

1.3.3 Example Script

Here is an example script that reads the sysfs file containing the temperature every second. You will have to manually choose `hwmon0` according to the device you use and adapt `temp1_input` to the sysfs attribute you want to read out (or write a better script that does this for you :)).

```
import time

sleep_time = 1 #interval time in seconds

def read_temp():
    with open('/sys/class/hwmon/hwmon0/temp1_input', 'r') as f:
        temp = f.readline().strip()    # truncating new line character
        print(float(temp)/1000)        # casting to float, convert to Celsius

while(True):
    read_temp()
    time.sleep(sleep_time)
```

Chapter 2

Connecting to the BPI

There are multiple ways to connect to the BPI. The SSH protocol might be the most well known, but there are reasons for using other protocols like serial UART (like troubleshooting U-Boot). In the following, we will go through the usage of the mentioned protocols.

2.1 SSH

When the BPI is connected to the same LAN as you, you can ssh into it using a private SSH key. As an IP is assigned to the BPI automatically (DHCP) you might first have to find that out. Afterward, you can connect to it using

```
$ ssh root@xx.xx.xx.xx -i ~/.ssh/id_ed25519
```

The private key `id_ed25519` is stored in the common TIQI bitwarden account.

2.2 Serial UART

The other option is to connect to the BPI using the serial UART protocol. For this, you can use the Adafruit FT232H breakout board [3]. Connecting to it via USB enables the computer to talk to peripherals like sensors or single-board computers through its GPIO pins over protocols like SPI, I2C, serial UART, JTAG, and more.

2.2.1 Wiring

The pinouts of the FT232H breakout can be seen in Fig. 2.1. The pins important for our application are

- **GND**: ground of the breakout board
- **D0**: TX or transmit pin. This is the serial data output of the FT232H. This should be connected to the RX / receive pin of a serial device
- **D1**: RX or receive pin. This is the serial data input of the FT232H. It should be connected to the TX / transmit pin of a serial device
- (**5V**: 5-volt power source connected directly to the USB bus)

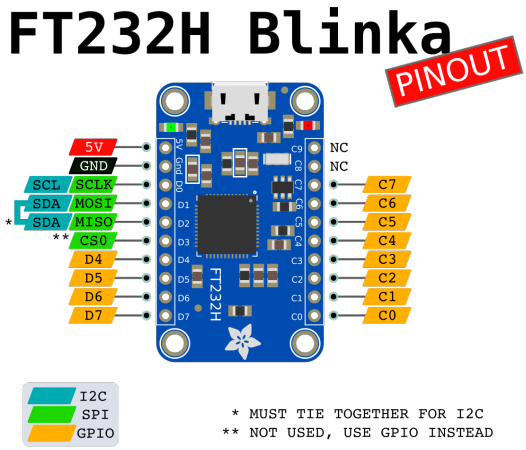

Figure 2.1: Pinouts of the FT232H breakout [4].

The GND, TX and RX pins of the FT232H board have to be connected to the GND, RX, and TX pins of the BPI, respectively, to interact with it over serial UART. Moreover, you have to plug a micro USB cable into the port on the FT232H breakout and connect it to a computer or laptop (you should see a green LED next to the 5V pin on the breakout light up then).

2.2.2 USB Serial Drivers

Before using the breakout as a serial UART you have to make sure the proper serial port drivers are installed. Most operating systems include FTDI's serial UART drivers (such as Mac OSX Mavericks or greater and Linux kernels since ~2.6). However, should this not be the case, refer to the serial UART section of the FT232H breakout webpage for manual installation instructions [3].

2.2.3 Serial Usage

After doing the wiring, you can start up the BPI and open a serial console for the FT232H breakout as follows:

1. List all serial ports using

   ```
   $ ls /dev/tty*
   ```

   Do this for both disconnected and connected board to make out the newly connected device.

2. Install and use `screen` to open the serial UART

   ```
   $ sudo apt install screen
   $ sudo screen /dev/ttyUSB0 115200
   ```

   where `/dev/ttyUSB0` is the name of the device you found in step 1. The number behind the name of the device (`115200`) is called the *baud rate*. It is a term for the number of bits per second that are transmitted over the wire. There's a couple of common baud rates (9600, 115200, 19200, 38400). If not set correctly, the console will typically show gibberish.

3. Login
   After connecting to the BPI you should be greeted by the following login prompt

```
Welcome to Buildroot for the TIQI Bananapi P2 Zero
TIQI login:
```

The default username is `root` with no password.

If this prompt does not show up, you can simply restart the BPI using the reset key between the GND pin and the red LED on the BPI (see Fig. 2.2).

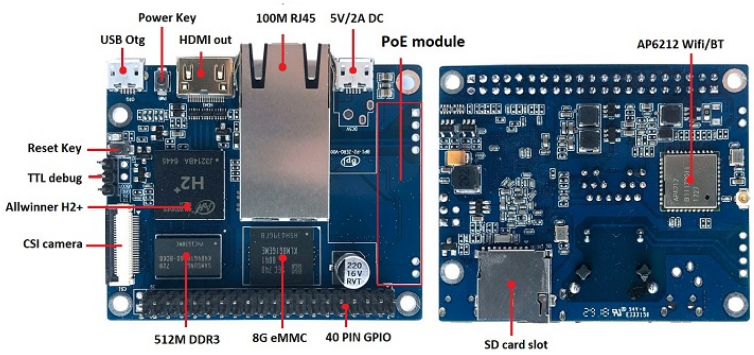

Figure 2.2: BPI F2 Zero interface (similar to P2 Zero) [5].

Chapter 3

Connecting I2C Sensors

3.1 Connecting an I2C Sensor

A sensor is easily connected to the BPI by connecting the 5V (Vin), GND, SCL, and SDA pins of the BPI with those of the sensor. The pins on each sensor are clearly stated, the 40-pin GPIO header of the BPI matches that of the Raspberry Pi 3 (see Fig. 3.1).

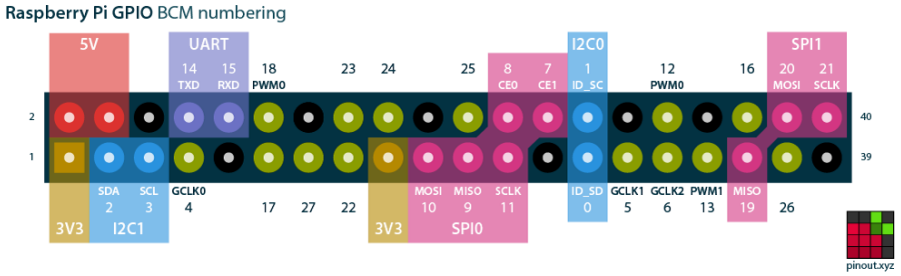

Figure 3.1: BPI GPIO pinout (matches RPI 3). The black dots correspond to ground [6].

3.2 Connecting multiple I2C Sensors

You can also connect multiple sensors to the same I2C port. The sensors then must be connected in parallel (see Fig. 3.2).

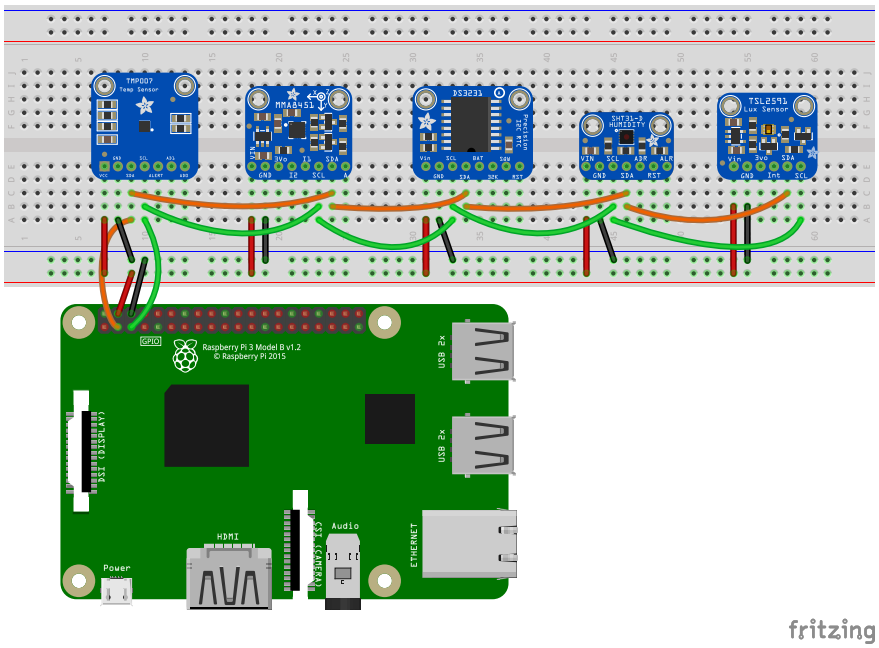

Figure 3.2: A RPI 3 connected to five devices at the same time (adapted from [7]). The red cables supplies the 5V input voltage, the black cables connect to the ground, the orange cables connect the SDA pins and the green cables the SCL pins.

Chapter 4

Adding Sensor Support

By default, new devices are not recognized by the BPI. They need to both be supported by the used Linux kernel and added to the devicetree of the BPI (as a devicetree overlay), as well.

4.1 Adding Linux Kernel Support

For your sensor to be supported by our Linux kernel, you have to add a driver that supports it. These drivers can be loaded directly into the kernel or only as a module (more on this below). The goal of this section is to provide an easy way to determine the driver that supports the sensor you want to add and to add this driver to the Linux kernel you are building for the BPI.

In buildroot, there are two ways of adding kernel drivers to the Linux kernel. You can either (de-)activate drivers in the Linux configuration editor (use `make linux-menuconfig` from the buildroot folder) or add the needed option manually to the `buildroot-external/linux.config` file (which is what we will do here).

1. Adding config line to linux.config

   To get the correct option to be added to the linux.config file your can either look at the kernel documentation of hardware monitoring kernel drivers or the Linux kernel tree (see [8, 9]). In the latter, all supported drivers are listed and you can search for your sensor (for some reason, not all hwmon kernel drivers are listed in the kernel documentation). For this, you need to look at the `Kconfig` files to get the name of the driver (so you could actually search all the files `drivers/iio/*/Kconfig` and `drivers/hwmon/Kconfig` for the name of your sensor). The name corresponding to the config entry of your sensor has to be included in our `linux.config` file:

   ```
   CONFIG_SHT3X=y # for sht31d
   CONFIG_BMP280=y # for bmp180
   CONFIG_IIO_ST_ACCEL_3AXIS=y # for lsm303dlhc
   CONFIG_TI_ADS1015=y # for ads1115
   CONFIG_SENSORS_JC42=y # for mcp9808
   ```

   The drivers are loaded into the kernel when specifying `=y` and only loaded as modules when using `=m`.

2. Optional: Adding modules to the kernel
   If you decide to load a kernel driver only as a module, you have to use `modprobe` to add the kernel module to the Linux kernel.

   You can list the available modules using

```
root@TIQI:~$ modprobe -l
kernel/lib/crc8.ko
kernel/drivers/hwmon/sht3x.ko
```

and then activate the needed one using

```
root@TIQI:~$ modprobe sht3x
```

You can then check whether it was successfully loaded using `lsmod`

```
root@TIQI:~$ lsmod
Module                  Size  Used by    Not tainted
sht3x                  16384  0
crc8                   16384  1 sht3x
```

4.2 Writing a Devicetree Overlay

A devicetree describes the hardware components of a computer so that the operating system's kernel can use and manage those components. The purpose of a *devicetree overlay* is to override specific parts of the kernel's live tree before booting the operating system. They are special devicetree blob fragments that contain modifications that can be applied to an existing devicetree blob (see [10, 11]).

The devicetree overlays written for the supported sensors are in the `buildroot-external/dtoverlays` folder (see appendix B). When writing a new devicetree overlay, you can use them for inspiration or use the following template:

```
/dts-v1/;
/plugin/;
/ {
    compatible = "sinovoip,bpi-m2-zero", "allwinner,sun8i-h2-plus";

    fragment@0 {
        target = <&i2c0>;
        __overlay__ {
            #address-cells = <1>;
            #size-cells = <0>;

            deviceName: deviceName@addr {
                compatible = "corp,foo";
                reg = <0xaddr>;
                status = "okay";
            };
        };
    };
};
```

Besides giving the `deviceName` node a custom name you mostly just have to adapt the `compatible` and `reg` properties for the devicetree overlay to work. However, for additional properties which might be needed to set up the device, please refer to its devicetree documentation (IIO [12], HWMON [13]).

Thus, the task here consists of

1. Choosing a sensible device name, e.g. the name of the sensor
2. Finding out the I2C address `reg = <?>` the sensor will respond to:
   After connecting the sensor to the BPI (see section 3.1) you can use `i2cdetect` to get the address of the sensor

```
root@TIQI:~$ i2cdetect -y 0
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- 44 -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

   In the above example, the address of the attached sensor is *44*, meaning you should change the register entry to `reg = <0x44>`.

3. Adapting `compatible`:
   The "compatible" line describes which platforms the devicetree overlay is designed to work with, starting with most compatible to the least compatible. Mostly, the string preceding the comma ("corp") corresponds to the manufacturer while "foo" is the sensor (hardware, more generally) itself.

   You can find the correct entry by

   - searching in the devicetree bindings (IIO [12], HWMON [13]). Mostly, an example of a devicetree entry is given there. However, not all sensors are listed but you can get a good idea of how the `compatible` string should look like (e.g. every manufacturer has its specific abbreviation, ...),
   - googling for a devicetree overlay of the new sensor, or
   - looking at the drivers itself (IIO [14], HWMON [15]) which are written in C. Sometimes you can find a `static const struct of_device_id` listing possible `compatible` strings.

4.2.1 Enabling a Devicetree Overlay

When a devicetree overlay has been written and added to the dtoverlays folder, it will be compiled from a devicetree source (`.dts`) into a devicetree binary file (`.dtb`) using `dtc`. As we are only concerned about overlay files we use `.dtbo` as file extension (this is just a convention). These files are then written into the `dtbs` folder on the `boot` partition of the `sdcard.img` file. This partition then looks like this:

```
.
├── banana.dtb
├── boot.scr
├── dtbs/
│   ├── ads1115.dtbo
│   ├── bmp180.dtbo
│   ├── config.txt
│   ├── lsm303dlhc.dtbo
│   ├── mcp9808.dtbo
```

```
│   └── sht3x.dtbo
└── zImage
```

To load and apply those `.dtbo` files to the main devicetree we have to add the following to the `config.txt` file:

```
overlay=overlay_name_1, overlay_name_2, ...
```

where the overlay name is the file name without the extension. After the name has been added the devicetree overlays will be dynamically merged into the main devicetree.

Appendix A

Setup

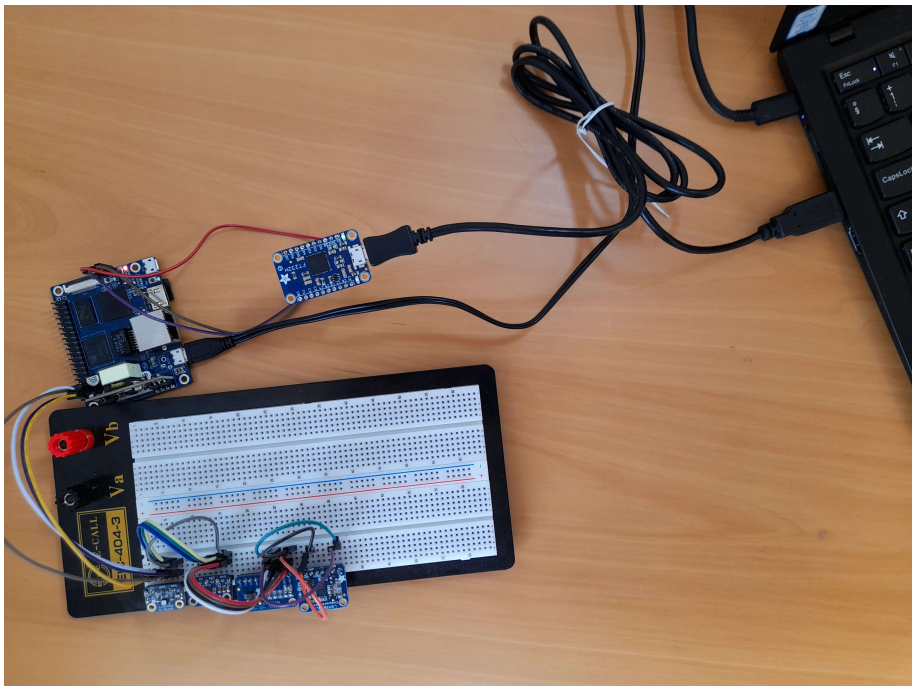

Figure A.1: The setup used in this project. A FT232H breakout is connected via USB to a laptop and via serial UART to a Banana Pi P2 Zero. To the BPI four I2C devices are connected in parallel on a breadboard.

Appendix B

Devicetree Overlay Examples

Listing B.1: `ads1115.dts`

```
/dts-v1/;
/plugin/;
/ {
    compatible = "sinovoip,bpi-m2-zero", "allwinner,sun8i-h2-plus";

    fragment@0 {
        target = <&i2c0>;
        __overlay__ {
            #address-cells = <1>;
            #size-cells = <0>;

            ads1115: ads1115@48 {
                compatible = "ti,ads1015";
                reg = <0x48>;
                #address-cells = <1>;
                #size-cells = <0>;
                status = "okay";
            };
        };
    };
};
```

Listing B.2: `bmp180.dts`

```
/dts-v1/;
/plugin/;
/ {
    compatible = "sinovoip,bpi-m2-zero", "allwinner,sun8i-h2-plus";

    fragment@0 {
        target = <&i2c0>;
        __overlay__ {
            #address-cells = <1>;
            #size-cells = <0>;

            bmp180: bmp180@77 {
```

```
                compatible = "bosch,bmp180";
                reg = <0x77>;
                status = "okay";
            };
        };
    };
};
```

Listing B.3: `lsm303dlhc.dts`

```
/dts-v1/;
/plugin/;
/ {
    compatible = "sinovoip,bpi-m2-zero", "allwinner,sun8i-h2-plus";

    fragment@0 {
        target = <&i2c0>;
        __overlay__ {
            #address-cells = <1>;
            #size-cells = <0>;

            lsm303dlhc: lsm303dlhc@19 {
                compatible = "st,lsm303dlhc-accel";
                reg = <0x19>;
                status = "okay";
            };
        };
    };
};
```

Listing B.4: `mcp9808.dts`

```
/dts-v1/;
/plugin/;
/ {
    compatible = "sinovoip,bpi-m2-zero", "allwinner,sun8i-h2-plus";

    fragment@0 {
        target = <&i2c0>;
        __overlay__ {
            #address-cells = <1>;
            #size-cells = <0>;

            mcp9808: mcp9808@18 {
                compatible = "microchip,mcp9808", "jedec,jc-42.4-temp";
                reg = <0x18>;
                status = "okay";
            };
        };
    };
};
```

Listing B.5: `sht3x.dts`

```
/dts-v1/;
/plugin/;
/ {
    compatible = "sinovoip,bpi-m2-zero", "allwinner,sun8i-h2-plus";

    fragment@0 {
        target = <&i2c0>;
        __overlay__ {
            #address-cells = <1>;
            #size-cells = <0>;

            sht3x: sht3x@44 {
                compatible = "sensirion,sht3x";
                reg = <0x44>;
                status = "okay";
            };
        };
    };
};
```

Bibliography

[1] Linux kernel Industrial I/O introduction. https://www.kernel.org/doc/html/latest/driver-api/iio/intro.html. [Online; accessed 17-January-2022].

[2] The sysfs-interface. https://www.kernel.org/doc/html/latest/hwmon/sysfs-interface.html. [Online; accessed 17-January-2022].

[3] Adafruit. FT232H breakout. https://learn.adafruit.com/adafruit-ft232h-breakout. [Online; accessed 17-January-2022].

[4] Adafruit. FT232H breakout pinouts. https://learn.adafruit.com/circuitpython-on-any-computer-with-ft232h/pinouts. [Online; accessed 17-January-2022].

[5] The Banana Pi P2 Zero. https://wiki.banana-pi.org/Banana_Pi_BPI-P2_Zero. [Online; accessed 17-January-2022].

[6] The Raspberry Pi GPIO pinout guide. https://pinout.xyz/. [Online; accessed 17-January-2022].

[7] Adafruit. I2C addresses. https://learn.adafruit.com/i2c-addresses. [Online; accessed 17-January-2022].

[8] Linux kernel HWMON driver documentation. https://www.kernel.org/doc/html/latest/hwmon/#hardware-monitoring-kernel-drivers. [Online; accessed 17-January-2022].

[9] Linux kernel tree. https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers. [Online; accessed 17-January-2022].

[10] Linux kernel devicetree overlay notes. https://www.kernel.org/doc/html/latest/devicetree/overlay-notes.html. [Online; accessed 17-January-2022].

[11] Digi.com. Use Device Tree Overlays to Patch Your Device Tree. https://www.digi.com/resources/examples-guides/use-device-tree-overlays-to-patch-your-device-tree. [Online; accessed 17-January-2022].

[12] IIO devicetree binding documentation. https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/Documentation/devicetree/bindings/iio. [Online; accessed 17-January-2022].

[13] HWMON devicetree binding documentation. https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/Documentation/devicetree/bindings/hwmon. [Online; accessed 17-January-2022].

[14] Linux kernel IIO drivers. https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/iio/. [Online; accessed 17-January-2022].

[15] Linux kernel HWMON drivers. https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/hwmon/. [Online; accessed 17-January-2022].