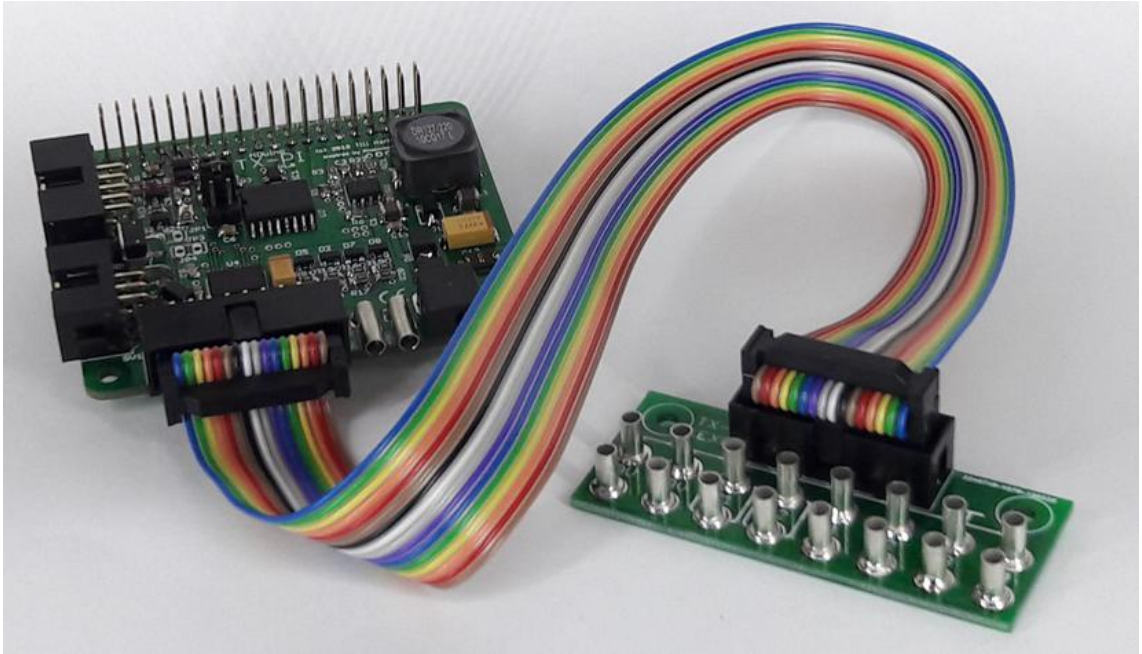
ft HAT

a fischertechnik compatible HAT for the Raspberry Pi



Manual

Dr.-Ing. Till Harbaum

June 30, 2020

Project homepage: http://tx-pi.de
Forum: https://forum.ftcommunity.de/

Contents

Chapter 1

Introduction

The ft HAT is an addon for the popular Raspberry Pi computer. It provides electrical compatibility between the world of the fischertechnik construction toy with its 9 volt powered motors, lamps etc and the Raspberry Pi.

1.1 Raspberry Pi

The Raspberry Pi is a full featured credit card sized Linux computer sold at a very low price. It's primarily targeted at the educational market but due to its low price and simple usage it has been widely adopted by the maker community.

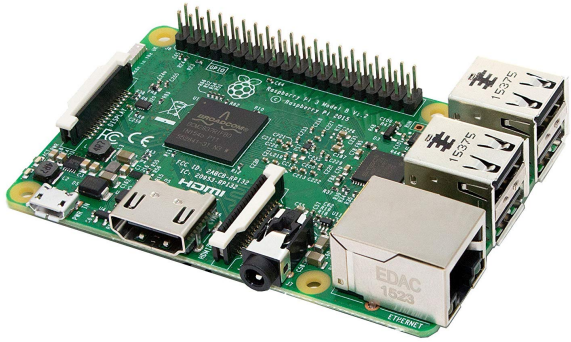


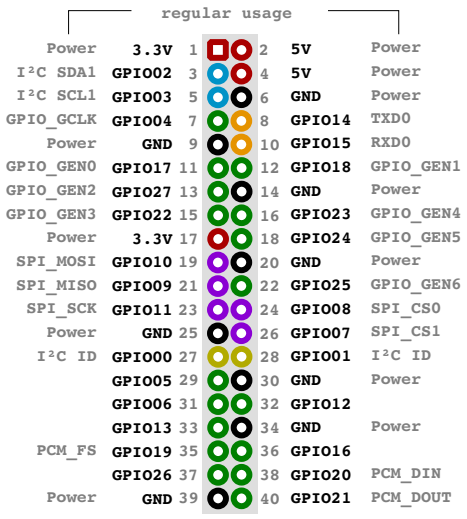
Figure 1.1: The Raspberry Pi version 3

The Raspberry Pi is being sold since 2012 and has reached its fourth version in 2019. The most popular versions of the Raspberry Pi include a multi core gigahertz arm CPU, up to 4GB RAM, several USB connectors incl. USB 3.0, Ethernet, WiFi and Bluetooth and HDMI. They can be powered from ubiquitous USB power supplies and are very affordable.

1.1.1 GPIO port

Unlike most regular computers the Raspberry Pi includes a so-called 40 pin GPIO header (General Purpose Input and Output) as depicted in figure 1.2 which allows to connect arbitrary hardware ranging from various sensors over small touchscreen displays to powerful motor drivers and similar.

The GPIO port allows to extend the Raspberry Pi using so-called HAT's. HAT stands for "Hardware attached on top" and describes hardware devices which have the mechanical footprint of the Raspberry Pi and which can be plugged directly into the GPIO connector and which then sit on top of the Raspberry Pi. Some of these boards can even be stacked so more than one HAT can be connected at a time. The ability to use more than one HAT requires careful selection of the HATs as the connections on the GPIO port can usually not be shared between several HATs. HATs being used simultaneously thus usually need to used separate signals of the GPIO port.



(a) GPIO pinout

(b) A display HAT

Figure 1.2: GPIO extension on the Raspberry Pi

1.1.2 Raspberry Pi compared to fischertechnik TXT controller

Since 2014 fischertechnik is selling the TXT controller, a Linux based computing device with built-in 2.4" touchscreen and fischertechnik compatible power supply, inputs and outputs. Lego is also selling the similar EV3 controller for their system.

The following table compares the EV3, the TXT, the Raspberry Pi4 and the TX Pi[1].

	EV3	TXT	Raspberry Pi4	TX Pi
CPU type	TI-AM1808-Sitara	TI-AM3359-Sitara	Broadcom BCM2711	
Core types	32 bit Cortex A8	32 bit Cortex A8	64 bit Cortex A72	
No of CPU Cores	1	1	4	
CPU clock	300 MHz	600 MHz	1500 MHz	
Memory type	DDR2	DDR3	DDR4	
Memory size	64 MB	256 MB	4 GB	
On board flash	16 MB	128 MB	-	
USB host	1 * USB 2.0	1 * USB 2.0	2 * USB 2.0, 2 * USB 3.0	
USB device	1 * USB 2.0	1 * USB 2.0	-	
WiFi	-	SDIO: 802.11n 2.4 GHz	SDIO: 802.11ac (2.4 & 5 GHz, 1x1)	
Bluetooth	BT 4.0	BT 4.1 incl. BLE	BT 5.0 incl. BLE	
Ethernet	-	-	Gigabit Ethernet	
Display	178×128 mono	2.4" 240x320 TFT	-	3.5" 320x480 TFT
Touchscreen controller	-	software	-	XPT2046
Display controller	-	ILI9341	-	ILI9486/ILI9488
Power supply	6*AA (LR6)	ft 9V= 2.5A	USB-C 5V 3A	ft 9V= 2.5A
I²C	Lego custom	3.3V	3.3V	3.3V & 5V
9V analogue inputs	3	8	-	-
9V digital inputs	-	4	-	4
9V motor outputs	3	4	-	2
Price est.	200€	200€	35€	100€

The TXT is most useful when it comes to control complex fischertechnik models and when focus lies on the interconnection with many fischertechnik components.

The Raspberry Pi on the other hand has a huge advantage with respect to computing power. Whenever the focus lies on computing the Raspberry Pi is the right choice. If more fischertechnik compatible connections are required in a Raspberry Pi setup then one or more ftDuino's[2] can be attached to the Raspberry Pi via USB or I²C.

[1] TX Pi = Raspberry Pi4 + touch display + ft HAT
[2] ftDuino, http://ftduino.de

1.2 What's in the package?

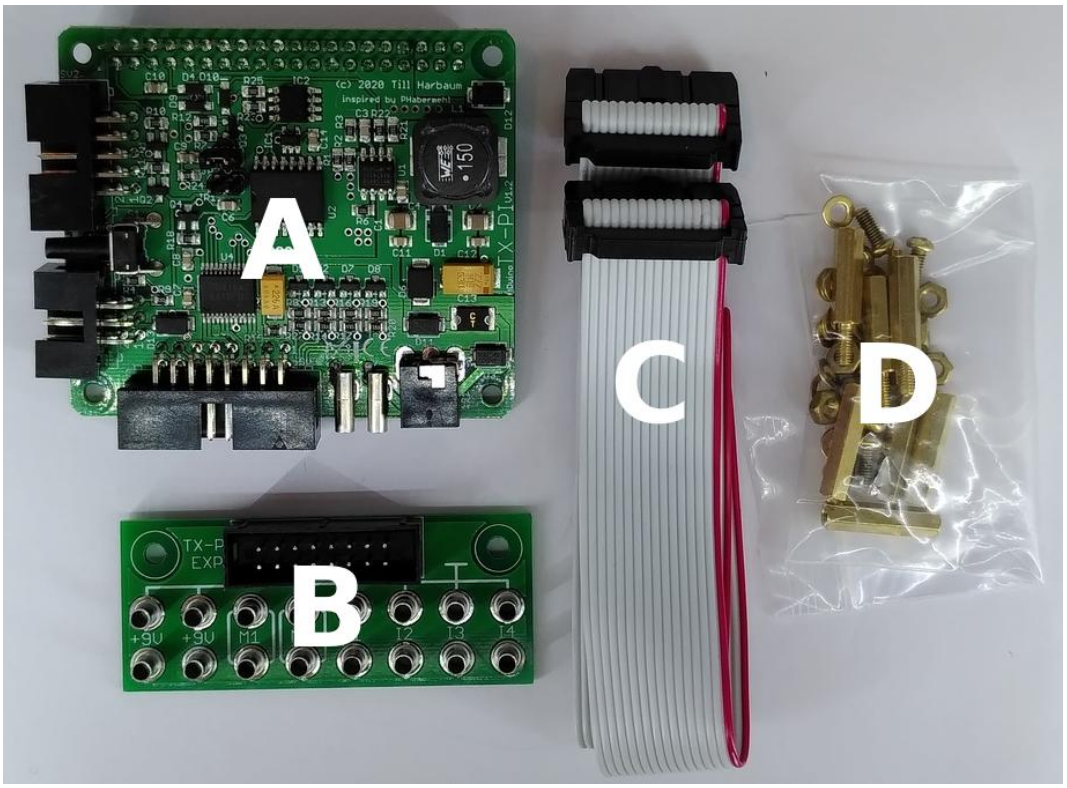


Figure 1.3: ft HAT package contents

The ft HAT comes with:

A: the ft HAT itself

B: the ft HAT breakout board

C: the ft HAT extension cable

D: a set of screws and bolts for the ft HAT

The extension cable connects the ft HAT to the breakout board. The screw and bolt set allows to firmly mount the ft HAT on top of the Raspberry Pi and optionally supports a display.

Chapter 2

The ft HAT

The ft HAT is a so called "HAT" for the Raspberry Pi. HAT stands for "Hardware attached on top" and describes a piece of hardware which can be attached to the top of the Raspberry Pi using its 40 pin GPIO header.

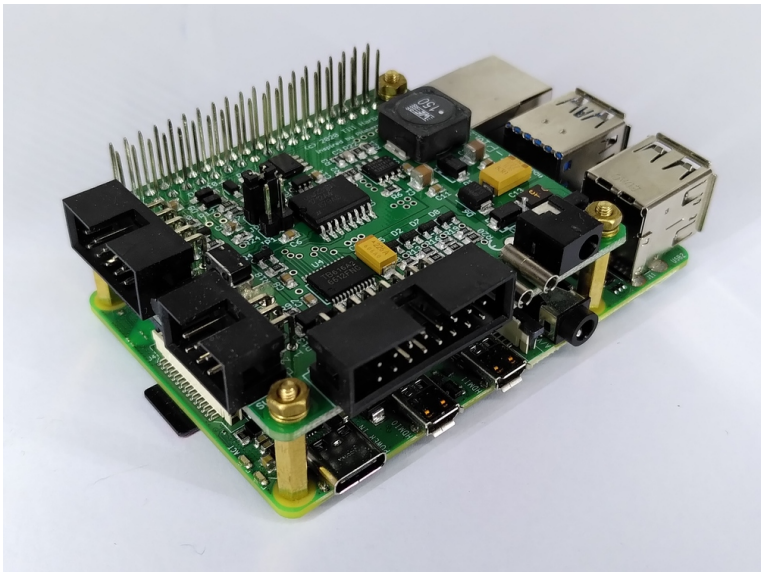


Figure 2.1: The ft HAT mounted on a Raspberry Pi

The ft HAT attaches to the top of the GPIO header and extends the pins on its own top to allow for further HATs to be stacked on top of the ft HAT. The ft HAT makes intensive use of the signals exposed by the Raspberry Pi on its 40 pin expansion connector. Care has thus to be taken if multiple HATs are being used at once.

The ft HAT was designed to address the following topics when using the Raspberry Pi in combination with the fischertechnik construction toy:

- The Raspberry Pi is typically powered via a USB connector from a 5V power source like a phone charger. The ft HAT allows the Pi to be powered from a regular 9V fischertechnik power source instead.
- The Raspberry Pi cannot control it's own power source. This means that a raspberry pi cannot completely switch off by itself. The ft HAT makes the power supply controllable by the Raspberry Pi allowing it to save power e.g. in battery driven setups.
- The GPIO pins of the Raspberry Pi are not electrically compatible with fischertechnik sensors and actors. The ft HAT gives the Raspberry Pi four fischertechnik compatible digital inputs and two fischertechnik compatible analog motor outputs.
- I^2C sensors are popular in the fischertechnik world as well as in the Raspberry Pi community. The ft HAT provides the Raspberry Pi with two fischertechnik compatible I^2C ports. The 10 pin 3.3V port is compatible with the EXT

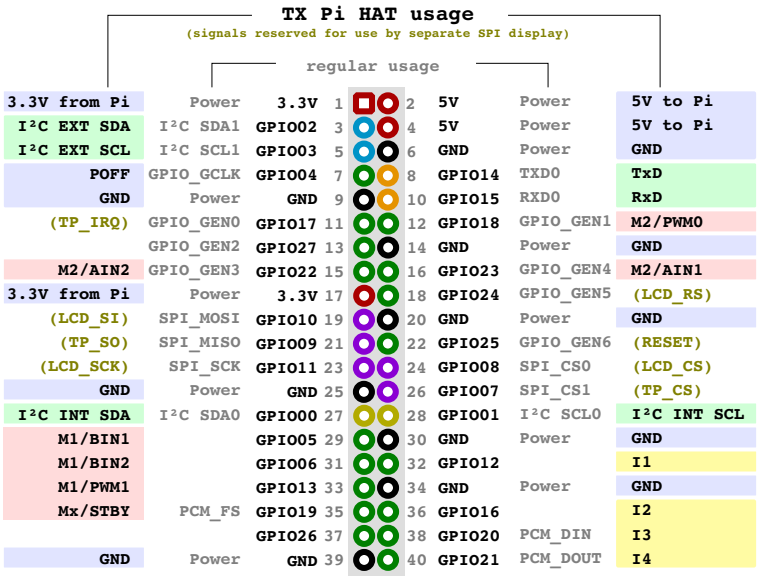


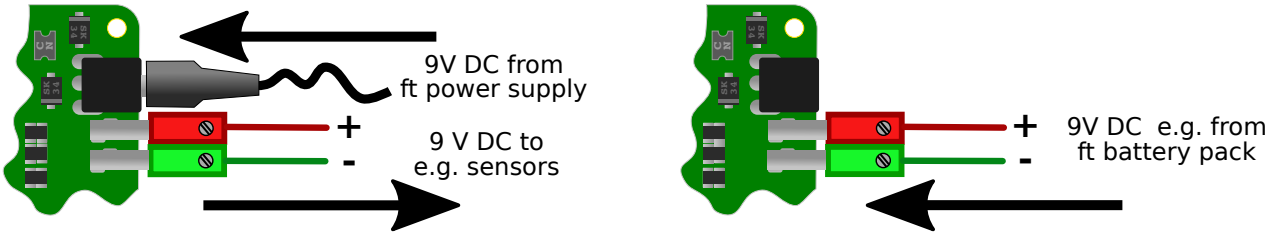
Figure 2.2: ft HAT usage of extension pins on the Raspberry Pi

port of the fischertechnik TXT and like this carries the console UART signals besides the I²C signals. The additional 6 pin port implements a 5V I²C bus and is compatible with the fischertechnik TX controller as well as the ftDuino and various other third party add on's. Both, the 5V and the 3.3V port can be used at the same time. But since they share the same logical I²C bus they also share the same I²C address space.

2.1 Power supply

One of the main features of the ft HAT is it's ability to power the Raspberry Pi from most regular fischertechnik power sources. Power sources can either be connected via the 4mm DC jack or via the pair of fischertechnik 2.5mm jacks left of the DC jack as depicted in figure 2.3.

Warning: The 9V fischertechnik operates with are a potential threat to the Raspberry Pi. The inputs and outputs of ft HAT meant to be connected to 9V devices are protected against these voltages. But the Raspberry Pi itself and some of the components on the ft HAT are not. Be careful when using the Raspberry Pi and/or the ft HAT without case and never let any part of the Raspberry Pi get into direct contact with the 9V from the fischertechnik side as this may damage the Raspberry Pi and the ft HAT.



(a) fischertechnik power supply (b) fischertechnik battery

Figure 2.3: Power supply options of the ft HAT

Supported and tested power sources are:

- fischertechnik power supply 505287 as delivered by fischertechnik for the TXT controller.
- Most stabilized 9V DC power supplies with matching adapters. The power supply should be able to deliver at least 2.5A.
- The fischertechnik 9V battery case using a regular disposable 9V battery.

- The fischertechnik rechargeable battery pack.

Both power supply options are protected against reversed polarity. The DC jack is additionally protected against reverse current flow from the ft HAT into the DC jack.

The 2.5mm fischertechnik jacks are not protected against reversed current flow as they are supposed to be used in both directions. They can either be used to power the ft HAT from an external power source as depicted in figure 2.3(b). Or when powered from the DC jack they can be used to provide power to external devices like sensors as depicted in figure 2.3(a).

Care has thus to be taken to not connect a battery and the DC power source at the same time. In this case the battery would be sourced from the power supply and the battery will likely be damaged.

2.1.1 Power indicator LED

The ft HAT includes a power LED below the power button. This LED will light up whenever the ft HAT's power supply is enabled. When this LED is lit then the attached Raspberry Pi is also powered.

2.1.2 Raspberry Pi power consumption

The latest Raspberry Pi with a few peripherals connected draws at most 3A at 5V under full load. The power supply on the ft HAT was designed to deliver this amount of power. Therefore more than 1.6A on 9V side may be required under full load. In most cases and under regular load the Raspberry Pi typically draws at most around 500mA on 9V via the ft HAT.

E.g. the fischertechnik rechargeable battery pack is rated at 1500mAh. It will thus allow to power the Raspberry Pi for about an hour under full load or nearly three hours under normal load. The fischertechnik power supply 505287 delivers up to 2.5A and easily power the Raspberry Pi even under full load. Some care still has to be taken as additional motors and lamps on the fischertechnik model may also consume significant power.

2.1.3 Power jumper settings

The power supply feature of the ft HAT can be configured using two jumpers on the board. These jumpers control the power-on behavior of the setup:

auto on If this jumper is installed then the Raspberry Pi will automatically be power on whenever power is applied to the ft HAT. If this jumper is removed then the ft HAT will stay off when power is applied and the power button needs to be used to power on the Raspberry Pi.

permanent on Installing this jumper overdrives any power control and if it's set the power supply will be permanently enabled. The Raspberry Pi is then not able to power itself off. This jumper is also needed if the ft HAT is being used stand alone without Raspberry Pi.

Usually the `auto on` jumper is being installed and and the `permanent on` jumper is not installed.

2.1.4 Power on

There are several ways to control the power supply of the ft HAT. All of these can only be used to enable the power supply. If any one of these is enabled then the power supply will be on.

permanent on jumper If this jumper is installed the power supply will be on regardless of any other signal.

Raspberry Pi `GPIO04` This GPIO pin is controlled by the Raspberry Pi and is used to keep power on during regular operation. The Raspberry Pi can be configured to use this pin to release the power supply once the PI is shut down.

Power button The power button enables the power supply as long as the power button is being pressed. In a typical setup the Raspberry Pi will take over power control via `GPIO04` once the power button is released.

Applying power If the `auto on` jumper is installed applying power will enabled the power supply for a short period of time. In a typical setup the Raspberry Pi will take over power control via `GPIO04` once power is stable for some time.

RTC The on board real time clock (RTC) can activate the power supply using its interrupt output. This alarm usually needs to be acknowledged by the Raspberry Pi via I²C. Otherwise this signal stays active and will keep the Raspberry Pi powered even if it's shut down.

If the ft HAT is powered off it goes into a low power state and the power consumption is less than 30 µA. The RTC is still enabled during power down in order to keep the time and to be able to power the system on via its alarm triggered interrupt output.

2.1.5 Raspberry Pi auto power off configuration

The ft HAT gives the Raspberry Pi control over it's own power supply. During regular operation the ft HAT power supply will be powered on for a short period of time by e.g. the power button. This will immediately wake up the Raspberry Pi itself which takes over and keeps the ft HAT power supply on via its `GPIO04` pin. The Raspberry Pi will do this by default and without further configuration due to a Raspberry Pi internal pull-up resistor on this GPIO pin.

The Raspberry Pi can be configured to release this pin on shut down allowing it to disable its own power supply once it has successfully finished its shut down sequence. This feature is not enabled by default. In order to allow the Raspberry Pi to power off on shut down the following line needs to be added to the `/boot/config.txt` file on the Raspberry Pi.

```
dtoverlay=gpio-poweroff,gpiopin=4,active_low=1
```

2.2 I²C

The I²C bus is a technology to connect electronic components. On the ft HAT it's used for two purposes. It's internally used to compontents of the ft HAT to the Raspberry Pi and it's used to connect external components like certain sensors to the ft HAT.

The Raspberry Pi provides access to two I²C busses on its 40 pin GPIO expansion connector. The ft HAT makes use of one of these busses for internal usage and the other one is used for external devices. This separation avoids interference between internal and external devices. The I²C bus `i2c-0` is present on pins 27 and 28 (`GPIO00` and `GPIO01`) while a second I²C bus `i2c-1` is provided on pins 3 and 5 (`GPIO02` and `GPIO03`) of the expansion connector as depicted in figure 2.2.

The I²C busses are named `i2c-0` and `i2c-1`. Bus `i2c-0` is used for the internal devices of the ft HAT while `i2c-1` is being used for the external devices.

2.2.1 Enabling I²C on the Raspberry Pi

By default I²C is *not* enabled on typical Raspberry Pi operating system setups like Raspbian.

i2c-1

The bus `i2c-1` can be enabled using the standard configuration utility `raspi-config`. This is invoked using sudo:

```
sudo raspi-config
```

On the following main menu select `6 - Interfacing Options`. In the submenu displayed afterwards `I2C` can be enabbled as depicted in figure 2.4.

i2c-0

The bus `i2c-0` is not enabled via the I²C configuration in `raspi-config`. Instead an additional line has to be added to the file `/boot/config.txt` file on the Raspberry Pi in order to enable it:

```
dtparam=i2c_vc=on
```

Afterwards a reboot of the Raspberry Pi is required before both busses are available. For initial tests the tool `i2cdetect` needs to be installed using e.g. the following command:

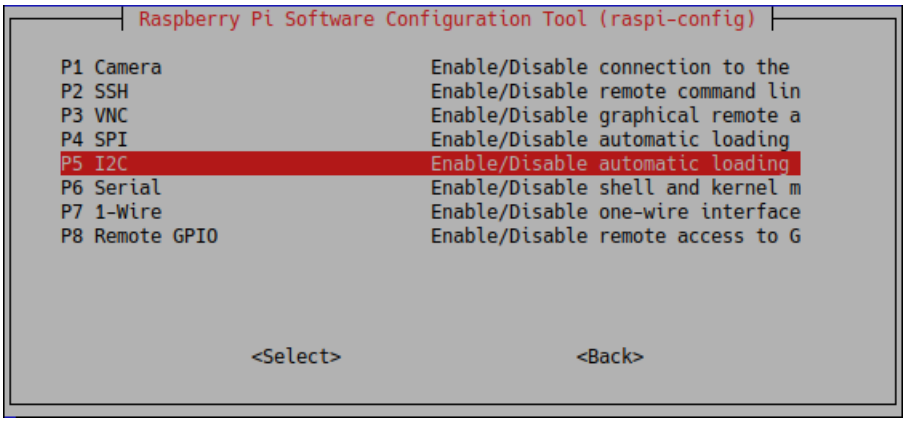


Figure 2.4: Enabling `i2c-1` in the `raspi-config` tool

```
sudo apt-get install -y i2c-tools
```

Once installed `i2cdetect` can be used to check the availability of both busses:

```
$ i2cdetect -l
i2c-0   i2c             bcm2835 I2C adapter                     I2C adapter
i2c-1   i2c             bcm2835 I2C adapter                     I2C adapter
```

Furthermore this tool can be used to check for the internal and external devices on the two busses.

On the internal bus `i2c-0` two devices should show up under addresses `0x50` and `0x68`:

```
$ sudo i2cdetect -y 0
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: 50 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

These are the internal real time clock at address `0x68` and the configuration EEPROM under address `0x50`.

The external bus can ales be check for attached devices. Typically nothing will be connected at this point and the result would not yield any addresses. But with e.g. the fischertechnik environmental sensor and the fischertechnik combi sensor attached the result would look like this:

```
$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: 10 -- -- -- -- -- -- -- 18 -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
```

The fact that four devices show up in this case is caused by the fact that the combi sensor is a combination of three devices showing up at addresses `0x10`, `0x18` and `0x68`. The envronmental sensor shows up at address `0x76`.

Also note that one part of the combi sensor shows up at the same addess `0x68` on `i2c-1` that is used by the internal RTC on `i2c-0`. While it would technically feasible to use only one bus for all internal and external devices these two devices would have their addresses colliding and thus rendering both devices inaccessible.

2.2.2 External I²C ports

The ft HAT exposes `i2c-1` port via two connectors as depicted in figures 2.5 and 2.7. The two connectors are compatible to the ports provided by the original fischertechnik TXT and TX controllers as well as the I²C port of the ftDuino.

2.2.3 3.3V I²C port

The ten pin I²C port of the ft HAT depicted in figure 2.5 is compatible to the EXT port of the fischertechnik TXT controller.

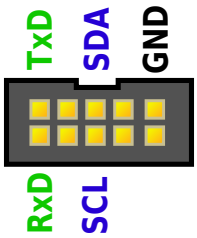


Figure 2.5: The 3.3V I²C port of the ft HAT

All signals on this port use a 3.3V signal level and should be connected to 3.3V signals only. Like the EXT connector of the TXT this port carries a I²C bus as well as the console TxD (transmit data) and RxD (receive data) of a serial console. Both the I²C signals as well as the console signals are connected directly to the according pins of the Raspberry Pi's expansion header. Over voltages and short circuits on these signals may therefore harm the Raspberry Pi.

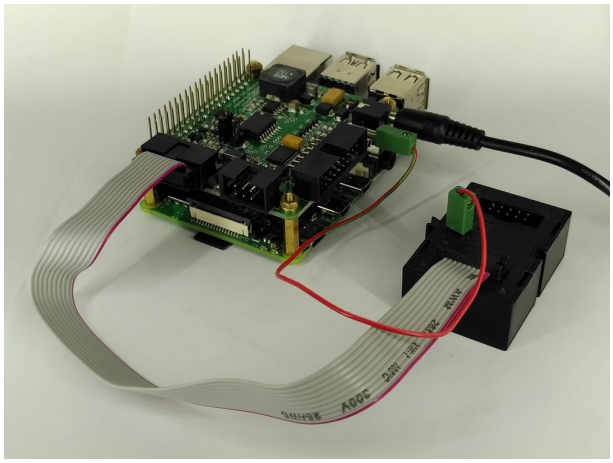


Figure 2.6: Combi sensor connected to the ft HAT

The fischertechnik sensors typically require an external 9V power supply. This can be taken from the ft HAT as explained in figure 2.3(a) and as shown in figure 2.6.

The serial console

The ft HAT just like the fischertechnik TXT controller provides access to the internal serial console of the Linux system through pins 9 and 10 of the ten pin I²C port. On both systems a adapter using e.g. a USB to 3.3v UART converter may be used to get access to the serial console of the TXT or the Raspberry Pi.

This console usually runs at 115200 bit/s and can be accessed from the PC side using regular terminal emulator software like Teraterm (Windows) or Minicom (Linux).

2.2.4 5V I²C port

The six pin I²C port of the ft HAT depicted in figure 2.7 is compatible with the EXT port of the fischertechnik TX controller and the I²C port of the ftDuino. It is connected via a level shifter to the same I²C bus `i2c-1` as the ten pin TXT compatible port. The level shifter provides 5 volt signal levels on the I²C signals. This port should therefore be used whenever a 5V I²C device is to be connected to the ft HAT.

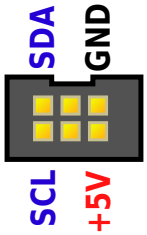


Figure 2.7: The 5V I²C port of the ft HAT

The same logical I²C bus is present on both connectors. From the Raspberry Pi's point of view these port are indistinguishable and they e.g. share the same address space.

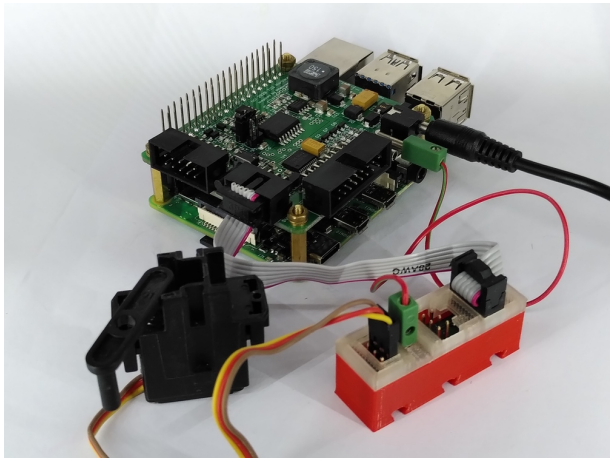


Figure 2.8: Third party mini servo adapter connected to the ft HAT

The six pin port also provides a 5V power source. This port can source up to 250mA for external devices like sensors or e.g. a ftDuino connected via I²C. The 5V connection on the six pin port does now allow to power the ft HAT or the attached Raspberry Pi. A protection diode makes sure that any external power source connected to this port is not being used. This protects any potential external power source like the ftDuino from overload by the up to 3A current the Raspberry Pi draws under normal operating conditions.

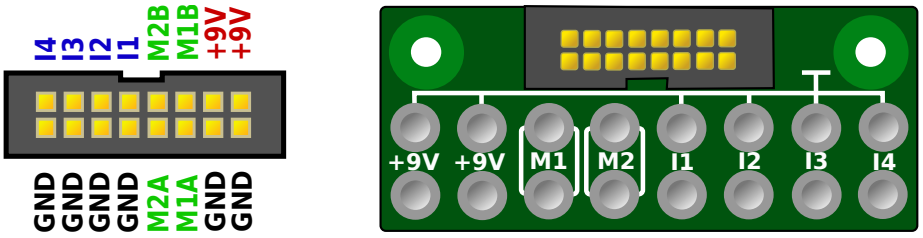
2.3 9V inputs and outputs

The ft HAT was designed to integrate the Raspberry Pi into the fischertechnik system. Besides a fischertechnik compatible power supply it also provides a few fischertechnik compatible inputs and output which can be used to connect switches, lamps, motors and similar devices from the fischertechnik system with the ft HAT.

The ft HAT was designed to be stacked on top of the Raspberry Pi and below a optional display add-on. Thus connectors can only be placed on the sides of the ft HAT which significantly limits the available space.

Thus all fischertechnik compatible connections have been moved to a single 16 pin IO connector as depicted in figure 2.9.

A cable carrying a matching connector on one end and regular 2.5mm fischertechnik plugs on the other can directly be connected to this port.



(a) ft HAT IO connector (b) optional breakout board

Figure 2.9: fischertechnik IO connections on the ft HAT

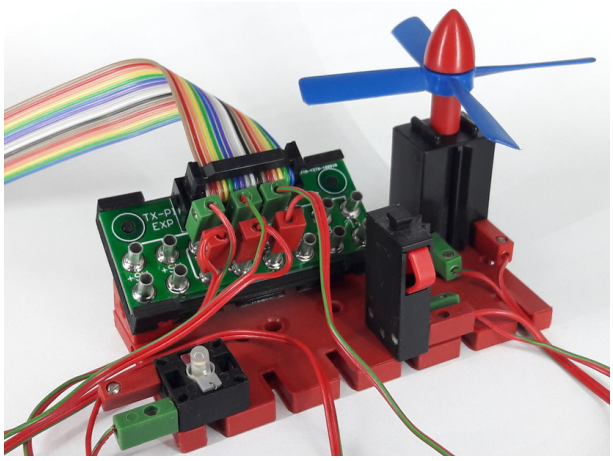


Figure 2.10: Connecting fischertechnik to the breakout board

To make usage easier the ft HAT comes with a breakout board carrying the regular fischertechnik sleeves which can be used with the regular 2.5mm fischertechnik plugs.

2.3.1 Inputs

The ft HAT features four fischertechnik compatible inputs `I1`, `I2`, `I3` and `I4`. These are digital inputs and can be used to connect buttons, switches and the fischertechnik photo transistor. These inputs are digital only (limited to on and off) and cannot read analogue voltages or resitances since the Raspberry Pi does not provide analog inputs on its GPIO port.

The inputs are protected against over voltage and can directly be connected to any fischertechnik power source. The regular use case is to connect the input against ground.

Input	Pin	GPIO
`I1`	32	`GPIO12`
`I2`	36	`GPIO16`
`I3`	38	`GPIO20`
`I4`	40	`GPIO21`

The four inputs `I1` to `I4` are mapped to the Raspberry Pi's pins `GPIO12`, `GPIO16`, `GPIO20` and `GPIO21` as depicted in figure 2.2.

2.3.2 Outputs

The ft HAT features two fischertechnik compatible motor outputs `M1` and `M2`. Each motor output consists of two individual outputs. The outputs can both be adjusted in polarity allowing to e.g. control the motors direction and the speed can be adjusted using a separate PWM signal.

The motor outputs are driven by a TB6612 dual motor bridge. The input pins of the TB6612 are mapped to the Raspberry Pi's GPIO pins as follows:

	M1	M2
IN1	BIN1/GPIO05	AIN1/GPIO23
IN2	BIN2/GPIO06	AIN2/GPIO22
PWM	PWM1/GPIO13	PWM0/GPIO18
STBY	GPIO19	

The state of the two motor outputs can be controlled from the Raspberry Pi by driving GPIO05, GPIO06 and GPIO13 for M1 and GPIO23, GPIO22 and GPIO18 for M2. The STBY signal is shared by both outputs and is controlled from the Raspberry Pi's GPIO19.

The four pins on one channel control the state of one motor output as follows:

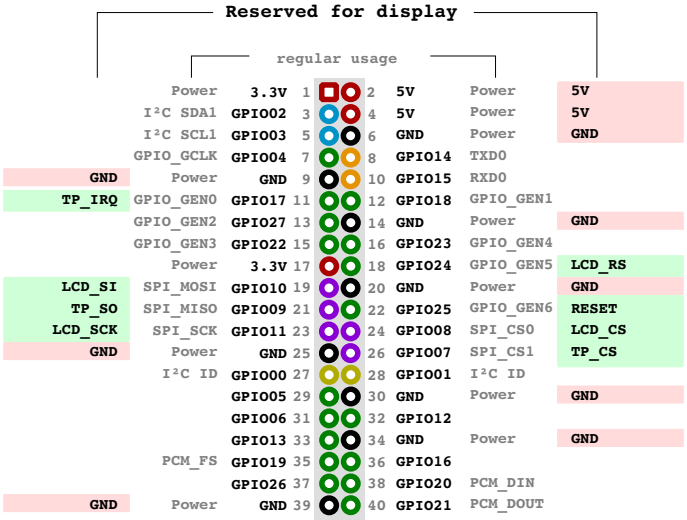
STBY	IN1	IN2	PWM	state
0	x	x	x	standby (high impedance)
1	0	0	1	stop (high impedance)
1	1	1	x	short brake (both outputs low)
1	0	1	1	turn CCW (one output low, one high)
1	0	1	0	short brake (both outputs low)
1	1	0	1	turn CW (one output high, one low)
1	1	0	0	short brake (both outputs low)

More information about programming the inputs and outputs is being given in section 3.

2.4 Using a touch display HAT

The ft HAT does not necessarily need to be used with a display. It can be used headless (without any display) or like most regular Raspberry Pi setups it may also be used with a regular screen connected to the HDMI.

But man use cases make more sense a screen. Many screens available for the Raspberry Pi can also be made to work with the ft HAT. But there are several limitations:



(a) GPIO pins reserved for display use

(b) Bottom of Waveshare 3.5" display

Figure 2.11: GPIO usage with display

As shown in figure 2.11 several pins of the GPIO connector have been reserved for use with a display HAT. This is due to the fact that most GPIO pins are expected to be exclusively used by a certain hardware. It thus has to be asured that the pins used by a display are not used by another addon and vice versa.

The different displays available for the Raspberry Pi differ significantly in the way they make use of the Raspberry Pi's GPIO pins. The popular Waveshare 3.5" touchscreen displays with 320x480 pixels resolution have been choosen as a reference. Several 3.5 inch third party displays are also compatible with this setup. Displays known to be compatible include:

Waveshare 3.5inch RPi LCD (A) Various compatible third party displays are available.

Waveshare 3.5inch RPi LCD (B) This display differs slightly in the display technology used and it requires a different driver but it is otherwise compatible with the (A) version and with the default pin usage of the ft HAT.

Waveshare 4.0inch RPi LCD (A) This display is slghtly bigger than the 3.5 inch version but is otherwise compatible and features the same resolution and the same pin usage.

Waveshare 3.5inch RPi LCD (C) This is a high speed version of the (A) model. The **MHS-3.5inch RPi Display** is known to be compatible with this.

In general all displays able to work with the default drivers for the Waveshare displays mentioned above are compatible with the pin usage of the ft HAT.

Figure 2.12: A 3.5 inch touch screen mounted on top of the ft HAT

Chapter 3

Programming the ft HAT

The ft HAT extends existing interfaces of Raspberry Pi to make them accessible for fischertechnik usage. Part of these extensions are accessed via I²C and part of them is accessed using the Raspberry Pi's GPIO pins. This chapter explains how to access these features.

Most of the code examples in this chapter use either Linux command line tools or are written in the Python programming language. Other programming languages can usually be used equally well. The internet usually provides examples showing how to access the I²C busses or the GPIO ports of the Raspberry Pi and used by the ft HAT in other languages. The Python examples of this section can be used as a template for other languages as well.

3.1 I²C

The two I²C busses used by the ft HAT on the Raspberry Pi are `i2c-0` and `i2c-1` which are both available on the 40 pin GPIO port as depicted in figure 2.2. Before these ports can be accessed by software they need to be enabled as explained in section 2.2.1.

The ft HAT itself already contains on-board I²C peripherals. Further I²C peripherals can be connected externally either via the 10 pin 3.3V port or via the 6 pin 5V port.

3.1.1 Internal real time clock

As an internal device the DS3231 real time clock (RTC) is connected to the internally used I²C bus `i2c-0` and shows up under address 0x68. Since the DS3231 is supported by a standard I²C kernel driver it can be made available to the system using the Linux on-board tools and drivers.

Using the kernel driver

Using the kernel driver has the major disadvantage that the alarm controlled wakeup/powerup mechanism of the ft HAT is not suppored by this. In order to use this feature follow the programming instructions in the next section instead.

Since there is no reliable way of auto detecting hardware on the I²C bus the Linux kernel needs to be told that there is a DS3231 device at address 0x68 of bus `i2c-0` using the following commands:

```
$ sudo modprobe rtc_ds1307
$ echo ds3231 0x68 | sudo tee -a /sys/class/i2c-adapter/i2c-0/new_device
ds3231 0x68
```

Afterwards the RTC should show up in the sys file system and the date and time can be read:

```
$ cat /sys/class/rtc/rtc0/date
2000-01-01
$ cat /sys/class/rtc/rtc0/time
```

```
00:10:15
```

The command `hwclock` can then be used to read the time from the RTC:

```
$ hwclock -r
2000-01-01 01:11:08.923619+0100
```

It's obvious that the RTC does not know the correct time. In most cases the Raspberry Pi will know the correct time already from a network service. This time can be written into the RTC:

```
$ hwclock -w
$ hwclock -r
2019-07-01 09:22:48.677249+0200
```

The time can be moved from the RTC back to the system time using the `hwclock -s` command.

Please be aware that the RTC does not come with a seperate power supply. If the ft HAT is disconnected from a power supply the time and date stored on the RTC will be reset to 2000-01-01 00:00:00 UTC. But the date and time is kept while the ft HAT is being connected to a power source even if the main power supply inside the ft HAT is switched off and while the Raspberry Pi is powered down. This allows to use the RTC to power up the Raspberry Pi at a given time.

Unloading the kernel driver

Any direct programming approach from e.g. Python will not work if the kernel driver is loaded and has gained exclusive access to the chip.

In this case the `i2cdetect -y 0` command will show `UU` instead of 68 if the address is blocked by a system driver:

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

In this case the driver can be unloaded:

```
$ echo 0x68 | sudo tee -a /sys/class/i2c-adapter/i2c-0/delete_device
```

Afterwards the RTC is available for direct programming and the alarm can be set from programs like the Python example.

Unfortunately the Linux kernel driver used this way does not provide access to the alarm functions of the RTC which are required to wake up the Raspberry Pi timed by the RTC alarm. Thus using the linux kernel driver for the RTC is not recommended. Instead the RTC should be accessed directly bypassing any RTC driver as outlined in the following section.

Programming the RTC in Python

Python like most other programming languages provides methods to directly access the I^2C busses and thus send arbitrary commands to any connected I^2C device like the RTC. A matching Python library for the DS3231 RTC can e.g. be found at github[1].

This library can be used from within any Python program:

```
ds3231 = DS3231.DS3231(0, 0x68)
# set RTC time from system time
ds3231.write_now()
# set alarm one minute in the future
```

[1] DS3231.py: https://github.com/harbaum/cfw-apps/tree/master/packages/tx-pi-hat-test

```
ds3231.setAlarm(datetime.datetime.now() + datetime.timedelta(minutes=1))
# shut down pi to let it wake up again
call(["sudo", "poweroff"])
```

A second snippet of code is needed once the RTC has woken up the Raspberry Pi since the RTC will keep the alarm signal engaged which in turn would prevent future attemps to power the Raspberry Pi down again.

```
# after RTC triggered powerup the RTC needs to be acknowledged
# to release the power supply
ds3231.ackPendingAlarm()
```

Acknowleding the alarm from the application itself us cumbersome as the applications needs to be started explicitely after an alarm triggered reboot. It may thus be more convenient to acknowledge the alarm at boot time from the Linux boot scripts. This way any pending alarm will be cleared during boot and neither the user nor any software has to care for pending alarms which may prevent the ft HAT from powering the Raspberry Pi down.

RTC alarm auto acknowledge

The RTC alarm acknowledge can be automated by adding the command `i2cset -y 0 0x68 0x0f 0x00` to the end of the `/etc/rc.local` file e.g. as follows:

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true

if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi

# ack pending RTC wakeup
/usr/sbin/i2cset -y 0 0x68 0x0f 0x00

exit 0
```

This will clear the alarm bit on every boot and will this allow to power down afterwards.

3.1.2 Internal EEPROM memory

The usage of the EEPROM is optional. It is not required for proper operation of the ft HAT. However, it can be used to store helpful system information that gives the Raspberry Pi some additional information about the HAT already at a very early boot stage. The ft HAT is being shipped with useful contents already stored in the EEPROM. Thus there is usually no need to touch the programming of this chip.

In order to be able to access the EEPORM its I²C bus may need to be enabled as explained in section 2.2.1.

The optional on board EEPROM can be accessed by the Raspberry Pi via the same I²C port as the RTC. The EEPROM can be used to store additional configuration information to be used by the Raspberry Pi at boot time to setup the system.

The EEPROM is optional and is usually not required[2].

Some ft HAT specific content of the EEPROM could be stored in file eeprom_settings.txt like this:

```
##############################################################################
# Vendor info

# 128 bit UUID.
product_uuid 392b690a-c74d-4049-8a37-2f2789a7946e

# 16 bit product id
product_id 0x4711

# 16 bit product version
product_ver 0x0000

# ASCII vendor string  (max 255 characters)
vendor "Till Harbaum"

# ASCII product string (max 255 characters)
product "fischertechnik HAT"

# If board back-powers Pi via 5V GPIO header pins:
# 2 = board back-powers and can supply the Pi with a minimum of 2A
# If back_power=2 then USB high current mode will be automatically
# enabled on the Pi
back_power 2
```

The EEPROM can be read by e.g. the following command:

```
$ sudo eepflash.sh -r -f=dump.eep -t=24c256 -d=0 -a=50
This will attempt to talk to an eeprom at i2c address 0x50 on bus 0. Make sure there is an
eeprom at this address.
This script comes with ABSOLUTELY no warranty. Continue only if you know what you are doing.
Do you wish to continue? (yes/no): yes
Reading...
32256 Bytes (32 kB, 32 KiB) kopiert, 3,01154 s, 10,7 kB/s
64+0 Datensätze ein
64+0 Datensätze aus
32768 Bytes (33 kB, 32 KiB) kopiert, 3,06013 s, 10,7 kB/s
Closing EEPROM Device.
Done.
```

Afterwards the 32768 bytes contents of the EEPROM are stored in the file dump.eep.

To write the aforementioned data from eeprom_settings.txt to the EEPROM the textual representation first has to be converted to bonary:

```
$ eepmake eeprom_settings.txt eeprom_settings.eep
Opening file eeprom_settings.txt for read
Done reading
Writing out...
Done.
```

Afterwards the data can be written to the EEPROM:

```
$ sudo eepflash.sh -w -f=eeprom_settings.eep -t=24c256 -d=0 -a=50
This will attempt to talk to an eeprom at i2c address 0x50 on bus 0. Make sure there is an
eeprom at this address.
This script comes with ABSOLUTELY no warranty. Continue only if you know what you are doing.
```

[2] Raspberry Pi HAT EEPROM data: https://github.com/raspberrypi/hats/tree/master/eepromutils

```
Do you wish to continue? (yes/no): yes
Writing...
0+1 Datensätze ein
0+1 Datensätze aus
114 Bytes kopiert, 0,600914 s, 0,2 kB/s
Closing EEPROM Device.
Done.
```

If everything worked correctly the HATs EEPROM config will be detected after next boot and shows up under /proc:

```
$ cat /proc/device-tree/hat/product
fischertechnik HAT
```

3.1.3 fischertechnik environmental sensor 167358

The environmental sensor from the fischertechnik home automation and IoT kits is based on the Bosch BME680 sensor.

Figure 3.1: The fischertechnik environmental sensor 167358

A matching Python library is available for Raspbian and can be installed:

```
sudo apt-get install python3-bme680
```

This library can be used in a Python program as follows[3]:

```
import bme680
import time

sensor = bme680.BME680()

sensor.set_humidity_oversample(bme680.OS_2X)
sensor.set_pressure_oversample(bme680.OS_4X)
sensor.set_temperature_oversample(bme680.OS_8X)
sensor.set_filter(bme680.FILTER_SIZE_3)

sensor.set_gas_status(bme680.ENABLE_GAS_MEAS)
sensor.set_gas_heater_temperature(320)
sensor.set_gas_heater_duration(150)
sensor.select_gas_heater_profile(0)

while True:
    if sensor.get_sensor_data():
        output = "{0:.2f} C,{1:.2f} hPa,{2:.2f} %RH".format(↩
           sensor.data.temperature, sensor.data.pressure, sensor.data.humidity)

        if sensor.data.heat_stable:
            print("{0},{1} ohms".format(output, sensor.data.gas_resistance))
        else:
```

[3] The small arrow ↩ indicates that the line does not end there but continues with the contents of the next line

```
            print(output)

    time.sleep(1)
```

The output will include temperatures, pressure and humidity:

```
$ python3 ./bme680.py
30.60 C,1005.56 hPa,27.84 %RH
30.63 C,1005.56 hPa,27.85 %RH,35178.86404694175 ohms
30.69 C,1005.59 hPa,27.82 %RH,49038.265142913355 ohms
30.76 C,1005.61 hPa,27.73 %RH,57726.21732917038 ohms
30.81 C,1005.60 hPa,27.65 %RH,63817.64696100763 ohms
30.86 C,1005.58 hPa,27.57 %RH,68219.75818501839 ohms
...
```

The ohms output is a value that can be used to determine air quality. Examples for this can be found on github[4].

3.1.4 fischertechnik combi sensor 158402

The combi sensor sold by fischertechnik under part number 158402 is based on the Bosch Sensortec BMX055 integrated circuit.

Figure 3.2: The fischertechnik combi sensor 158402

This integrated circuit bascially combines three seperate units into one case. These three units are a accelerometer, a gyroscope and a magnetometer (compass). All three are accessible under their own I^2C addresses:

BMX055 component	I^2C address
Accelerometer	0x18
Gyroscope	0x68
Magnetometer	0x10

Using the combi sensor thus shows up like this when being scanned using `i2cdectect`:

```
$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: 10 -- -- -- -- -- -- -- 18 -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

The fact that this gyroscope uses address 0x68 is the main reason why the ft HAT's internal RTC is connected to the Raspberry Pi's other I^2C bus `i2c-0` as it uses the same address 0x68 and having both devices on the same bus would let their addresses collide and making them inaccessible.

[4] bme680-python indoor-air-quality.py, https://github.com/pimoroni/bme680-python/blob/master/examples/indoor-air-quality.py

Python examples for this can be found at github[5]. However, to make this work with the fischertechnik combi sensor on a Raspberry Pi we had to comment line 78 like so:

```
# BMX055 Mag address, 0x10(16)
# Select Mag register, 0x4B(75)
#               0x83(121)       Soft reset
#bus.write_byte_data(0x10, 0x4B, 0x83)
```

After this line is commented this python program will read the sensor data:

```
$ python ./BMX055.py
Acceleration in X-Axis : 2
Acceleration in Y-Axis : -1
Acceleration in Z-Axis : 1018
X-Axis of Rotation : -14
Y-Axis of Rotation : -24
Z-Axis of Rotation : 44
Magnetic field in X-Axis : -34
Magnetic field in Y-Axis : -74
Magnetic field in Z-Axis : -133
```

A comlete python example from the TX-Pi project (see chapter 4) making use of the BMX055 can be founf in the CFW apps repository[6]. It ws written for the fischertechnik combi sensor and runs on the fischertechnik TXT controller as well as the Raspberry Pi. Its output is depicted in figure 3.3.

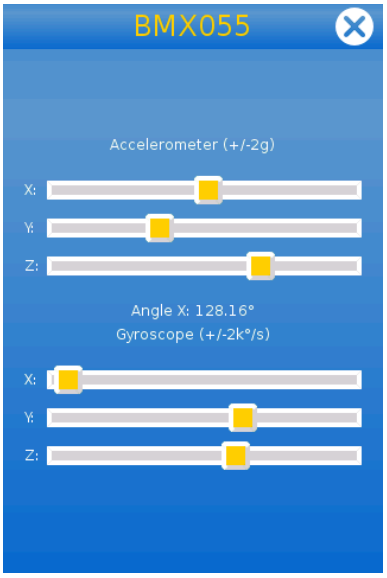


Figure 3.3: BMX055 CFW app

3.1.5 Third party sensors

Besides the sensors sold by fischertechnik the I²C bus has also become popular in the aftermarket. The ft HAT allows to connect these directly if they follow the connection scheme used by the fischertechnik TXT or TX controllers.

In the following image are the two fischertechnik sensors connected to the 10 pin 3.3V port while a OLED display and a mini servo adapter are connected to the 6 pin 5V port. All devices show up on bus `i2c-1`:

```
$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
```

[5] BMX055 python demo: https://github.com/ControlEverythingCommunity/BMX055/blob/master/Python/BMX055.py
[6] BMX055 CFW app: https://github.com/harbaum/cfw-apps/blob/master/packages/bmx055/bmx055.py

```
10: 10 11 -- -- -- -- -- -- 18 -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- 3c -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
```

The I²C bus is popular in the Raspberry Pi community and thus drivers and examples can easily be found in the internet for most common I²C peripherals.

Example code for the 128*64 OLED display showing up at address 0x3c can e.g. found in the Adafruit repository [7].

The mini servo adapter on the other hand is a community project aimed at the ftDuino[8] and thus does not come with a ready to use library for the Raspberry Pi. But the standard I²C command line tools can be used to control the mini servo adapter. E.g. the command

```
i2cset -y 1 0x11 0 100
```

will move the servo connected to servo output 0 of the mini servo adapter to position 100. The same can be achieved from within python:

```
$ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import smbus
>>> bus = smbus.SMBus(1)
>>> bus.write_byte_data(0x11,0,100)
>>>
```

This approach works for many simple I²C devices and allows to easily write a python software making use of nearly any I²C chip.

3.2 Programming the fischertechnik inputs and outputs

The four fischertechnik inputs `I1` to `I4` and the two motor outputs `M1` to `M2` are connected directly to the GPIO pins of the Raspberry Pi as explained in sections 2.3.1 and 2.3.2.

Programming them is thus done using the regular methods of dealing with the GPIOs of the Raspberry Pi. The GPIOs of the Raspberry Pi can be controlled from most popular programming languages as well as from the command line.

3.2.1 Command line

Command line usage makes use of the `/sys` file system. E.g. the following command will activate `GPIO12` as input and then read the state of `GPIO12`. Since `GPIO12` is connected with the input `I1` this will read the current state of that input:

```
echo "12" > /sys/class/gpio/export
echo "in" > /sys/class/gpio/gpio12/direction
cat /sys/class/gpio/gpio12/value
```

The other inputs `I2` to `I4` can be evaluated in the same way using 16, 20 and 21 as the GPIO values.

Outputs can be configured in a similar way. The following code will bring the motor outputs out of standby:

```
echo "19" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio19/direction
echo "1" > /sys/class/gpio/gpio19/value
```

[7] Adafruit OLED/SSD1306 python library: https://github.com/adafruit/Adafruit_Python_SSD1306
[8] ftDuino: http://ftduino.de

Enabling M1 to turn full speed several GPIO pins need to be driven according to the tables in section 2.3.2:

```
echo "19" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio19/direction
echo "1" > /sys/class/gpio/gpio19/value

echo "5" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio5/direction
echo "1" > /sys/class/gpio/gpio5/value

echo "6" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio6/direction
echo "0" > /sys/class/gpio/gpio6/value

echo "13" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio13/direction
echo "1" > /sys/class/gpio/gpio13/value
```

3.2.2 Python

Similar can be done in most programming languages. In python reading `I1` via `GPIO12` looks like:

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(12, GPIO.IN)
print("I1:", GPIO.input(12))
```

Again, the other inputs `I2` to `I4` can be evaluated in the same way using 16, 20 and 21 as the GPIO values.

Leaving the standby mode of the motor bridge can be done as follows:

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(19, GPIO.OUT)
GPIO.output(19, GPIO.HIGH)
```

The four pins required to turn the motor `M1` can e.g. set in the following manner:

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)

pins = { 19:GPIO.HIGH, 5:GPIO.HIGH, 6:GPIO.LOW, 13:GPIO.HIGH }
for p in list(pins.keys()):
    GPIO.setup(p, GPIO.OUT)
    GPIO.output(p, pins[p])
```

The example above switches the PWM on `GPIO13` just "on". This means that any motor connected to `M1` will run full speed or that any lamp will light at full brighness.

In order to control the speed the PWM features of the Raspberry Pi can be used as shown below:

```
GPIO.setup(13, self.GPIO.OUT)
pwm = GPIO.PWM(13, 200)    # PWM at 200 Hz
pwm.start(50)              # run output at 50%
```

A complete example

The following is a complete example showing how to use Python to enable motor output `M1` and dim a lamp connected to this port.

```
# m1_dim.py
# Dimming a lamp on M1

import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# enable TB6612 driver chip
GPIO.setup(19, GPIO.OUT)
GPIO.output(19, GPIO.HIGH)

# set mode to clockwise (direction doesn't really matter for the lamp)
GPIO.setup(5, GPIO.OUT)
GPIO.output(5, GPIO.HIGH)
GPIO.setup(6, GPIO.OUT)
GPIO.output(6, GPIO.LOW)

# prepare PWM
GPIO.setup(13, GPIO.OUT)
pwm = GPIO.PWM(13, 50)    # 50 Hz is sufficient for a lamp
pwm.start(0)

duty = 0
step = 10
while True:
    # set duty cycle in %
    pwm.ChangeDutyCycle(duty)
    print("Duty =", duty, "%")
    duty += step
    if duty == 0 or duty == 100:
        step -= step

    time.sleep(0.5)
```

Chapter 4

The TX Pi project

The ft HAT is part of the TX Pi project located under http://tx-pi.de. The TX Pi project aims to integrate the Raspberry Pi with the fischertechnik construction toy. To achieve this goal the TX Pi project provides:

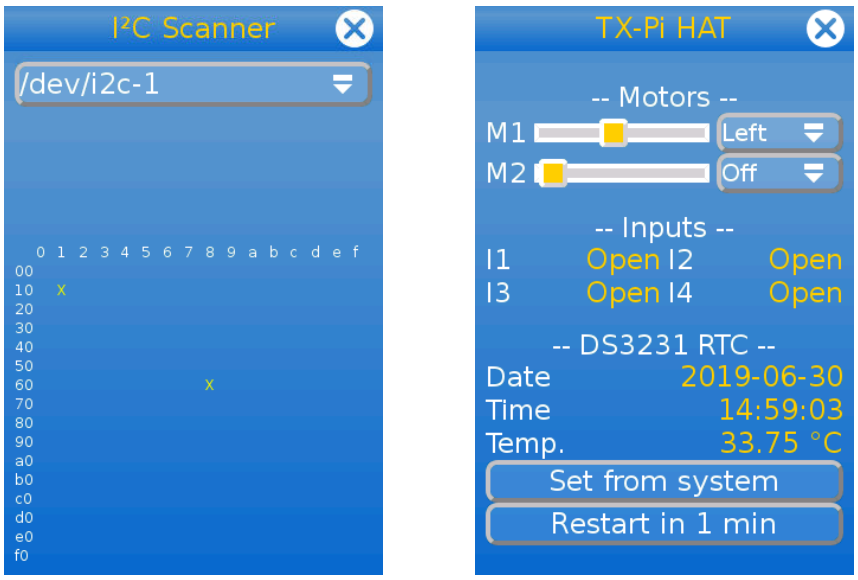
1. Hardware recommendations and custom hardware designs like the ft HAT for the electrical integration
2. Software configurations for the Raspberry Pi suitable to control fischertechnik devices for the software integration
3. Case designs for all major components for the mechanical integration.

4.1 Software and applications

The TX Pi project originates from the fischertechnik community firmware for the TXT-Controller and shares much of its software with this and runs it in top of a standard Raspbian[1] installation.

Please refer to http://tx-pi.de for further details and installation instructions.

The TX Pi project uses many applications written for the TXT. But it also comes with Raspberry Pi specific applications and even some ready to use ft HAT specific ones.



(a) I²C scanner app

(b) TX Pi HAT demo app

Figure 4.1: Apps making use of the ft HAT

[1] Raspbian: https://www.raspberrypi.org/downloads/raspberry-pi-os/

These applications are distributed via Github at https://github.com/ftCommunity/tx-pi-apps.

4.2 Display

The TX Pi setup is designed to be used with a lowrez touch screen stacked on top of the Raspberry Pi. Some of these displays are compatible with the ft HAT and can be used in conjunction with this. See section 2.4 for more details on this.

4.3 Case designs

The TX Pi prject also provides case designs for 3D printed cases to protect the various hardware components as well as making them mechanically compatible with the fischertechnik system.

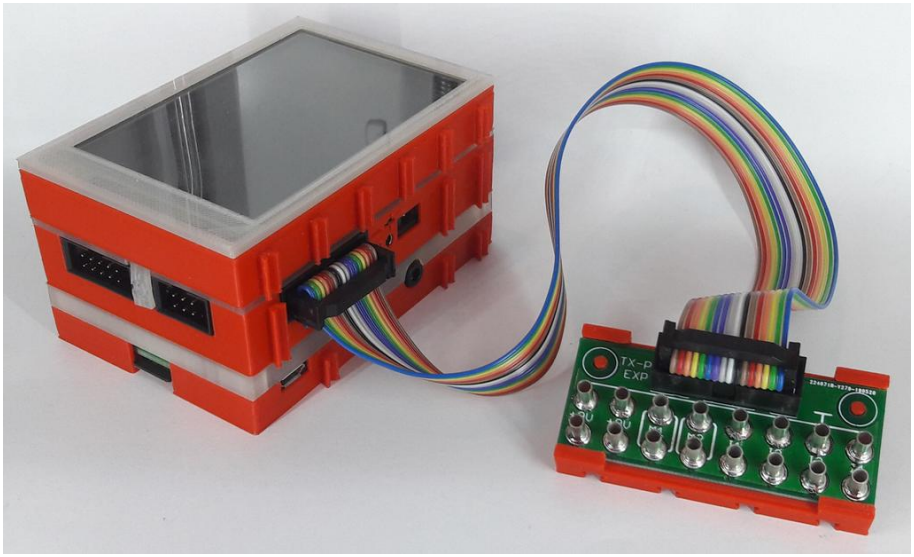


Figure 4.2: A complete TX Pi setup consisting of the Raspberry Pi3, the ft HAT and a 3.5" display

Case designs can be found at https://github.com/ftCommunity/tx-pi/tree/master/cases. This includes cases for the Raspberry Pi itself as well as the ft HAT and some displays.