BigTreeTech PI V1.2

YouTube Video #61

Sponsor

Vielen Dank an folgende Sponsoren für dieses Video:

- **Konstantin Schuller** für das Sponsoring eines BTT Pi 1.2 nebst CAN Modul
- **BigTreeTech** für das Sponsoring eines CAN Moduls, eines 3,5“ Displays und eines ADXL345 Sensormoduls

Links

- Github
  https://github.com/bigtreetech/BTT-Pi
- Images
  https://github.com/bigtreetech/CB1/releases
- Enabling panfrost GPU acceleration on the Bigtreetech PAD7
  https://gist.github.com/adelyser/e814dd7737026765a3ab390ec21f78cc
- https://github.com/bigtreetech/cb1
- https://github.com/bigtreetech/CB1#40-pin-gpio
- https://github.com/So6Rallye/BTT-Pi/blob/master/BIGTREETECH%20Pi%20V1.2%20-%20Board%20Fan%20Pin%20Configuration
- Manual
  https://github.com/bigtreetech/BTT-Pi/blob/master/BIGTREETECH%20Pi%20V1.2%20User%20Manual.pdf
- [HOWTO] 40-pin GPIO & ADLX345 on CB1
  https://github.com/bigtreetech/CB1/discussions/47

Pros / Cons

- \+ 24V direkt rein für kleine Drucker

- + Linux Kernel Quellen und Config File
- + CAN Onboard
- + IR Input

- - keine Anzeige für SD Aktivität
- - kein CSI / DSI
- - schlechte Doku bezüglich SPI / I2C
- - Kernel Updates nur über neues Image

Default User / Passwort

- User : **biqu**
- Passwort : **biqu**

Image Installation

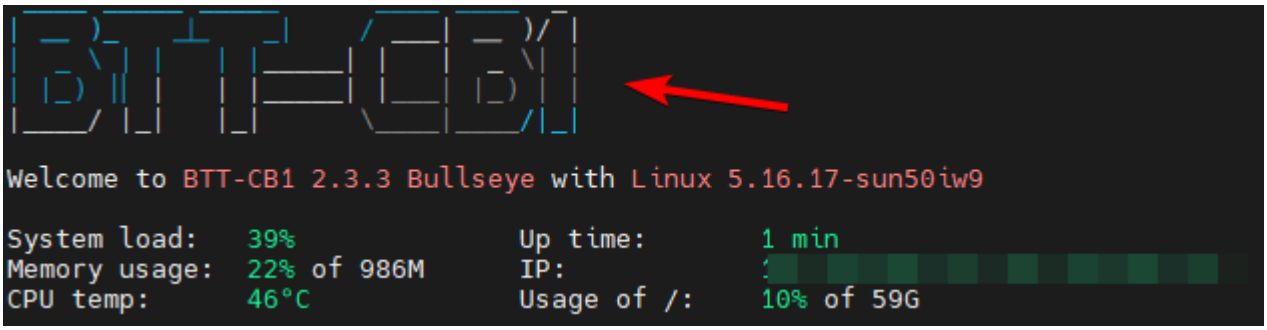
> **ACHTUNG**
> Das BTT Pi 1.2 (wie auch der BTT CB1) laufen **nicht mit dem Raspberry Pi Image**!
> Es muss ein Image von BTT genutzt werden das an den SBC angepasst ist.

- Image Download
  https://github.com/bigtreetech/CB1/releases
  Hier kann der aktuelle Stand vom BTT Image geladen werden. Grundsätzlich reicht für den Betrieb von Klipper das Minimal Image (Stand jetzt wäre das : `CB1_Debian11_minimal_kernel5.16_20230712.img.xz`). In dem größeren Image (`CB1_Debian11_Klipper_kernel5.16_202300712.img.xz`) ist u.A. noch ein Grafiktreiber für den SBC integriert. Klipper läuft mit beiden Images problemlos.
- Image mit dem Raspberry Pi Imager auf eine SD Karte schreiben
  https://www.raspberrypi.com/software/
  - OS Wählen
  - ganz unten auf “Eigenes Image”
  - Hier jetzt die geladene XZ Datei auswählen
  - SD Karte wählen
  - Schreiben (Einstellungen kann man nicht mit schreiben lassen. Die sind nicht kompatibel zu dem Image!)
- Wlan einrichten
  Um das WLan einzurichten muss auf der SD Karte eine Datei angepasst werden:
  - Auf dem Laufwerk “BOOT” die Datei `system.cfg` mit einem Texteditor öffnen
  - Wifi Settings anpassen:

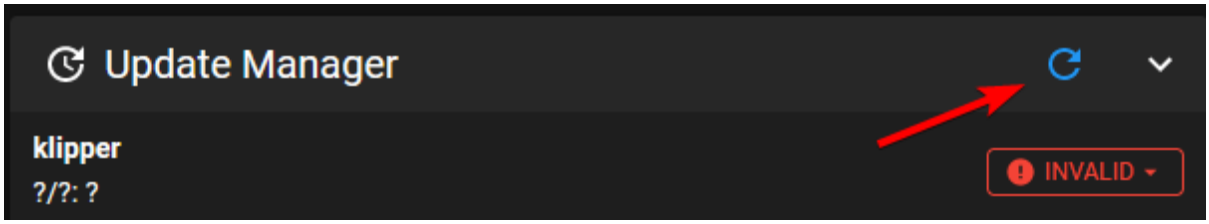
```
# wifi name
WIFI_SSID="WLAN_SSID"
# wifi password
WIFI_PASSWD="WLAN_PASSWORT"
```

- Hostname ggf. anpassen
  Wird auch in der Datei `system.cfg` eingetragen:
  `hostname=“BTT-PI12”`

- Karte in den BTT Pi einsetzen
- Kühlkörper und WLAN Antenne nicht vergessen anzubringen
- BTT Pi starten ...

> **Hinweis**
>
> 
>
> Nicht wundern beim Login ... Da steht CB1 und nicht PI1.2. Das Image ist für beide System identisch!

Updates

- per SSH einloggen
- Updates und ein paar Tools installieren
  `sudo apt update && sudo apt upgrade -y && sudo apt install -y git silversearcher-ag wavemon hexedit sudoku tcpdump iptraf mc htop dcfldd nano usbutils ranger tldr ncdu can-utils multitail fd-find && mkdir -p ~/.local/share && tldr -u`
- überflüssige Dienste entfernen
  `sudo apt autoremove alsa* modem* cups* pulse* avahi*`
- IP ermitteln
  `ip a`
- Mainsail Weboberfläche öffnen über die IP
  - Unter **Mashine** einmal die Repos neu laden um die roten “Invalid” Meldungen zu entfernen
    

  - Jetzt alle Komponenten aktuslisieren lassen mit **UPDATE ALL COMPONENTS**

Linux MCU einrichten

Wer den BTT Pi mit einem ADXL345 Sensor für Input Shaper ausstatten möchte, sollte den Linux Prozess noch einrichten. Der Sorgt dafür, das die GPIO Pins vom BTT Pi aus Klipper Sicht genutzt werden können und ermöglicht eben den Betrieb von extra Sensoren wie dem ADXL345.

- auf dem BTT Pi einloggen mittels SSH
- `cd ~/klipper/`
- `sudo cp ./scripts/klipper-mcu.service /etc/systemd/system/`

- `sudo systemctl enable klipper-mcu.service`
- `make menuconfig`
  - hier wählt man dann `Microcontroller Architecture` `Linux process` aus

```
[*] Enable extra low-level configuration options
    Micro-controller Architecture (Linux process)  --->
()  GPIO pins to set at micro-controller startup
```

  - mit Q beenden und mit Y speichern
- `sudo service klipper stop`
- `make flash -j4`
- `sudo service klipper start`

Minimale printer.cfg

Um Klipper vorübergehend in Betrieb zu nehmen, kann man folgende Mini Konfiguration verwenden:

printer.cfg

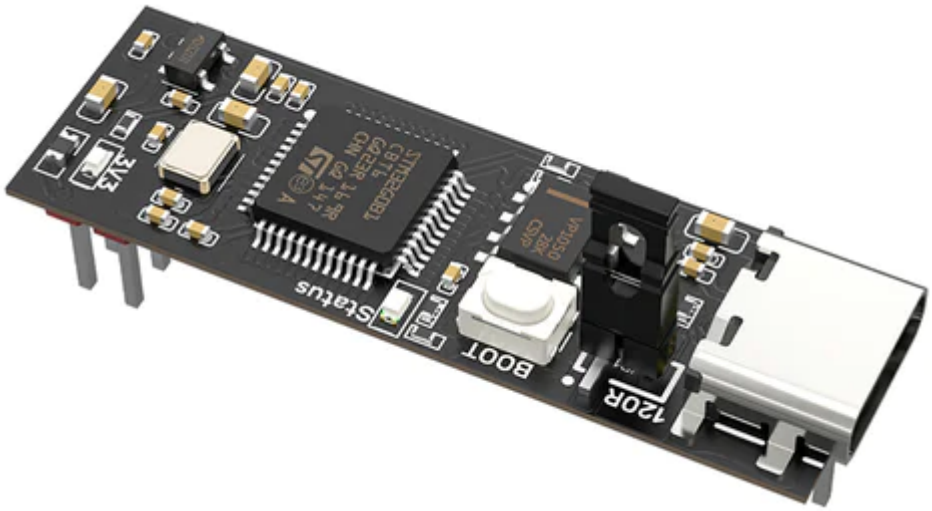
```
[include mainsail.cfg]
[mcu]
serial : /tmp/klipper_host_mcu

#[mcu Board]
#serial : /dev/serial/by-id/usb-
Klipper_stm32f407xx_2B00350011473934373033337-if00

[printer]
kinematics: none
max_velocity: 1000
max_accel: 1000
```


Mit dieser Konfig kann der Drucker natürlich rein gar nichts. Aber man wird erstmal alle Fehlermeldungen in MainSail los.

CAN einrichten



Controller : STM32G0B1
Die folgenden Schritte sind nur notwendig, wenn man das CAN Modul besitzt und einsetzen möchte.

Adapter flashen

Bei mir war die aufgespielte Firmware komplett nutzlos. Weder das Gerät ließ sich in Linux finden, noch konnte ich den CAN Bus aktivieren. Das Problem haben wohl auch andere → https://github.com/bigtreetech/BTT-Pi/issues/11 Ich würde also als erstes die Firmware aktualisieren

- benötigte Pakete installieren
  `sudo apt install cmake gcc-arm-none-eabi -y`
- `cd ~`
- Repository von Github klonen
  `git clone --depth=1 -b stm32g0_support https://github.com/bigtreetech/candleLight_fw`
- `cd ~/candleLight_fw`
- Vorbereitungen für das Kompilieren (Create Toolchain)
  ```
  mkdir build
  cd build
  cmake .. -DCMAKE_TOOLCHAIN_FILE=../cmake/gcc-arm-none-eabi-8-2019-q3-update.cmake
  ```
- Kompilieren
  `make budgetcan_fw`
- Jetzt den CAN Adapter über ein USB-C Kabel an den BTT Pi anschließen und dabei den BOOT Taster auf dem Board gedrückt halten
  - Wer mag kann in der Konsole `sudo dmesg -Wtd` eingeben und sollte in etwa so eine Ausgabe bekommen:

```
[<    0.000000>] usb 6-1: new full-speed USB device number 2 using ohci-platform
[<    0.235034>] usb 6-1: New USB device found, idVendor=0483, idProduct=df11, bcdDevice= 2.00
[<    0.000047>] usb 6-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[<    0.000021>] usb 6-1: Product: DFU in FS Mode
[<    0.000015>] usb 6-1: Manufacturer: STMicroelectronics
```

```
[<    0.000015>] usb 6-1: SerialNumber: 2056384F464D
```

  - Alternativ geht auch `dfu-util -l` . Wenn das `Found DFU: [0483:df11] …` meldet passt auch alles.
  - Wenn nichts davon eintritt nochmal abstecken und Vorgang wiederholen 🙂
- Den CAN Adapter flashen
`dfu-util -R -a 0 -s 0x8000000:mass-erase:force -D ~/candleLight_fw/build/budgetcan_fw.bin`

```
biqu@BTT-PI12:~/candleLight_fw/build$ dfu-util -R -a 0 -s 0x08000000:mass-erase:force -D ~/candleLight_fw/build/budgetcan_fw.bin
dfu-util 0.9

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2016 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to http://sourceforge.net/p/dfu-util/tickets/

dfu-util: Invalid DFU suffix signature
dfu-util: A valid DFU suffix will be required in a future dfu-util release!!!
Opening DFU capable USB device...
ID 0483:df11
Run-time device DFU version 011a
Claiming USB DFU Interface...
Setting Alternate Setting #0 ...
Determining device status: state = dfuIDLE, status = 0
dfuIDLE, continuing
DFU mode device DFU version 011a
Device returned transfer size 1024
DfuSe interface name: "Internal Flash  "
Performing mass erase, this can take a moment
Downloading to address = 0x08000000, size = 19480
Download        [=========================] 100%        19480 bytes
Download done.
File downloaded successfully
Resetting USB to switch back to runtime mode
```

Adapter einsetzen

- BTT Pi stromlos machen
- CAN Adapter einsetzen
- BTT Pi wieder starten

CAN Bus aktivieren

Damit der Adapter im Betriebsystem auch erkannt wird muss eine Netzwerk Interface Konfiguration (`/etc/network/interfaces.d/can0`) angelegt werden. Es ist möglich das die Datei schon im BTT Image eingepflegt ist …

- `sudo nano /etc/network/interfaces.d/can0`
- folgenden Inhalt in der Datei einfügen :

```
auto can0
iface can0 can static
    bitrate 1000000
    up ifconfig $IFACE txqueuelen 1024
```

- Editor mit **STRG + x → dann Y → dann Enter** verlasen
- System neu starten mittels
`sudo reboot`
- Nach dem Reboot sollte der Befehl `ip a` ein CAN Interface listen :

```
can0: <NOARP,UP,LOWER_UP,ECHO> mtu 16 qdisc pfifo_fast state UP group
default qlen 1024
```

```
    link/can
```

- Wichtig dabei ist **state UP**. Wird hier ein DOWN angegeben, ist das Interface nicht betriebsbereit!

Can Bus Test

- `sudo systemctl stop klipper.service`
- `~/klippy-env/bin/python ~/klipper/scripts/canbus_query.py can0`
- `sudo systemctl start klipper.service`

HDMI Display

Am HDMI Anschluss sollte nach dem Start sofort eine Ausgabe erfolgen. TBD: Prüfen was wenn mit SPI Monitor! Anpassungen kann man an zwei Stellen vornehmen:

- `sudo nano /boot/BoardEnv.txt`

```
## Specify HDMI output resolution (eg. extraargs=video=HDMI-
A-1:800x480-24@60)
#extraargs=video=HDMI-A-1:1024x600-24@60
```

Hier in der zweiten Zeile die # entfernen und die Auflösung anpassen

- `sudo nano /boot/system.cfg`

```
##############################################
# HDMI klipperScreen rotation
# ks_angle: Rotation angle
#       normal: 0;          inverted: 180;
#       left: 90 to left;   right: 90 to right;
#ks_angle="normal"
```

Hiermit kann die grafische Ausgabe auf dem HDMI Port gedreht werden.
 Die Textkonsole bleibt davon leider unberührt 

3,5" SPI Display

- Die Auflösung beträgt 460×320 Pixel
- Das TFT ist ziemlich stromhungerig und wird merklich warm
- Aktivieren über `sudo nano /boot/BoardEnv.txt`
  - bei den Overlays `tft35_spi` hinzufügen → Bsp: `overlays=tft35_spi ir`
  -  **ACHTUNG** 
    Es darf in der BoardEnv.txt Datei nur eine (!!) Zeile mit overlays geben. Also alle Overlays

in einer Zeile zusammenfassen. Bei mehreren Overlays Zeilen zählt nur der letzte Eintrag in der Datei !!

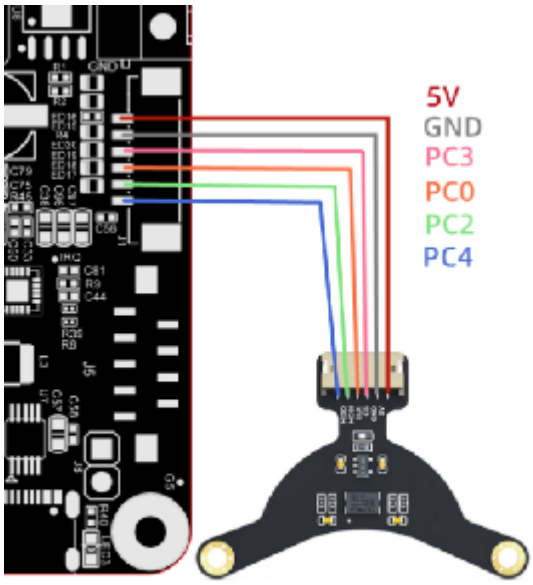
- Wenn das TFT aktiviert wird hat es Vorrang vor dem HDMI Ausgang
- Der Bildschirm kann auch bei Bedarf kalibriert werden https://bigtreetech.github.io/docs/libinput_calibration.html

Links

- https://github.com/bigtreetech/TFT35-SPI
- https://bigtreetech.github.io/docs/TFT35%20SPI.html
- https://bigtreetech.github.io/docs/libinput_calibration.html

ADXL345

BTT Modul



ADXL345 pin	Function	Pi 1.2 Pin	Pi 1.2 GPIO
3V3 (or VCC)	Power	3,3V	3.3V DC power
GND	GND	GND	GND
CS	Chip Select	PC3	(3-1)*32 + 3 = gpio67
SDO	MISO	PC4	(3-1)*32 + 4 = gpio68
SDA	MOSI	PC2	(3-1)*32 + 2 = gpio66
SCL	Clock	PC0	(3-1)*32 + 0 = gpio64

Das Modul verwendet `spidev0_0` als SPI Bus. Für die Konfig ist folgendes notwendig:

- Aktivieren über `sudo nano /boot/BoardEnv.txt`
  - bei den Overlays `spidev0_0` hinzufügen → Bsp: `overlays=spidev0_0 ir`

  - **ACHTUNG**
    Es darf in der BoardEnv.txt Datei nur eine (!!) Zeile mit overlays geben. Also alle Overlays in einer Zeile zusammenfassen. Bei mehreren Overlays Zeilen zählt nur der letzte Eintrag in der Datei !!
- Die Linux MCU muss laufen!
- Klipper Konfig erweitern

```
[adxl345]
cs_pin: BTTPI:gpio67
spi_bus: spidev0.0

[resonance_tester]
accel_chip: adxl345
probe_points:
    60, 60, 20  # an example
```

DIY

Pi 1.2 Pin	Function	Pi 1.2 GPIO	Wird genutzt von …
PC3	CS spidev0_0	(3-1)*32 + 3 = gpio67	spidev0_0 ADXL345
PC4	SPI0 MISO	(3-1)*32 + 4 = gpio68	
PC2	SPI0 MOSI	(3-1)*32 + 2 = gpio66	
PC0	SPI0 Clock	(3-1)*32 + 0 = gpio64	
PH6	SPI1 CLK	GPIO230	
PH7	SPI1 MOSI	GPIO231	
PH8	SPI1 MISO	GPIO232	
PC11	CS of spidev1_0	GPIO75	spidev1.0 MCP2515
PC7	CS of spidev1_1	GPIO71	spidev1.1 TFT
PC13	CS of spidev1_2	GPIO77	spidev1.2 frei

> **Hinweis 1**
> Die Default CS Pins können nicht in Klipper genutzt werden. Da kommt ein Fehler “Unable to open out GPIO chip line”
> **Hinweis 2**
> Für Klipper einen freien GPIO verwenden und den als CS Pin nutzen (Beispiel gpio74)

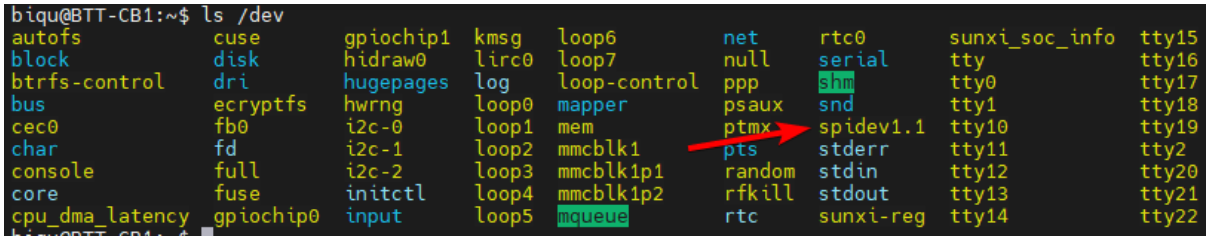
- Aktivieren über `sudo nano /boot/BoardEnv.txt`
  - bei den Overlays `spidev1_1` hinzufügen → Bsp: `overlays=spidev1_1 ir`
  - **ACHTUNG**
    Es darf in der BoardEnv.txt Datei nur eine (!!) Zeile mit overlays geben. Also alle Overlays in einer Zeile zusammenfassen. Bei mehreren Overlays Zeilen zählt nur der letzte Eintrag in der Datei !!
- Die Linux MCU muss laufen!
- Klipper Konfig erweitern

```
[adxl345]
cs_pin: BTTPI:gpio74
spi_bus: spidev1.1
```

```
[resonance_tester]
accel_chip: adxl345
probe_points:
    60, 60, 20  # an example
```

- Hier gibt es eine Diskussion zu dem Thema https://github.com/bigtreetech/CB1/discussions/47

Tests

- Wenn ein SPI Bus über das Overlay aktiviert wurde sollte der im Device Verzeichnis `/dev` auftauchen ´nach einem Reboot
  - 

  - Hier wurde folgendes in der `/boot/BoardEnv.txt` eingetragen : `overlays=spidev1_1 ir`
- Wenn der Bus in `/dev` vorhanden ist kann man in der MainSail Konsole den Input Shaper Sensor abfragen mit ACCELEROMETER_QUERY
  - Bei einer erfolgreichen Abfrage liefert der Sensor Werte

```
11:23  accelerometer values (x, y, z): 0.000000, -1406.391290, -7890.783629

11:23  ACCELEROMETER_QUERY
```

  - Wenn der Sensor nicht antwortet kommt eine Fehlermeldung

```
11:24  Invalid adxl345 id (got 0 vs e5).

11:24  Invalid adxl345 id (got 0 vs e5).
       This is generally indicative of connection problems
       (e.g. faulty wiring) or a faulty adxl345 chip.
```

    In dem Fall Konfig und Verkabelung prüfen!

GPIO nutzen (Linux MCU)

Die GPIO Pins vom BTT Pi können von Klipper angesteuert werden. Man muss nur mit der Pin Bezeichnung aufpassen ...

Pin Berechnung

Um die GPIO Pins in Klipper nutzen zu können muss ggf. umgerechnet werden. Die Pins sind im Normalfall nach dem Schema **PxNN** benannt. **x** kann dabei von A..G gehen und **NN** ist eine Zahl. Das nutzt aber in Klipper nichts, weil dort die richtigen GPIO Nummern angegeben werden müssen. Dafür gibt es folgende Rechnung:

- Der Buchstabe wird in eine Zahl gewandelt. A = 0, B = 1 ... G = 6
- Die Zahl * 32
- Anschließend noch die NN Nummer addieren
- Beispiel PC15
  - C = 2, NN = 15
  - 2*32 + 15 = 79
  - PC15 → gpio79

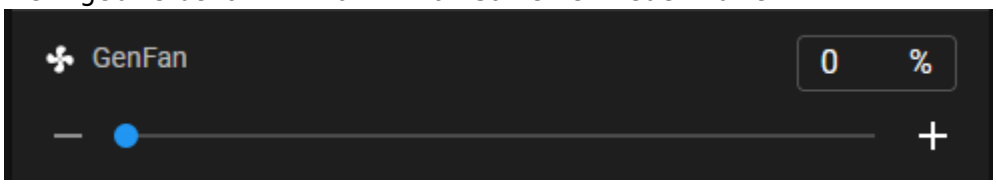
In der Klipper Konfig trägt man an der Stelle für den Pin dann gpio79 und **nicht** PC15 ein.

Beispiel Lüfter

Auf dem BTT Pi 1.2 Board ist ein extra Lüfter Anschluss mit Mosfet. Um diesen Lüfter zu nutzen kann man folgende Konfig verwenden:

```
[fan_generic GenFan]
pin             : BTTPI:gpio211   # Hier ggf. den Host anpassen -> Name der
Linux MCU
max_power       : 1.0
#shutdown_speed:
cycle_time      : 0.010
hardware_pwm    : false
kick_start_time: 1.100
```

Als Ergebnis bekommt man in MainSail einen neuen Lüfter



Beispiel OutPin

Einen einfachen Ausgang zum Schalten von was auch immer kann man so realisieren:

```
[output_pin OutPin]
pin   : gpio79
pwm   : false
value : 0
```

> **ACHTUNG**
> Hier darf nicht einfach irgendeine Last angeschlossen werden. Die GPIO Pins können nämlich kaum Strom abgeben. Als im Zweifel mit einem Mosfet arbeiten oder jemanden fragen der sich damit auskennt 😉

Ergebnis in MainSail

Links

- https://github.com/bigtreetech/CB1#40-pin-gpio
- https://github.com/So6Rallye/BTT-Pi/blob/master/BIGTREETECH%20Pi%20V1.2%20-%20Board%20Fan%20Pin%20Configuration

WS2812 Licht

III. How can I enable ws2812 RGB lightning and which Pins are used for that?

Only PC14 can be used for ws2812 and if you can modify the dts, you can modify the dts to any io

Please check the attachment 4 for your reference.

IR Sensor nutzen

Ab Kernel 4.x gibt es einen IR Treiber im Kernel. Sowas wie Lirc ist also nicht mehr wirklich vonnöten.

- Erkennen, ob das IR Modul geladen wurde
  `dmesg |grep IR`
```
[    8.113451] rc rc0: sunxi-ir as /devices/platform/soc/7040000.ir/rc/rc0
[    8.115687] rc rc0: lirc_dev: driver sunxi-ir registered at minor = 0, raw IR receiver, no transmitter
[    8.121535] input: sunxi-ir as /devices/platform/soc/7040000.ir/rc/rc0/input4
[    8.125278] sunxi-ir 7040000.ir: initialized sunXi IR driver
```
- `cat /proc/bus/input/devices` muss auch ein Device melden
```
I: Bus=0019 Vendor=0001 Product=0001 Version=0100
N: Name="sunxi-ir"
P: Phys=sunxi-ir/input0
S: Sysfs=/devices/platform/soc/7040000.ir/rc/rc0/input4
U: Uniq=
H: Handlers=kbd event2
B: PROP=20
B: EV=100017
B: KEY=1000000 0 0
B: REL=3
B: MSC=10
```
- Anzeigen welche Protokolle genutzt werden
  `cat /sys/class/rc/rc0/protocols`
```
biqu@BTT-CB1:~$ cat /sys/class/rc/rc0/protocols
rc-5 nec rc-6 jvc sony rc-5-sz sanyo sharp mce_kbd xmp imon rc-mm [lirc]
```
  HIer ist nur das lirc Protokoll aktiv
- Die meisten einfachen Fernbedienungen nutzen das NEC Protokoll. das kann man ändern indem man sich als root anmeldet `sudo su -` und dann das Protokoll hinzufügt `echo nec > /sys/class/rc/rc0/protocols`. Danach sollte es auch mit eckigen Klammern versehen sein
```
root@BTT-CB1:~# cat /sys/class/rc/rc0/protocols
rc-5 [nec] rc-6 jvc sony rc-5-sz sanyo sharp mce_kbd xmp imon rc-mm [lirc]
```
  - Das überlegt aber keinen Reboot. Hier ggf. mit cron einen reboot job einfügen ...

```
sudo crontab -e
@reboot echo nec > /sys/class/rc/rc0/protocols
```

- Testen kann man das nun unterschiedlich …
  - `cat /dev/input/event0`

    cat auf das event File geht, ist aber unschön 
  - `evtest` → sunxi-ir wählen

```
Event: time 1694613088.515563, type 4 (EV_MSC), code 4 (MSC_SCAN), value 40
Event: time 1694613088.515563, -------------- SYN_REPORT ------------
Event: time 1694613088.567283, type 4 (EV_MSC), code 4 (MSC_SCAN), value 40
Event: time 1694613088.567283, -------------- SYN_REPORT ------------
Event: time 1694613089.050239, type 4 (EV_MSC), code 4 (MSC_SCAN), value 15
Event: time 1694613089.050239, -------------- SYN_REPORT ------------
```

  - `sudo apt install ir-keytable`
    `ir-keytable -t`

```
biqu@BTT-CB1:~$ ir-keytable -t
Testing events. Please, press CTRL-C to abort.
203.473153: lirc protocol(nec): scancode = 0x45
203.473177: event type EV_MSC(0x04): scancode = 0x45
203.473177: event type EV_SYN(0x00).
203.524905: lirc protocol(nec): scancode = 0x45 repeat
203.524926: event type EV_MSC(0x04): scancode = 0x45
203.524926: event type EV_SYN(0x00).
204.318951: lirc protocol(nec): scancode = 0x46
204.318975: event type EV_MSC(0x04): scancode = 0x46
```

  - Python (https://unix.stackexchange.com/questions/753554/how-to-shutdown-linux-with-an-infrared-remote-control)
    `sudo pip3 install evdev –target /usr/lib/python3/dist-packages`
    `sudo apt install thonny`

```
from evdev import InputDevice
dev = InputDevice('/dev/input/event0')

for event in dev.read_loop():
    print(event)
```

Links

- https://linux-sunxi.org/IR
- https://forum.armbian.com/topic/17007-enabling-infrared-ir-receiver-on-nanopi-neo/
- https://unix.stackexchange.com/questions/753554/how-to-shutdown-linux-with-an-infrared-remote-control

USB Boot

Root FS umhängen
setenv bootargs “root=${rootdev} rootwait rootfstype=${rootfstype} ${consoleargs} consoleblank=0 loglevel=7 ubootpart=${partuuid} usb-storage.quirks=${usbstoragequirks} ${extraargs} ${extraboardargs}”

boot.cmd anpassen …

https://linux-sunxi.org/FEL/USBBoot

Configs

/boot/BoardEnv.txt

BoardEnv.txt

```
bootlogo=true
overlay_prefix=sun50i-h616

## 'sun50i-h616-biqu-sd' for CB1, 'sun50i-h616-biqu-emmc' for CB1 eMMC
version
fdtfile=sun50i-h616-biqu-sd

## default 'display' for debug, 'serial' for /dev/ttyS0
console=display

## Specify HDMI output resolution (eg. extraargs=video=HDMI-
A-1:800x480-24@60)
#extraargs=video=HDMI-A-1:1024x600-24@60

## uncomment for ws2812
#overlays=ws2812

## uncomment for i2c-gpio, pwm3, disable uart0 for pwm3
#overlays=light

overlays=spi-spidev
param_spidev_spi_bus=1
param_spidev_spi_cs=1
param_spidev_max_freq=1000000

## uncomment for TFT35_SPI screen
#overlays=tft35_spi

## uncomment MCP2515 spi to canbus module
#overlays=mcp2515

## Chipselect Lines !
## uncomment overlays=spidev1_2 to release 'spidev1.2' to user space
(For example: adxl345),
## spidev1.0 is used by MCP2515,
## spidev1.1 is used by tft35_spi.
## uncomment to release 'spidev0.0' to user space
#overlays=spidev0_0

## uncomment to release 'spidev1.0' to user space
```

```
#overlays=spidev1_0

## uncomment to release 'spidev1.1' to user space
#overlays=spidev1_1

## ADXL345
## uncomment to release 'spidev1.2' to user space
#overlays=spidev1_2

## uncomment to set 'PH10' for IR
#overlays=ir

## write the config after the 'overlays' and separate it with a space
when multiple functions are enabled
#overlays=disable_uart0 pwm ws2812 light tft35_spi mcp2515 spidev0_0
spidev1_0 spidev1_1 spidev1_2 ir

#-----------------------------------------------#
rootdev=UUID=e345f9fc-00c8-4132-8e8e-8afa1304e8ed
rootfstype=ext4
```

printer.cfg

printer.cfg

```
[include mainsail.cfg]
[virtual_sdcard]
path           : /home/biqu/printer_data/gcodes
on_error_gcode : CANCEL_PRINT

[mcu]
serial         : /tmp/klipper_host_mcu

[printer]
kinematics: none
max_velocity   : 1000
max_accel      : 1000

# PIN CALC
# PG19 is definitely a GPIO pin. It may not be on the 40 pin GPIO
header but any input / output pin on the BTT Pi and CB1 is GPIO.
#The formula to convert from PG19 to GPIO is:
#[(PG - PA)*32] + {Pin Number}
#Where PA = 1, PB =2….so PG = 7
#So [(7-1)*32] + 19 = 211
# Test PC15
# (3-1)*32 + 15 = 79 -> GPIO79
```

```
[fan_generic GenFan]
#host:
pin            : gpio211
max_power      : 1.0
#shutdown_speed:
cycle_time     : 0.010
hardware_pwm   : false
kick_start_time: 1.100

[output_pin OutPin]
pin: gpio79
pwm: false
#   Set if the output pin should be capable of pulse-width-modulation.
#   If this is true, the value fields should be between 0 and 1; if it
#   is false the value fields should be either 0 or 1. The default is
#   False.
value: 0
#   The value to initially set the pin to during MCU configuration.
#   The default is 0 (for low voltage).
##cycle_time: 0.0100
#   The amount of time (in seconds) per PWM cycle. It is recommended
#   this be 10 milliseconds or greater when using software based PWM.
#   The default is 0.100 seconds for pwm pins.
##hardware_pwm: False

[adxl345]
cs_pin  : gpio74
spi_bus : spidev1.1

[resonance_tester]
accel_chip: adxl345
probe_points:
    150,150,20  # Bettmitte
```