GPIO

Grundlagen

https://wiki.openwrt.org/doc/hardware/port.gpio

Pin-Belegung

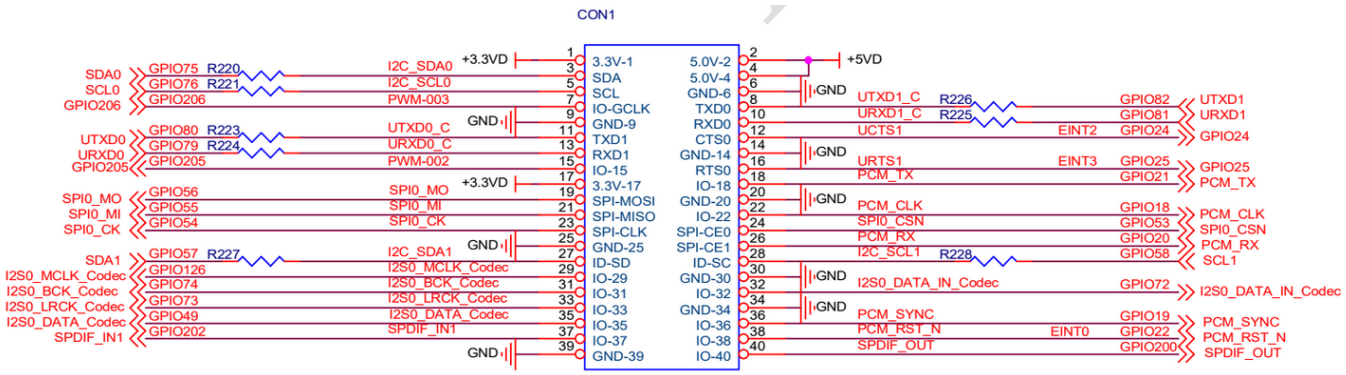
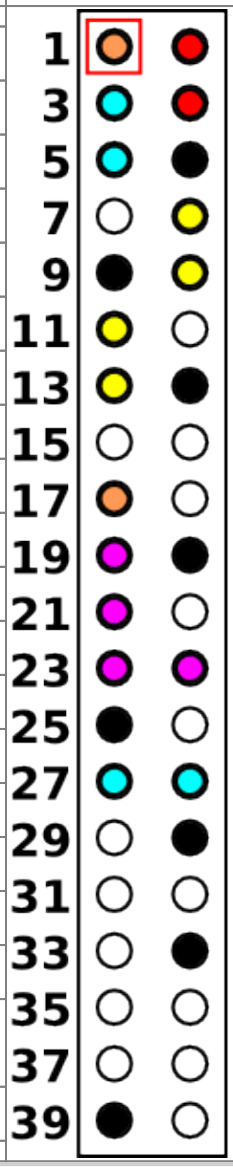


Bild aus den BPI-R2 Schematics

Nebenfunktion	Hauptfunktion	pin#	xxxxxxxxxxxxxxxxxxxx	pin#	Hauptfunktion	Nebenfunktion
-	3V3	1		2	5V	-
I2C_SDA0	GPIO 75	3		4	5V	-
I2C_SCL0	GPIO 76	5		6	GND	-
PWM3	GPIO 206	7		8	GPIO 82	UART1 TX
-	GND	9		10	GPIO 81	UART1 RX
UART0 TX	GPIO 80	11		12	UCTS1 / INT2	GPIO 24 (*)
UART0 RX	GPIO 79	13		14	GND	-
PWM2	GPIO 205	15		16	GPIO 25 / INT3	URTS1
-	3V3	17		18	GPIO 21	PCM_TX
SPI0_MO	GPIO 56	19		20	GND	-
SPI0_MI	GPIO 55	21		22	GPIO 18	PCM_CLK
SPI0_CK	GPIO 54	23		24	GPIO 53	SPI0_CSN
-	GND	25		26	GPIO 20	PCM_RX
I2C_SDA1	GPIO 57	27		28	GPIO 58	I2C_SCL1
	GPIO 126	29		30	GND	-
I2S0_BCK	GPIO 74	31		32	GPIO 72	I2S0_DATA_IN
I2S0_LRCK	GPIO 73 (?)	33		34	GND	-
I2S0_DATA	GPIO 49 (M)	35		36	GPIO 19	PCM_SYNC
SPDIF_IN1	GPIO 202	37		38	INT0	GPIO 22 (*) / PCM_RST_IN
-	GND	39		40	GPIO 200	SPDIF_OUT



	main	spare
Pin 1	3V3	-
Pin 2	5V	-
Pin 3	GPIO 75	I2C_SDA0
Pin 4	5V	-
Pin 5	GPIO 76	I2C SCL0
Pin 6	GND	-
Pin 7	GPIO 206	PWM3
Pin 8	GPIO 82	UART1 TX
Pin 9	GND	-
Pin 10	GPIO 81	UART1 RX
Pin 11	GPIO 80	UART0 TX
Pin 12	Int2	GPIO 24 (*) / UCTS1
Pin 13	GPIO 79	UART0 RX
Pin 14	GND	-
Pin 15	GPIO 205	PWM2
Pin 16	GPIO 25 / Int3	URTS1
Pin 17	3V3	-

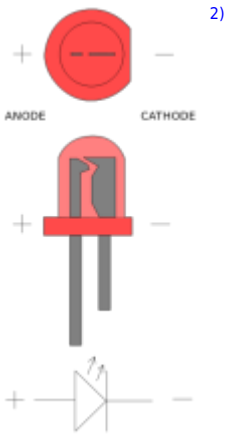
	main	spare
Pin 18	GPIO 21	PCM_TX
Pin 19	GPIO 56	SPI0_MO
Pin 20	GND	-
Pin 21	GPIO 55	SPI0_MI
Pin 22	GPIO 18	PCM_CLK
Pin 23	GPIO 54	SPI0_CK
Pin 24	GPIO 53	SPI0_CSN
Pin 25	GND	-
Pin 26	GPIO 20	PCM_RX
Pin 27	GPIO 57	I2C_SDA1
Pin 28	GPIO 58	I2C_SCL1
Pin 29	GPIO 126	
Pin 30	GND	-
Pin 31	GPIO 74	I2S0_BCK
Pin 32	GPIO 72	I2S0_DATA_IN
Pin 33	GPIO 73 (?)	I2S0_LRCK
Pin 34	GND	-
Pin 35	GPIO 49 (M)	I2S0_DATA
Pin 36	GPIO 19	PCM_SYNC
Pin 37	GPIO 202	SPDIF_IN1
Pin 38	INT0	GPIO 22 (*) / PCM_RST_IN
Pin 39	GND	-
Pin 40	GPIO 200	SPDIF_OUT

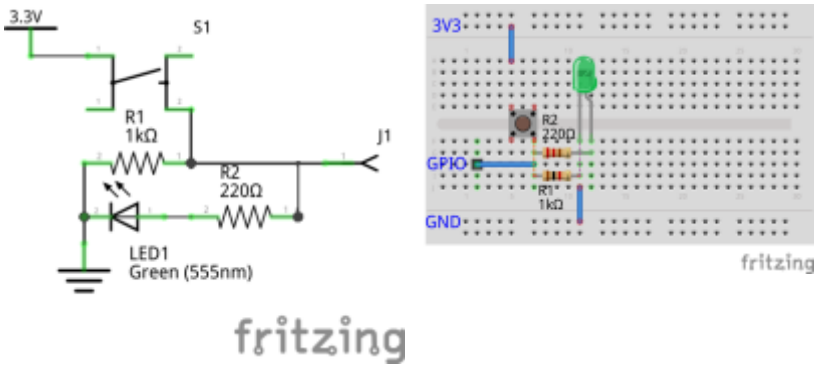
```
(*) Spezial-GPIO benötigen Speicher-Patch und Mode-Einstellung (4.4.70)
(?) GPIO getested, noch nicht funktionsfähig
(M) Mode=0 erforderlich
```

Schaltungen

1)

2)





Fritzing-Datei

https://docs.labs.mediatek.com/resource/linkit7697-arduino/en/tutorial/smd-buttons

Kernel 4.4.70

Standard GPIO

```
root@bpi-iot-ros-ai:~# GPIO=/sys/devices/platform/1000b000.pinctrl/mt_gpio
root@bpi-r2:~# echo "mode 25 0" >$GPIO #nicht immer nötig
root@bpi-r2:~# echo "dir 25 1" >$GPIO
root@bpi-r2:~# echo "out 25 1" >$GPIO
```

funktioniert mit LED an Pin 14 (-) und Pin 16 (+), inkl. Vorwiderstand (220 Ohm)

Spezial GPIO

für die GPIOs 22/(23??)/24 ist es nötig, vorher ein Register zu setzen (siehe issue#17 Kommentar #15)

mwrite

```
root@bpi-iot-ros-ai:~# ./mwrite /dev/mem 0x10005b10 0x00000038
./mwrite offset : 10005b10, val : 00000038
b6f03b10
root@bpi-r2:~# GPIO=/sys/devices/platform/1000b000.pinctrl/mt_gpio
root@bpi-r2:~# echo "mode 24 0" >$GPIO
root@bpi-r2:~# echo "dir 24 1" >$GPIO
root@bpi-r2:~# echo "out 24 1" >$GPIO
```

zusätzlich musste ich beim GPIO24 (pin 12) den mode noch auf 0 setzen

Kernel 4.14

GPIO_SYSFS und CONFIG_DEBUG_GPIO müssen in Kernel-Config gesetzt sein (.config)

Standard GPIO

```
root@bpi-r2# mount -t debugfs none /sys/kernel/debug
root@bpi-r2# cat /sys/kernel/debug/pinctrl/1000b000.pinctrl/gpio-ranges
GPIO ranges handled:
0: 1000b000.pinctrl GPIOS [232 - 511] PINS [0 - 279]
root@bpi-r2# GPIO_NO=$((232+25))
root@bpi-r2# echo $GPIO_NO
257
root@bpi-r2# echo $GPIO_NO > /sys/class/gpio/export
```

Pin 14=GND/16=GPIO25

GPIO als Ausgang

```
root@bpi-r2# echo out > /sys/class/gpio/gpio${GPIO_NO}/direction
root@bpi-r2# echo 1 > /sys/class/gpio/gpio${GPIO_NO}/value
root@bpi-r2# echo 0 > /sys/class/gpio/gpio${GPIO_NO}/value
```

schaltet LED (inkl. Vorwiderstand) an Pin 14=GND/16=GPIO25 ein (1) und wieder aus (0)

GPIO als Eingang

high-active Taster-Schaltung an GPIO 200 (Pin 40 zwischen Schalter und Widerstand, Pin 39 als GND [Widerstand] und Pin 17 als 3v3-vcc)

```
[10:54] root@bpi-r2:~# echo in > /sys/class/gpio/gpio${GPIO_NO}/direction
[10:56] root@bpi-r2:~# cat /sys/class/gpio/gpio${GPIO_NO}/value
0 #Taster nicht gedrückt
[10:56] root@bpi-r2:~# cat /sys/class/gpio/gpio${GPIO_NO}/value
1 #Taster gedrückt
[10:56] root@bpi-r2:~# cat /sys/class/gpio/gpio${GPIO_NO}/value
0 #Taster nicht gedrückt

#dauerhaft alle 1/4s abfragen
watch -n 0.25 cat /sys/class/gpio/gpio${GPIO_NO}/value
```

Special GPIO

Speicher-hack (wie in 4.4.70) nicht notwendig

Beispiel für GPIO24 (pin12):

```
root@bpi-r2# GPIO_NO=$((232+24))
root@bpi-r2# echo $GPIO_NO > /sys/class/gpio/export
root@bpi-r2# echo out > /sys/class/gpio/gpio${GPIO_NO}/direction
```

```
root@bpi-r2# echo 1 > /sys/class/gpio/gpio${GPIO_NO}/value
```

LED geht an :)

on-board LEDs

Die On-Board-LEDs welche hier angesteuert werden befinden sich nahe der Netzteil-Buchse (nicht neben der GPIO-Leiste)

http://forum.banana-pi.org/t/control-on-board-leds/4287/13

an ⇒

```
echo 1 > /sys/class/leds/bpi-r2:isink:green/brightness
```

aus ⇒

```
echo 0 > /sys/class/leds/bpi-r2:isink:green/brightness
```

blinken (erstellt delay_on/off-knoten zur Frequenz-Kontrolle) ⇒

```
echo timer > /sys/class/leds/bpi-r2:isink:green/trigger
```

ändern der Blink Frequenz (an/aus-Zeit in ms) ⇒

```
echo 100 > /sys/class/leds/bpi-r2:isink:green/delay_on
echo 100 > /sys/class/leds/bpi-r2:isink:green/delay_off
```

in meinen Tests, grün blinkt beim anschalten (rot+blau gehen einfach an/aus), bisher weis ich noch nicht, wie man das Blinken der grünen LED deaktivieren kann

```
L=/sys/class/leds/bpi-r2\:isink
echo 0 > $L:red/brightness          #rot aus
echo 1 > $L:red/brightness          #rot an
echo 0 > $L:green/brightness        #grün aus
echo 1 > $L:green/brightness        #grün blinkt
```

```
[16:08] root@bpi-r2:~# L=/sys/class/leds/bpi-r2\:isink
[17:41] root@bpi-r2:~# L2=/sys/class/leds/bpi-r2\:pio
[17:42] root@bpi-r2:~# echo 1 > $L2:green/brightness
[17:42] root@bpi-r2:~# echo 1 > $L2:blue/brightness
[17:42] root@bpi-r2:~# echo 0 > $L2:green/brightness
[17:42] root@bpi-r2:~# echo 0 > $L2:blue/brightness
```

UART

DTS(i) anpassen

unter Kernel 4.4.x fehlen die DeviceTree-Abschnitte, diese kann man man aber einfach aus einem höheren Kernel nachtragen (dtsi). in der *bpi*.dts bzw. *bananapi*.dts dann auf enabled setzen

definition in der mt7623.dtsi:
http://elixir.free-electrons.com/linux/v4.13-rc7/source/arch/arm/boot/dts/mt7623.dtsi

nun in der bananapi.dts den uart noch auf „status=okay“ setzen

bei uart muss darauf geachtet werden, dass in der mt7623.dtsi erst uart2 und dann die anderen kommen, da sonst nach der uboot-Meldung „Starting Kernel“ keine Ausgabe mehr auf dem Terminal kommt

Uart3 kann auf UCTS2/URTS2 gerouted werden. Diese befinden sich neben dem Anschluss für Debug-UART (hier)

Einstellungen des Ports

```
#Einstellungen des seriellen Ports anzeigen (ersetze ttyS2 mit ttyUSB0 wenn ein USB2serial-Adapter verwendet wird):
stty -F /dev/ttyS2 -a

#Das setzt die Baudrate auf 9600, 8 bits, 1 stop bit, keine parität:
stty -F /dev/ttyS2 9600 cs8 -cstopb -parenb

#verarbeitung deaktivieren (Zeichenkonvertierung,Zeilenumbrüche,...)
stty -F /dev/ttyS2 -opost

#raw Modus
stty -F /dev/ttyS2 raw
```

Nutzung

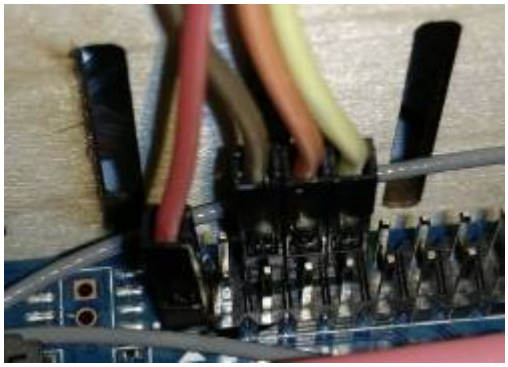
```
pin 8/10 = uart1 (tx/rx) = 11003000
pin 11/13 = uart0 (tx/rx) = 11002000
```

```
#!/bin/bash
DEV=/dev/ttyS2
#stty -F ${DEV} sane
#stty -F ${DEV} 9600 cs8 -cstopb -parenb -crtscts -echo
stty -F ${DEV} 9600 cs8 -cstopb -parenb raw -echo

dmesg | grep "ttyS.*MMIO" | sed 's/^\[.*\] \(\d*.*\) at.*$/\1/'

echo "11002000 = uart0 (tx/rx) = pin 11/13"
echo "11003000 = uart1 (tx/rx) = pin 8/10"
```

```
echo "using $DEV"
echo "send data using \"echo \"AT\" >$DEV\""

while read line; do
#  if [[ -n "$line" ]]; then
    echo "["$(date "+%Y-%m-%d %H:%M:%S")"] received: "$line
#  fi
done < <(cat $DEV)
```

```
echo "AT" >/dev/ttyS2
```

einfaches Beispiel für Arduino (Nano)



```
PI             Levelshifter       Arduino
1 (3V3)  -------  LV   HV -------- 5V
6 (GND)  ---------  GND  --------- GND
8 (TX)   ------- LV2  HV2 -------- RX
10 (RX)  ------- LV1  HV1 -------- TX
```

PWM

Kernel-Option PWM_MEDIATEK muss gesetzt sein (Modul möglich), benötigt PWM(=y)

gpio 206 (pin 7) als pwm3 verwenden

```
echo 3 >/sys/class/pwm/pwmchip0/export
echo 200000 >/sys/class/pwm/pwmchip0/pwm3/period
echo 100000 >/sys/class/pwm/pwmchip0/pwm3/duty_cycle
echo 1 >/sys/class/pwm/pwmchip0/pwm3/enable
```

https://www.kernel.org/doc/Documentation/pwm.txt

```
period The total period of the PWM signal (read/write). Value is in
nanoseconds and is the sum of the active and inactive time of the PWM.
```

```
duty_cycle The active time of the PWM signal (read/write). Value is in
nanoseconds and must be less than the period.
```

```
period=200000ns=200ms=5Hz
duty_cycle=100000ns=1/2 period=50% high + 50% low Signal
```

aktuell ist aber die Ausgangsfrequenz nicht korrekt (statt 5kHz kommt 1kHz raus) siehe Forum und Fehlerreport

seit 2.3.2018 ist die Frequenz richtig: Commit in 4.14-main

SPI

http://forum.banana-pi.org/t/bpi-r2-spi-communication/4779/27

I2C

Echtzeituhr via i2c

https://tutorials-raspberrypi.de/raspberry-pi-rtc-modul-i2c-echtzeituhr/

ggf.

```
apt-get install i2c-tools
```

(benötigt unter ubuntu 18.4 universe in der /etc/apt/sources.list)

```
[17:13] root@bpi-r2:~# modprobe i2c-dev
[17:14] root@bpi-r2:~# i2cdetect -y 0
```

Realtime-Clock DS1307 (mit entfernten pullup-Widerständen) an i2c0 (I2C_SDA0=pin3, I2C_SCL0=pin5, 5V=pin4, GND=pin6)

```
#!/bin/bash
modprobe i2c-dev
modprobe rtc-ds1307
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-0/new_device
#cat /sys/class/i2c-dev/i2c-0/device/0-0068/rtc/rtc0/time
#read rtc
hwclock -r
#set system-clock to rtc-value
#hwclock -s
#set rtc to system-time
#hwclock -w
```

1)

Quelle: wiki.openwrt.org

2)

Quelle: commons.wikimedia.org