

Raspberry Pi Jam - Raspjamming

Autoren und Mitwirkende:

Martin Strohmayer
Christoph Wörgötter
Christof Hirndler
Manfred Wallner
Janka Pfragner

Version 18.4
26. April 2018
PDF Edition

Inhaltsverzeichnis

1	Installation Raspberry Pi	5
1.1	Raspbian	5
1.1.1	Etcher	5
1.1.2	dd Kommandozeilenprogramm	6
1.2	USB Gadget / OTG Modus Raspberry Pi Zero	6
1.2.1	Host (DHCP-Client)	6
1.3	Verbindung	7
1.3.1	SSH	7
1.3.2	SSH über Shell-Script	8
1.4	Einstellungen	8
1.5	Aktualisierungen und Programme	9
1.5.1	Geany - Entwicklungsumgebung	9
1.5.2	wiringPi C-Bibliothek / gpio Kommandozeilenprogramm	12
1.5.3	GPIOZero Python-Bibliothek / pinout Kommandozeilenprogramm	13
1.5.4	wiringPi mit C# und Mono	14
2	Projekte	15
2.1	LED	15
2.1.1	Shell	15
2.1.2	C	16
2.1.3	C#	17
2.1.4	Python	18
2.1.5	Assembler	18
2.2	Ampel	20
2.2.1	C	20
2.2.2	C#	21
2.2.3	Python	23
2.3	7-Segment Display	24
2.4	Temperatur-/Feuchtesensor DHT22/AM2303	26
2.5	Distanzsensor HC-SR04	27

Kapitel 1

Installation Raspberry Pi

1.1 Raspbian

Raspbian ist das offizielle Betriebssystem für den Raspberry Pi. Es kann in zwei unterschiedlichen Varianten heruntergeladen werden. Man kann zwischen einer minimalen Version (Lite) oder einer Version mit grafischer Oberfläche und vielen vorinstallierten Programmen wählen. Es wird in beiden Fällen mindestens eine 4 GB große MicroSD-Karte benötigt. Man sollte aber, um Reserven zu haben, mindesten 8 GB verwenden.

Zur Installation benötigt man ein beliebiges Linux System mit einem MicroSD-Kartenlesegerät. Die aktuelle Raspbian Version kann als Image-Datei von der Seite <https://www.raspberrypi.org/downloads/raspbian/> heruntergeladen werden. Bei dieser Anleitung wird Raspbian Stretch Lite verwendet.

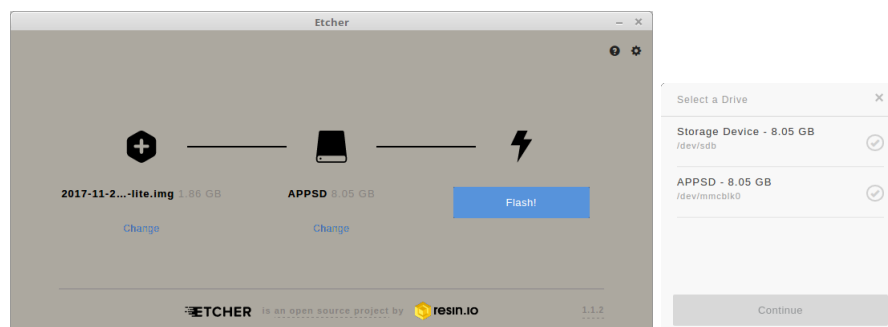
```
wget --trust-server-names http://downloads.raspberrypi.org/raspbian_lite_latest
unzip 2018-03-13-raspbian-stretch-lite.zip
rm 2018-03-13-raspbian-stretch-lite.zip
```

1.1.1 Etcher

Das grafische Programm Etcher (<https://etcher.io>) kann zum Übertragen der Image-Datei verwendet werden. Es ist vor Allem für Anfänger zu empfehlen, da beim Konsolenprogramm dd das Risiko besteht, dass Daten einer falschen Partition bzw. eines Laufwerks zerstört werden. Das Programm muss allerdings manuell installiert werden.

```
wget https://github.com/resin-io/etcher/releases/download/v1.3.1/etcher-1.3.1-linux-x86_64.zip
unzip etcher-1.3.1-linux-x86_64.zip
chmod +x etcher-1.3.1-linux-x86_64.AppImage
sudo mv etcher-1.3.1-linux-x86_64.AppImage /usr/local/bin/etcher
etcher &
```

Nach dem Starten wird danach gefragt ob eine Verknüpfung zum Programm erstellt werden soll. Dies sollte man mit „Yes“ beantworten. Danach kann man mit der Schaltfläche „Image“ die Image-Datei auswählen. Ist nur ein mögliches Ziel vorhanden, wird es bereits vorausgewählt, z. B. die SD-Karte im Karten-Slot (/dev/mmcblk0) oder im USB-Adapter (/dev/sdb). Sind mehrere mögliche Ziele vorhanden, wird die „Select Drive“ Schaltfläche freigeschaltet. Dann kann ein Laufwerk manuell ausgewählt werden.



Wenn man noch etwas ändern will, kann die entsprechende „Change“ Schaltfläche ausgewählt werden. Zum Schluss wird der Schreibvorgang mit der „Flash!“ Schaltfläche gestartet. Möglicherweise wird vom Programm allerdings noch das System-Passwort abgefragt. Das Laufwerk bzw. die Partitionen werden nun aus dem System ausgehängt und der Schreibvorgang gestartet. Der Fortschritt, die durchschnittliche Übertragungsrate und die Restlaufzeit werden während des Vorgangs angezeigt.

1.1.2 dd Kommandozeilenprogramm

Die erhaltene Image-Datei kann mit dem Kommandozeilenprogramm dd auf eine MicroSD-Karte übertragen werden.

Es ist unbedingt vor dem Ausführen des Befehls zu prüfen, ob das angegebene Laufwerk bzw. Device auch der vorgesehenen MicroSD-Karte entspricht!

Bei USB-Kartenlesern bzw. USB-Adaptern ist die Ermittlung des Devices leicht über die Systemmeldungen möglich.

```
dmesg | tail -n 10

scsi 3:0:0:0: Direct-Access      MXT-USB  Storage Device    1308 PQ: 0 ANSI: 0 CCS
sd 3:0:0:0: Attached scsi generic sgl type 0
sd 3:0:0:0: [sdb] 15730688 512-byte logical blocks: (8.05 GB/7.50 GiB)
sd 3:0:0:0: [sdb] Write Protect is off
sd 3:0:0:0: [sdb] Mode Sense: 03 00 00 00
sd 3:0:0:0: [sdb] No Caching mode page found
sd 3:0:0:0: [sdb] Assuming drive cache: write through
sdb: sdb1 sdb2
sd 3:0:0:0: [sdb] Attached SCSI removable disk
EXT4-fs (sdb2): mounted filesystem with ordered data mode. Opts: (null)
```

Die MicroSD-Karte wurde im Beispiel als Device „sdb“ über einen USB-Adapter eingebunden. Nun kann man die Image-Datei mit dem Programm dd auf die MicroSD-Karte übertragen. Mit dem Parameter „of“ muss der komplette Device-Name, in diesem Fall „/dev/sdb“, angegeben werden. Bei Parameter „if“ wird die entpackte Image-Datei angegeben. Die Blöckgröße bzw. der Cache wird mit Parameter „bs“ gesetzt. Eine größere Blockgröße erhöht die Schreibgeschwindigkeit. Sie wird im Beispiel mit 4 MB angegeben.

Zu Beachten ist, dass der USB-Massenspeicher möglicherweise bereits automatisch gemountet wurde. Dann sollte man die Partitionen mit dem Befehl „umount“ zuerst auswerfen.

```
umount /dev/sdb1 /dev/sdb2
dd if=2018-03-13-raspbian-stretch-lite.img of=/dev/sdc bs=4M

1323+0 Datensätze ein
1323+0 Datensätze aus
1387266048 Bytes (1,4 GB) kopiert, 127,6147 s, 10,4 MB/s
```

1.2 USB Gadget / OTG Modus Raspberry Pi Zero

Für den USB Gadget Modus der Raspberry Pi Zero wird bereits eine vorkonfigurierte MicroSD-Karte bzw. ein MicroSD-Karten-Image zur Verfügung gestellt. Darauf sind bereits alle Voreinstellungen durchgeführt worden.

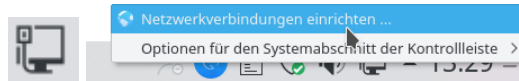
Sollte die Einrichtung manuell erfolgen oder wird Microsoft Windows als Host verwendet, so muss man sich das PDF-Dokument 'Raspberry Pi Jam - Raspjamming Admin' besorgen.

1.2.1 Host (DHCP-Client)

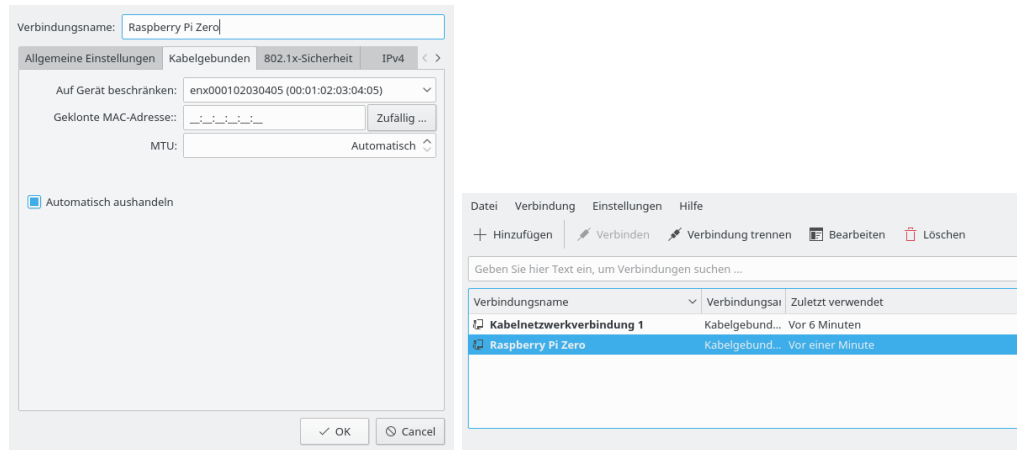
Kubuntu 16.04

Am Host-PC muss bei den IPv4-Einstellungen die Methode „Automatisch“ eingestellt sein. Wenn es sich um eine neue Verbindung handelt, ist dies bereits voreingestellt, eine Parametrierung kann dann entfallen. Ansonsten muss zur Konfiguration unter Linux (Kubuntu 16.04) zuerst der Dialog „Netzwerkverbindungen“ geöffnet werden.

Dazu klickt man mit der rechten Maustaste auf das Netzwerksymbol in Infobereich rechts unten. Dann kann die Option „Netzwerkverbindungen einrichten...“ ausgewählt werden.



Nun könnte die neue „Kabelnetzwerkverbindung“ umbenannt werden, z. B. in Raspberry Pi Zero. Erkennen kann man das Netzwerk an der Mac-Adresse, die man bei „g_ether.host_addr“ angegeben hat (z. B. 00:01:02:03:04:05).



Nun kann bei den IPv4-Einstellungen die Methode „Automatisch“ eingestellt werden.

Nach der Einrichtung des Netzwerks kann der Raspberry Pi Zero mit dem Namen „raspberrypi.local“ erreicht werden. Um den Raspberry Pi Zero mit dem Internet verbinden zu können, müssen einige Einstellungen am Host gemacht werden. Man muss den Namen des Netzwerkgeräts am Host-PC kennen, das mit dem Internet verbunden ist. Dies ermittelt man über die Netzwerkeinstellungen oder über den Terminal mit nmcli. Im Beispielfall ist der Name „enp0s25“ das richtige Gerät.

```
nmcli d

GERÄT      TYP      STATUS      VERBINDUNG
enx000102030405 ethernet verbunden Raspberry Pi Zero
enp0s25     ethernet verbunden Netzwerkverbindung 1
lo         loopback nicht verwaltet --
```

Damit der Internetzugang für den Raspberry Pi Zero freigegeben wird, müssen am Host-PC folgende Befehle im Terminal eingegeben werden. „enp0s25“ muss durch den Namen des Netzwerkgeräts ersetzt werden, das mit dem Internet verbunden ist.

```
sudo sysctl -w net.ipv4.ip_forward=1
sudo iptables -t nat -A POSTROUTING -o enp0s25 -j MASQUERADE
```

Alternativ können die Einstellungen auch automatisch über eine Shell-Script durchgeführt werden (siehe Kapitel 1.3.2 SSH über Shell-Script).

1.3 Verbindung

1.3.1 SSH

Nach der Einrichtung kann per SSH-Client eine Verbindungen zum Raspberry Pi hergestellt werden. Dazu muss in einem Terminal folgender Befehl eingegeben werden:

```
ssh -X pi@raspberrypi.local
```

Um grafische Programme am Host anzeigen zu können, muss der Parameter -X angegeben werden. Dann wird eine X-Server Verbindung via SSH hergestellt. Verbindet man sich zum ersten Mal mit dem Raspberry Pi, so wird noch eine Sicherheitswarnung ausgegeben. Der kryptographische Schlüssel für die Verbindung ist dem System noch nicht bekannt.

```
The authenticity of host 'raspberrypi.local (169.254.229.192)' can't be established.  
ECDSA key fingerprint is SHA256:Dcf3HYgE2GHnNz8Xhv8iJ9yA+zvfXBC9Com2eL9i0w.  
Are you sure you want to continue connecting (yes/no)?  
Warning: Permanently added 'raspberrypi.local,169.254.229.192' (ECDSA) to the list of known hosts.
```

Die Frage muss mit **yes** bestätigt werden. Anschließend muss das Default-Passwort von Raspbian **raspberry** eingegeben werden. Nun sollte man den Raspberry Pi Prompt **pi@rasbperrypi:~\$** sehen.

Wechselt man zu einem anderen Raspberry Pi mit den gleichen Namen, so kann es passieren, dass eine Fehlermeldung ausgegeben wird, weil sich der kryptographische Schlüssel geändert hat.

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
@      WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!      @  
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

In diesem Fall muss man die Verbindung aus den bekannten Hosts löschen. Hierfür muss folgendes Kommando ausgeführt werden:

```
ssh-keygen -R raspberrypi.local
```

Optional kann auch der Parameter **-o UserKnownHostsFile=/dev/null** beim **ssh**-Befehl hinzugefügt werden. Dann erfolgt keine Überprüfung der Verbindung.

1.3.2 SSH über Shell-Script

Am Einfachsten ist es die Verbindung zu Raspberry Pi über das vorbereitete **PiConnect.sh** Shell-Script herzustellen. Es setzt automatisch die Internetweiterleitung und startet die SSH-Verbindung. Es kann vom vorbereiteten Raspbian-Image oder von Github geladen werden.

```
scp pi@raspberrypi.local:/home/pi/scripts/PiConnect.sh .
```

oder

```
wget --trust-server-names https://goo.gl/uwguQo  
sh PiConnect.sh
```

1.4 Einstellungen

Folgende Einstellungen sind bei vorbereiteten Image bereits durchgeführt worden, zur Veranschaulichung aber nochmals explizit aufgelistet.

Als erstes kann das Raspberry Pi Software Configuration Tool (**raspi-config**) gestartet werden. Dazu verwendet man den Aufruf „**sudo raspi-config**“.

Der Menüpunkt „Change User Password“ ändert das Passwort für den Benutzer „pi“.

Danach sollte man die Regionseinstellungen mit dem Menüpunkt „Localisation Options“ einstellen. Im folgenden Untermenü kann man mit „Change Locale“ die Sprache und Zeichensatz des Systems setzen. Hier wählt man z. B. „de_AT.UTF-8 UTF8“ für Österreich oder „de_DE.UTF-8 UTF8“ für Deutschland. „en_GB.UTF-8 UTF8“ und „C.UTF-8“ sind bereits ausgewählt und können zusätzlich aktiv sein. Im nächsten Fenster kann man dann die Sprache „de_AT.UTF-8 UTF8“ bzw. „de_DE.UTF-8 UTF8“ als Standardeinstellung übernehmen.

Nun kann man mit „Change Timezone“ die aktuelle Zeitzone auswählen. Für das geografische Gebiet kann man „Europe“ auswählen. Danach kann man als Zeitzone die Stadt „Vienna“ oder „Berlin“ auswählen.

Bei der dritten Einstellung „Change Keyboard Layout“ kann man das Tastatur-Layout aktualisieren. Die vierte Einstellung „Change Wi-fi Country“ passt die WLAN-Einstellungen an die Ländervorgaben an (verfügbare Kanäle und Frequenzen). Hier kann „AT Austria“ für Österreich gewählt werden.

Nun kann noch die Speicherzuweisung der GPU auf lediglich 32 MB gesetzt werden. Dazu wählt man den Menüpunkt „Advanced Options“ und „Memory Split“, dann kann man 32 für die Speichergröße eingeben.

Sollte für die Root-Partition nicht die gesamte Kapazität der SD-Karte verwendet werden, kann über den Menüpunkt „Advanced Options“ und „Expand Filesystem“ eine Vergrößerung durchgeführt werden.

Beim Beenden des Programms muss ein Neustart (**reboot**) durchgeführt werden.

1.5 Aktualisierungen und Programme

Folgende Programme sind auf dem vorbereiteten Image bereits installiert. Demnach müssen die angeführten Anweisungen nur bei einem Raspbian Image ausgeführt werden. Auf die größeren Programme *CodeLite* und *Code::Blocks* wurde allerdings beim vorinstallierten Image verzichtet, da bei der Anleitung Geany verwendet wird.

```
sudo apt-get update
sudo apt-get -y upgrade
sudo apt-get install geany vim wiringpi pigpio git build-essential automake
sudo apt-get install minicom screen
sudo apt-get install python-dev python-openssl rpi.gpio
sudo apt-get install python3-dev python3-openssl python3-rpi.gpio python3-gpiozero python3-pip
sudo apt-get install mono-complete
sudo apt-get install codelite codeblocks
sudo apt-get clean
sudo reboot
```

Die Abkürzung *IDE* bezeichnet eine integrierte Entwicklungsumgebung (Integrated Development Environment). Sie stellt umfassende Funktionen und Programme zur Entwicklung von Programmen bereit. Die Hauptkomponenten sind Editor, Compiler und Debugger.

Eine Versionsverwaltung ist ein System, das zur Verwaltung, Archivierung und Erfassung von Änderungen an Source-Dateien verwendet wird. GitHub ist ein webbasierter Online-Dienst zur Versionsverwaltung, der viele Software-Entwicklungsprojekte bereitstellt.

CodeLite ist eine freie plattformübergreifende IDE, die auf die Programmiersprachen C, C++, PHP und JavaScript (Node.js) spezialisiert ist. *Code::Blocks* ist eine freie plattformübergreifende IDE für die Programmiersprachen C, C++ und Fortran.

Geany ist ein Texteditor mit grundlegenden Funktionen. Es wurde entwickelt, um eine kleine und schnelle IDE bereitzustellen. Nachteil gegenüber einer IDE wie *CodeLite* ist vor allem das Fehlen eines Debuggers und Probleme bei multiplen Projektdateien. Der Benutzer muss sich darum kümmern, dass die Kompileranweisung alle Sourcecode Komponenten einschließt. Compilerfehler werden im Source nicht hervorgehoben, sondern nur in einem Fenster ausgegeben. Für die einfachen Beispiele in dieser Anleitung ist es aber gut geeignet. Das Programm unterstützt alle wichtigen Entwicklungsumgebungen wie C, C++, C# und Python.

WiringPi und *pigpio* sind zwei C-Bibliotheken, die das Arbeiten mit den GPIOs der Raspberry Pi ermöglichen. Sie sind nicht kompatibel, man muss sich also für eine entscheiden. In der Anleitung wird ausschließlich die *WiringPi* Bibliothek benutzt. *git* ist ein Versionsverwaltungssystem das Zugriff auf Source-Dateien von Projekte ermöglicht. Diese werden zumeist auf Online-Dienst GitHub zur Verfügung gestellt. Das Paket *build-essential* enthält GNU C und C++ Compiler sowie die GNU-C-Bibliothek um C/C++-Projekte kompilieren bzw. erstellen zu können.

minicom und *screen* sind Programme, um auf der seriellen Schnittstelle (UART) der Raspberry Pi kommunizieren zu können.

python-dev bzw. *python3-dev* enthält Bibliothek und Entwicklungswerkzeuge zum Erstellen von Python-Scripte, sowie den Python-Interpreter selbst. In der Anleitung wird ausschließlich die aktuelle Version 3 von Python verwendet. *rpi.gpio* ist eine Python Bibliothek die Basisfunktionalitäten der GPIOs unterstützt. *gpiozero* ist eine aktuelle Python Bibliothek die viele Funktionen der GPIOs zur Verfügung stellt und von der Raspberry Pi Foundation empfohlen wird. Bei den Beispielen wird ausschließlich diese Library verwendet.

mono-complete enthält Compiler, Librarys und die Runtimeumgebung um CIL (Common Intermediate Language) Bytecode, auch als Assemblies bekannt, erzeugen und ausführen zu können. Es wird für die Erstellung von C# Programmen benötigt.

1.5.1 Geany - Entwicklungsumgebung

Nach dem Herstellen der Verbindung zur Raspberry Pi über SSH kann die grafische Entwicklungsumgebung Geany gestartet werden.

```
cd ~
geany &
```

Soll Geany in Englisch ausgeführt werden, obwohl das System auf Deutsch gestellt wurde, so muss man vor dem Start die Variable „LANG“ auf „C“ setzen.

```
LANG=C geany &
```

Nun kann eine Source-Datei geladen oder erstellt werden. Dann kann das Konfigurationsfenster geöffnet werden, indem man im Menü **Erstellen** → **Kommandos zum Erstellen konfigurieren** auswählt.

Die Parameter für **Kompilieren** und **Erstellen** werden bereits anhand der Source-Datei (Extension) vorbelegt. Zumeist müssen aber noch Anpassungen vorgenommen werden, um Bibliotheken oder geänderte Compiler verwenden zu können. Werden zwei Source-Dateien im Projekt benötigt, so müssen auch beide in der Anweisung vorkommen ("g++ -o Zielfeld Quelldatei1.cpp Quelldatei2.cpp"). Es können Platzhalter mit verschiedenen Funktionen eingefügt werden. **%f** wird durch den Dateinamen der im Editor ausgewählten Datei ersetzt. **%e** wird durch den Dateinamen ohne Extension der im Editor ausgewählten Datei ersetzt.

Was genau bei den Feldern einzutragen ist, wird in den folgenden Beispielpogrammen der einzelnen Programmiersprachen angegeben.

Danach kann das Programm mit der Taste **Erstellen** erzeugt werden und mit der Taste **Ausführen** in einem Terminal gestartet werden.

Die Tastenkürzel für alle Funktionen können über das Menü **Bearbeiten** → **Einstellungen** → **Tastenkürzel** vorgegeben werden. Dazu wählt man die gewünschte Aktion aus und drückt die Taste „Ändern“. Danach kann man die Taste bzw. Tastenkombination drücken, die dann umgehend im Dialog angezeigt wird. Wenn nun die OK-Taste gedrückt wird, wird die Änderung übernommen und das Tastenkürzel führt in Zukunft die Aktion aus. Im Beispiel wurde der Aktion „Erstellen“ dem Tastenkürzel bzw. der Taste **F7** zugewiesen.

Eine Beschreibung zur Erstellung und Parametrierung eines Projekts kann dem Kapitel 2.1 LED bzw. 2.2 Ampel entnommen werden.

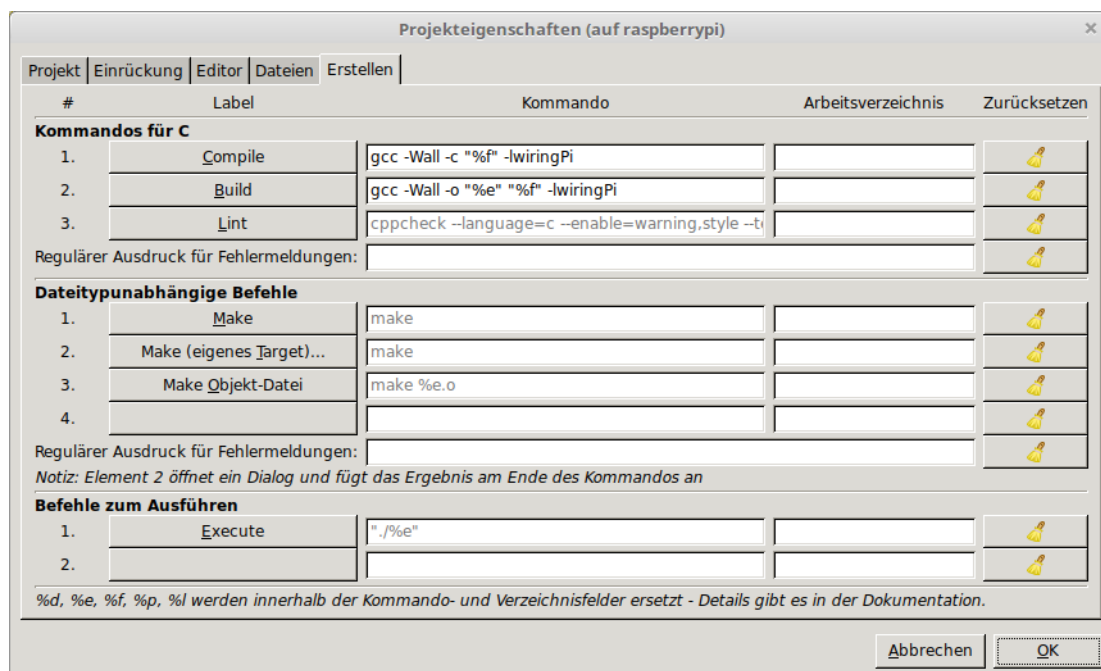


Abbildung 1.1: Kommandos zum Erstellen konfigurieren für C Projekt

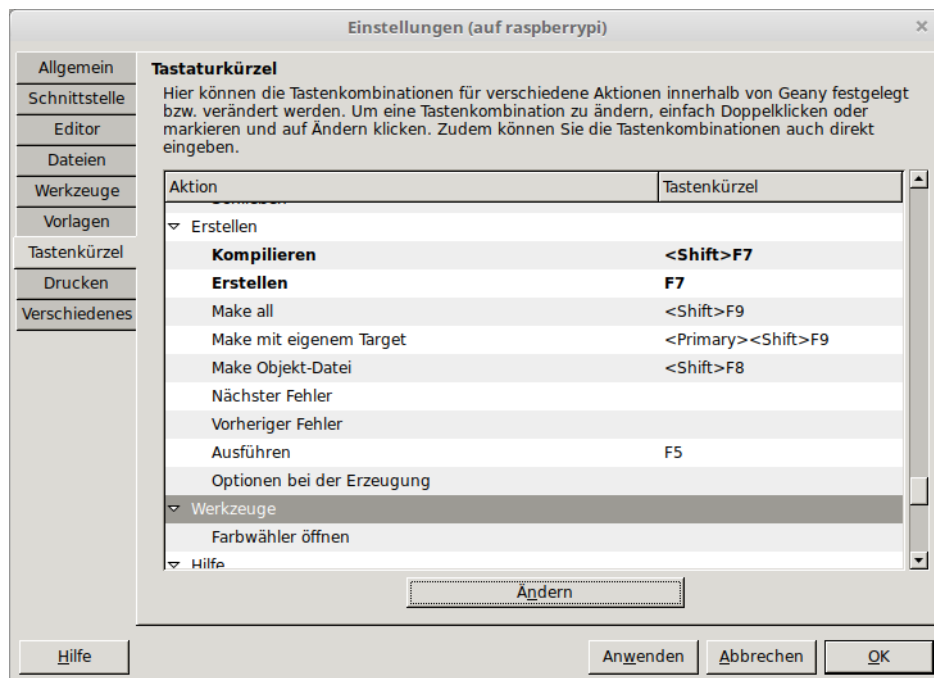


Abbildung 1.2: Tastaturkürzel

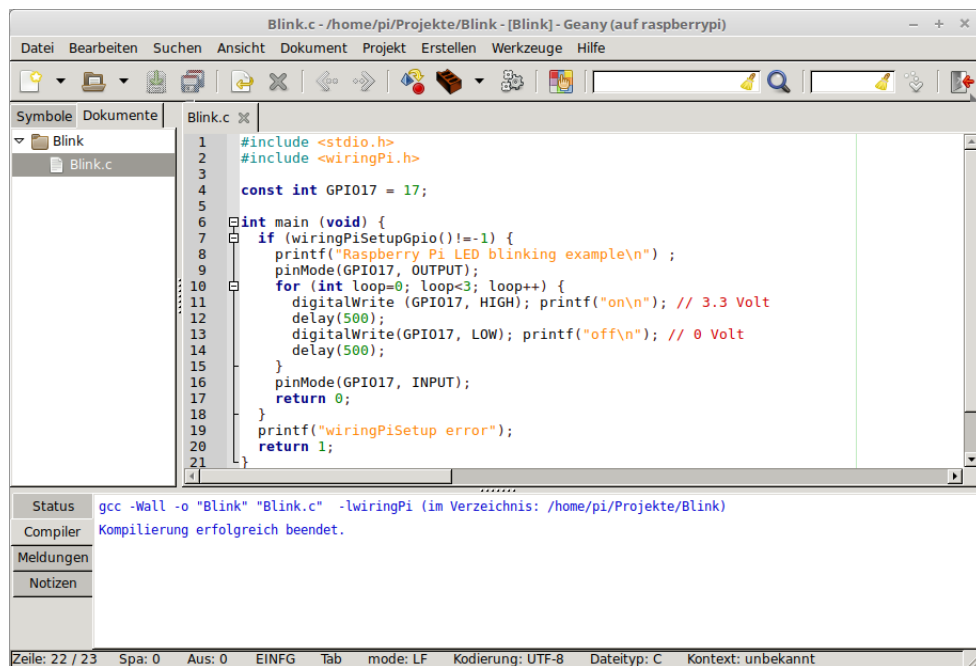


Abbildung 1.3: Geany Oberfläche mit Blink-LED Beispiel

1.5.3 GPIOZero Python-Bibliothek / pinout Kommandozeilenprogramm

GPIOZero ist eine der vielen Python Bibliotheken, welche genutzt werden kann um auf die GPIOs der Raspberry Pi zuzugreifen. Sie wird unter der BSD Lizenz angeboten. Das Paket beinhaltet auch das Kommandozeilenprogramm „pinout“. Dieses zeigt Hardwareinformationen und die Pin-Anordnung in der Kommandozeile an.

```
pinout

.------.
| ooooooooooooooooooooo J8 |
| loooooooooooooooooooo |c
---+      +---+ PiZero W|s
sd|      |SoC|  V1.1 |i
---+|hdmi| +---+  usb pwr |
`---|      |-----|  |-|  |-'

Revision          : 9000c1
SoC                : BCM2835
RAM               : 512Mb
Storage           : MicroSD
USB ports         : 1 (excluding power)
Ethernet ports    : 0
Wi-fi             : True
Bluetooth         : True
Camera ports (CSI) : 1
Display ports (DSI): 0

J8:
 3V3  (1) (2)  5V
GPIO2 (3) (4)  5V
GPIO3 (5) (6)  GND
GPIO4 (7) (8)  GPIO14
  GND (9) (10) GPIO15
GPIO17 (11) (12) GPIO18
GPIO27 (13) (14) GND
GPIO22 (15) (16) GPIO23
 3V3  (17) (18) GPIO24
GPIO10 (19) (20) GND
GPIO9  (21) (22) GPIO25
GPIO11 (23) (24) GPIO8
  GND (25) (26) GPIO7
GPIO0  (27) (28) GPIO1
GPIO5  (29) (30) GND
GPIO6  (31) (32) GPIO12
GPIO13 (33) (34) GND
GPIO19 (35) (36) GPIO16
GPIO26 (37) (38) GPIO20
  GND (39) (40) GPIO21

For further information, please refer to https://pinout.xyz/
```

Die Bezeichnung der Pins ist sehr wichtig, da es unterschiedlichste Arten gibt die Pins anzusprechen. Die GPIOZero Bibliothek verwendet per Default die Broadcom Nummerierung (BCM numbering). Das heißt, will man den physikalisch dritten Pin (direkt unter den 3.3V) schalten, muss man wissen dass dieser die Bezeichnung GPIO2 besitzt und demzufolge die Pin-Nummer 2 verwendet werden muss. Technisch kann das Pin-Schema durch das Austauschen der Pin-Factories der GPIOZero Bibliothek erreicht werden (*gpiozero.Device.pin_factory*). Hier wird aber nicht näher auf diese Möglichkeiten eingegangen.

1.5.4 wiringPi mit C# und Mono

Um mit C# Zugriff auf Funktionen der GPIOs zu erhalten, wird die wiringPi C-Bibliothek (siehe 1.5.2) und Mono benötigt.

Mono ist die quelloffene Implementierung von Microsofts .NET Framework und wird unter der MIT Lizenz angeboten.

Damit die C# Projekte aus Kapitel 2 kompiliert werden können, benötigt man einen Wrapper für die wiringPi Funktionen. Nachfolgend ist ein Auszug einer Implementierung eines C# wiringPi Wrappers angegeben.

```
namespace Raspjamming
{
    using System.Runtime.InteropServices;

    public class WiringPi
    {
        public const int Low = 0;
        public const int High = 1;
        public const int Input = 0;
        public const int Output = 1;

        [DllImport("libwiringPi.so", EntryPoint = "wiringPiSetupGpio")]
        public static extern int WiringPiSetupGpio();

        [DllImport("libwiringPi.so", EntryPoint = "pinMode")]
        public static extern void PinMode(int pin, int mode);

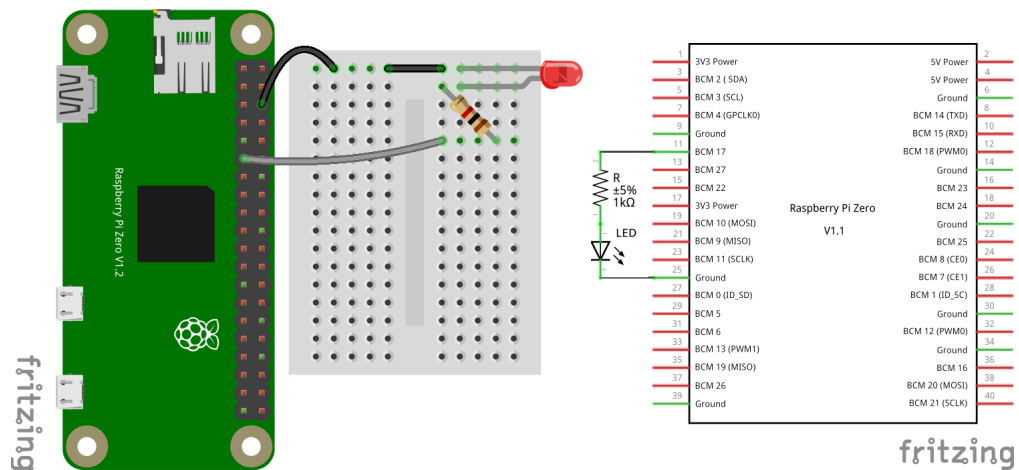
        [DllImport("libwiringPi.so", EntryPoint = "digitalWrite")]
        public static extern void DigitalWrite(int pin, int value);
    }
}
```

Ein vollständiger Wrapper kann z. B. unter <https://github.com/EvilVir/WiringPi.NET/raw/master/Wrapper/WiringPi.cs> bzw. <https://goo.gl/isrNeJ> heruntergeladen werden. Am vorbereitet Image ist der Wrapper unter `/home/pi/Projekte/wiringPi.cs` zu finden.

Kapitel 2

Projekte

2.1 LED



Lasse die LED blinken [Beispiele]
Verändere die Helligkeit der LED in dem du ein PWM-Signal erzeugst



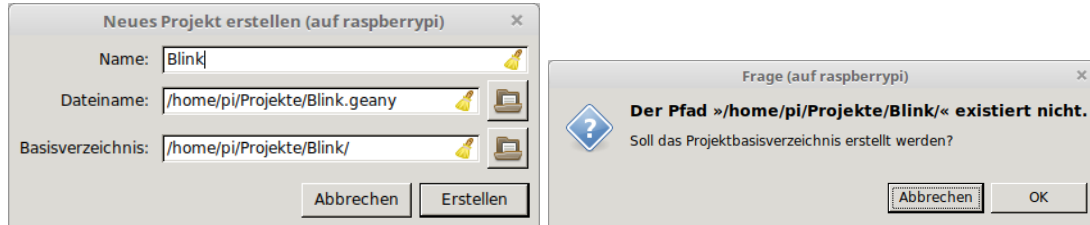
2.1.1 Shell

```
gpio -g mode 17 out
gpio -g write 17 1
gpio -g write 17 0
for i in {1..3}; do gpio -g write 17 1; sleep 1; gpio -g write 17 0; sleep 1; done
gpio -g mode 17 in
```

2.1.2 C

geany &

Zuerst wird ein neues Projekt erstellt. Dazu wählt man im Menü **Projekt** → **Neu...** Dann gibt man den Namen des Projekts an, z. B. Blink. Das Anlegen der Verzeichnisse muss danach auch noch bestätigt werden.



Weitere Einstellungen wie Zeichen für Einrückungen und Zeilenumbruch können unter **Projekt** → **Eigenschaften** vorgenommen werden.

Danach wählt man **Datei** → **Speichern unter** um die unbenannte Datei mit dem Namen „Blink.c“ speichern zu können. Nun kann man den folgenden C-Source eingeben.



```
#include <stdio.h>
#include <wiringPi.h>

const int GPIO17 = 17;

int main (void) {
    if (wiringPiSetupGpio() != -1) {
        printf("Raspberry Pi LED blinking example\n") ;
        pinMode(GPIO17, OUTPUT);
        for (int loop=0; loop<3; loop++) {
            digitalWrite (GPIO17, HIGH); printf("on\n"); // 3.3 Volt
            delay(500);
            digitalWrite(GPIO17, LOW); printf("off\n"); // 0 Volt
            delay(500);
        }
        pinMode(GPIO17, INPUT);
        return 0;
    }
    printf("wiringPiSetup error");
    return 1;
}
```

Im Menü muss man nun unter **Erstellen** → **Kommandos zum Erstellen konfigurieren** die WiringPi Library mit „-lwiringPi“ bei **Compile** und **Build** ergänzen.

Kommandos für C		
1.	Compile	gcc -Wall -c "%f" -lwiringPi
2.	Build	gcc -Wall -o "%e" "%f" -lwiringPi

Dann kann man das Projekt mit den Ziegel-Icon  erstellen bzw. kompilieren und danach mit dem Zahnrad-Icon  ausführen. Beendet wird das Programm mit der Tastenkombination **Strg**+**C**.

2.1.3 C#

geany &

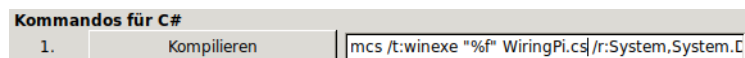
Zuerst wird ein neues Projekt erstellt. Dazu wählt man im Menü **Projekt** → **Neu...** Dann gibt man den Namen des Projekts an, z. B. `csBlink`. Das Anlegen der Verzeichnisse muss danach auch noch bestätigt werden. Danach wählt man **Datei** → **Speichern unter** um die unbenannte Datei mit dem Namen „*Blink.cs*“ speichern zu können. Nun kann man den folgenden C#-Source eingeben.

```
namespace Raspjamming
{
    using System;
    using System.Threading;

    static class Blink
    {
        const int GPIO17 = 17;



        static int Main()
        {
            if (WiringPi.WiringPiSetupGpio() != -1)
            {
                Console.WriteLine("Raspberry Pi LED blinking example");
                WiringPi.PinMode(GPIO17, WiringPi.Output);
                for (int loop = 0; loop < 3; loop++)
                {
                    WiringPi.DigitalWrite(GPIO17, WiringPi.High);
                    Console.WriteLine("on");
                    Thread.Sleep(500);
                    WiringPi.DigitalWrite(GPIO17, WiringPi.Low);
                    Console.WriteLine("off");
                    Thread.Sleep(500);
                }
                return 0;
            }
            Console.WriteLine("wiringPiSetup error");
            return 1;
        }
    }
}
```

Um das Programm kompilieren zu können, muss im Menü unter **Erstellen** → **Kommandos zum Erstellen konfigurieren** der Pfad zur Sourcedatei des C# WiringPi Wrapper (siehe 1.5.4) hinzugefügt werden. Hierfür im Textfeld **Kompilieren** den Pfad zu `WiringPi.cs` ergänzen „*mcs /t:winexe \"%f\"WiringPi.cs /r:System,System.Drawing*“.



Es muss die WiringPi Wrapper Datei in das Projektverzeichnis kopiert werden.

```
cp ~/Projekte/WiringPi.cs ~/Projekte/csBlink/
```

Anschließend kann man das Programm mit dem Kompilieren-Icon  erstellen bzw. kompilieren und mit dem Zahnrad-Icon  ausführen. Das Programm kann mit der Tastenkombination **Strg** + **C** vorzeitig beendet werden.

2.1.4 Python

geany &

Zuerst wird ein neues Projekt erstellt. Dazu wählt man im Menü **Projekt** → **Neu...** Dann gibt man den Namen des Projekts an, z. B. PyBlink. Das Anlegen der Verzeichnisse muss danach auch noch bestätigt werden. Danach wählt man **Datei** → **Speichern unter** um die unbenannte Datei mit dem Namen „*Blink.py*“ speichern zu können. Nun kann man den folgenden Python-Source eingeben.


```
# -*- coding: utf-8 -*-

from gpiozero import LED
import time

print("Raspberry Pi LED blinking example")
l = LED(17)
while 1:
    l.on()
    print(" on");
    time.sleep(1)
    l.off()
    print(" off");
    time.sleep(1)
```

Um das Programm auszuführen zu können, muss im Menü unter **Erstellen** → **Kommandos zum Erstellen konfigurieren** der Python Interpreter von Version 2 auf Version 3 umgestellt werden. Hierfür einfach im Textfeld **Compile** und im Textfeld **Execute** „*python*“ durch „*python3*“ ersetzen.

Kommandos für Python		
1.	Compile	python3 -m py_compile "%f"
2.		
3.	Lint	pep8 --max-line-length=80 "%f"
Regulärer Ausdruck für Fehlermeldungen: (.+):([0-9]+):([0-9]+)		
Dateitypunabhängige Befehle		
1.	Make	make
2.	Make (eigenes Target)...	make
3.	Make Objekt-Datei	make %e.o
4.		
Regulärer Ausdruck für Fehlermeldungen:		
Notiz: Element 2 öffnet ein Dialog und fügt das Ergebnis am Ende des Komr		
Befehle zum Ausführen		
1.	Execute	python3 "%f"

Anschließend kann man das Programm mit dem Zahnrad-Icon  ausführen. Achtung, nach dem Start braucht die Initialisierung der GPIOZero Library ein paar Sekunden, bevor das Programm startet. Beendet wird das Programm mit der Tastenkombination **Strg** + **C**.

2.1.5 Assembler

Echte Hardcore-Programmierer können natürlich auch das Beispiel in Assembler schreiben. ;-)

```
.data
.balign 4
@ WiringPi Pin 0 == BCM GPIO 17 == Physical 11
pin: .int 0
delayMs: .int 250

.text
.global main
.extern wiringPiSetup
.extern delay
.extern digitalWrite
.extern pinMode
```

```
main:
push  {ip, lr}
bl   wiringPiSetup

@ Set pin to output
ldr  r0, =pin
ldr  r0, [r0]
mov  r1, #1
bl   pinMode

@ Blink 10 times
mov  r2, #0
mov  r3, #10

loopBegin:
cmp  r2, r3
bgt  loopExit
@ Save registers
push {r2, r3}

ldr  r0, =pin
ldr  r0, [r0]
mov  r1, #1
bl   digitalWrite

ldr  r0, =delayMs
ldr  r0, [r0]
bl   delay

ldr  r0, =pin
ldr  r0, [r0]
mov  r1, #0
bl   digitalWrite

ldr  r0, =delayMs
ldr  r0, [r0]
bl   delay

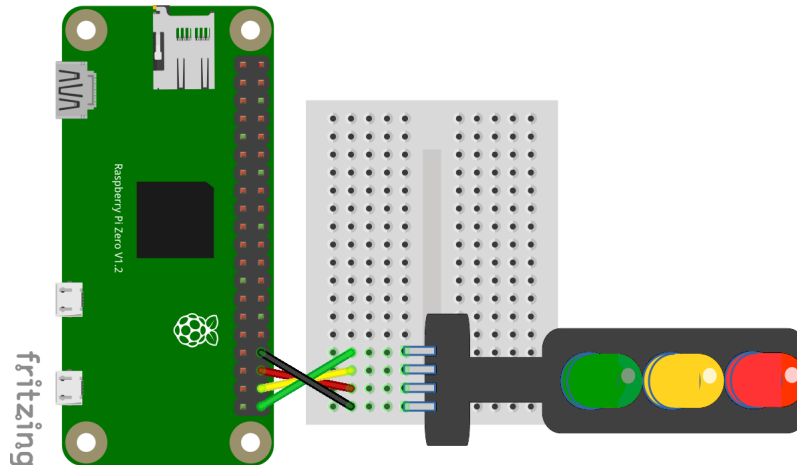
@ Restore loop state
pop  {r2, r3}
add  r2, #1
b    loopBegin

loopExit:
pop  {ip, pc}
```

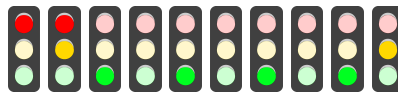
Um das Programm zu kompilieren und auszuführen müssen folgende Befehle in die Kommandozeile eingegeben werden:

```
as -o Blink.o Blink.asm
gcc -o Blink Blink.o -lwiringPi
./Blink
```

2.2 Ampel



Bilde die Standardfunktion einer österreichischen Ampel nach [Beispiele]
 Schalte über einen Eingang auf Rot für den Fußgängerübergang
 Überwache die CPU-Last und schalte bei über 90 % auf Rot und über 50 % auf Gelb



2.2.1 C

geany &

Zuerst wird ein neues Projekt erstellt. Dazu wählt man im Menü **Projekt** → **Neu...** Dann gibt man den Namen des Projekts an z. B. TrafficLight. Das Anlegen der Verzeichnisse muss danach auch noch bestätigt werden. Weitere Einstellungen wie Zeichen für Einrückungen und Zeilenumbruch können unter **Projekt** → **Eigenschaften** vorgenommen werden. Danach wählt man **Datei** → **Speichern** unter um die unbenannte Datei mit dem Namen „TrafficLight.c“ speichern zu können. Nun kann man den folgenden C-Source eingeben.

```
#include <stdio.h>
#include <wiringPi.h>
#include <signal.h>
#include <string.h>

//BCM GPIO connection
const int RED    = 16;
const int YELLOW = 20;
const int GREEN  = 21;
static sig_atomic_t end = 0;

static void sighandler(int signo) {
    end = 1;
}

void SetRYG(int Red, int Yellow, int Green) {
    digitalWrite(RED,    Red    );
    digitalWrite(YELLOW, Yellow);
    digitalWrite(GREEN,  Green );
}
```

```

void WaitMS(int MS) {
    if (!end) delay(MS);
}

int main(void) {
    struct sigaction sa;

    setbuf(stdout, NULL); // deactivate stdout buffering
    if (wiringPiSetupGpio() == -1) { // Init with BCM GPIO numbers
        perror("wiringPiSetup failed");
        return 1;
    }
    memset(&sa, 0, sizeof(struct sigaction)); //Handling stop events, like ctrl+c
    sa.sa_handler = sighandler;
    sigaction(SIGINT, &sa, NULL);
    sigaction(SIGQUIT, &sa, NULL);
    sigaction(SIGTERM, &sa, NULL);

    printf("Raspberry Pi traffic light (press ctrl+c to quit)\n");
    pinMode(RED, OUTPUT);
    pinMode(YELLOW, OUTPUT);
    pinMode(GREEN, OUTPUT);
    SetRYG(LOW, LOW, LOW);
    while(!end) {
        SetRYG(HIGH, LOW, LOW);
        WaitMS(5000);
        SetRYG(HIGH, HIGH, LOW);
        WaitMS(1000);
        SetRYG(LOW, LOW, HIGH);
        WaitMS(4000);
        for (int nBlinkCount=0; nBlinkCount<3; nBlinkCount++) {
            if (end) break;
            SetRYG(LOW, LOW, LOW);
            WaitMS(1000);
            SetRYG(LOW, LOW, HIGH);
            WaitMS(1000);
        }
        SetRYG(LOW, HIGH, LOW);
        WaitMS(2000);
    }
    SetRYG(LOW, LOW, LOW);
    pinMode(RED, INPUT);
    pinMode(YELLOW, INPUT);
    pinMode(GREEN, INPUT);
    return 0;
}

```

Im Menü unter Erstellen → Kommandos zum Erstellen konfigurieren die WiringPi Library mit „*lwiringPi*“ bei Compile und Build zu ergänzen.

Kommandos für C		
1.	Compile	gcc -Wall -c "%f" -lwiringPi
2.	Build	gcc -Wall -o "%e" "%f" -lwiringPi

Nun kann man das Projekt mit den Ziegel-Icon  erstellen bzw. kompilieren und danach mit dem Zahnrad-Icon  ausführen.

2.2.2 C#

geany &

Zuerst wird ein neues Projekt erstellt. Dazu wählt man im Menü **Projekt** → **Neu...** Dann gibt man den Namen des Projekts an, z. B. csTrafficLight. Das Anlegen der Verzeichnisse muss danach auch noch bestätigt werden. Nachfolgend unter **Datei** → **Speichern unter** die unbenannte Datei mit dem Namen „TrafficLight.cs“ speichern. Nun kann man den folgenden C#-Source eingeben.

```
namespace Raspjamming
```

```

{
    using System;
    using System.Threading;

    static class TrafficLight
    {
        const int Red = 16;
        const int Yellow = 20;
        const int Green = 21;
        private static volatile bool _end = false;

        static void Console_CancelKeyPress(object sender, ConsoleCancelEventArgs e)
        {
            e.Cancel = true;
            _end = true;
        }

        static void SetRYG(int red, int yellow, int green)
        {
            WiringPi.DigitalWrite(Red, red);
            WiringPi.DigitalWrite(Yellow, yellow);
            WiringPi.DigitalWrite(Green, green);
        }

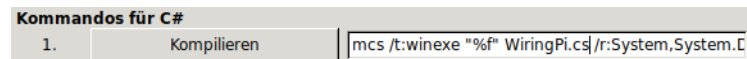
        static void WaitMS(int ms)
        {
            if (!_end)
            {
                Thread.Sleep(ms);
            }
        }

        static int Main()
        {
            Console.CancelKeyPress += Console_CancelKeyPress;
            // Init with BCM GPIO numbers
            if (WiringPi.WiringPiSetupGpio() == -1)
            {
                Console.WriteLine("wiringPiSetup failed");
                return 1;
            }
            Console.WriteLine("Raspberry Pi traffic light (press ctrl+c to quit)");
            WiringPi.PinMode(Red, WiringPi.Output);
            WiringPi.PinMode(Yellow, WiringPi.Output);
            WiringPi.PinMode(Green, WiringPi.Output);
            SetRYG(WiringPi.Low, WiringPi.Low, WiringPi.Low);
            while (!_end)
            {
                SetRYG(WiringPi.High, WiringPi.Low, WiringPi.Low);
                WaitMS(5000);
                SetRYG(WiringPi.High, WiringPi.High, WiringPi.Low);
                WaitMS(1000);
                SetRYG(WiringPi.Low, WiringPi.Low, WiringPi.High);
                WaitMS(4000);
                for (int blinkCount = 0; blinkCount < 3; blinkCount++)
                {
                    if (_end)
                    {
                        break;
                    }
                    SetRYG(WiringPi.Low, WiringPi.Low, WiringPi.Low);
                    WaitMS(1000);
                    SetRYG(WiringPi.Low, WiringPi.Low, WiringPi.High);
                    WaitMS(1000);
                }
                SetRYG(WiringPi.Low, WiringPi.High, WiringPi.Low);
                WaitMS(2000);
            }
            SetRYG(WiringPi.Low, WiringPi.Low, WiringPi.Low);
            WiringPi.PinMode(Red, WiringPi.Input);
            WiringPi.PinMode(Yellow, WiringPi.Input);
            WiringPi.PinMode(Green, WiringPi.Input);
            return 0;
        }
    }
}

```



```
}
}
}
```

Um das Programm kompilieren zu können, muss im Menü unter **Erstellen** → **Kommandos** zum **Erstellen konfigurieren** der Pfad zur Sourcedatei des C# WiringPi Wrapper (siehe 1.5.4) hinzugefügt werden. Hierfür im Textfeld **Kompilieren** den Pfad z. B. WiringPi.cs ergänzen. „mcs /t:winexe \"%f\"WiringPi.cs /r:System,System.Drawing“.



Es muss die WiringPi Wrapper Datei in das Projektverzeichnis kopiert werden.

```
cp ~/Projekte/WiringPi.cs ~/Projekte/csTrafficLight/
```

Anschließend kann man das Programm mit dem Kompilieren-Icon  erstellen bzw. kompilieren und mit dem Zahnrad-Icon  ausführen. Das Programm kann mit der Tastenkombination **Strg**+**C** vorzeitig beendet werden.

2.2.3 Python

Zuerst wird ein neues Projekt erstellt. Dazu wählt man **Projekt** → **Neu...** Dann gibt man den Namen des Projekts an, z. B. pyTrafficLight. Das Anlegen der Verzeichnisse muss danach auch noch bestätigt werden. Nachfolgend unter **Datei** → **Speichern** unter die unbenannte Datei mit dem Namen „TrafficLight.py“ speichern. Nun kann man den folgenden Python-Source eingeben.


```
# -*- coding: utf-8 -*-

from gpiozero import TrafficLights
from time import sleep

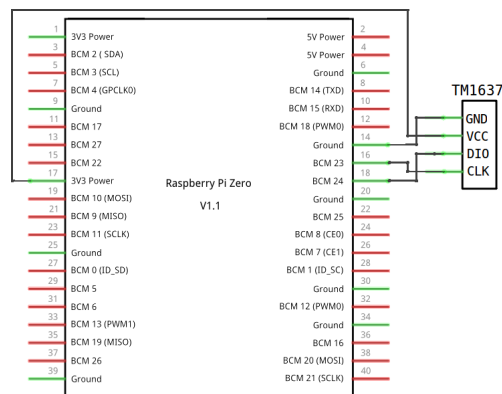
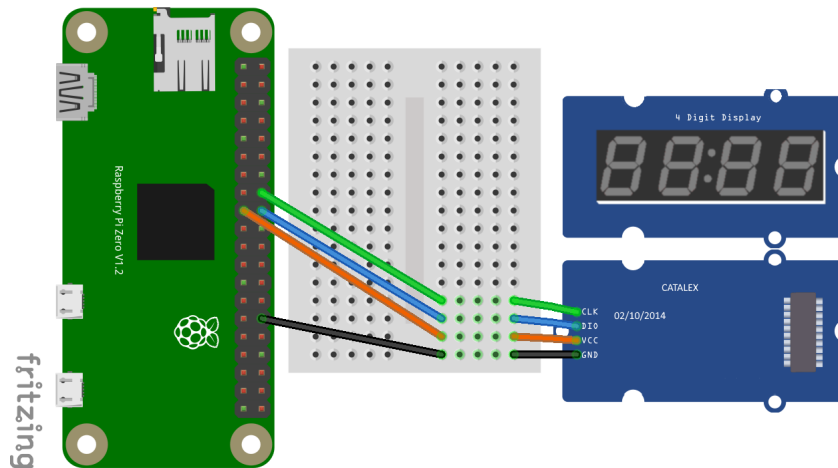
def SetRYG(red, yellow, green):
    if not green:
        lights.green.off()
    else:
        lights.green.on()
    if not red:
        lights.red.off()
    else:
        lights.red.on()
    if not yellow:
        lights.yellow.off()
    else:
        lights.yellow.on()

lights = TrafficLights(16, 20, 21)

while True:
    sleep(10)
    SetRYG(red=True, yellow=False, green=False)
    sleep(5)
    SetRYG(red=True, yellow=True, green=False)
    sleep(1)
    SetRYG(red=False, yellow=False, green=True)
    sleep(4)
    for i in range(3):
        SetRYG(red=False, yellow=False, green=False)
        sleep(1)
        SetRYG(red=False, yellow=False, green=True)
        sleep(1)
    SetRYG(red=False, yellow=True, green=False)
    sleep(2)
```

Anschließend kann man das Programm mit dem Zahnrad-Icon  ausführen.

2.3 7-Segment Display



Gib Zahlen und Texte aus [Beispiele]
 Gib die aktuelle CPU Temperatur aus [Beispiel Python]
 Gib die gemittelte aktuelle CPU Last in Prozent aus
 Realisiere eine Laufschrift



C:

```
git clone https://github.com/mstroh76/TM1637Display
cd TM1637Display
g++ -o TM1637Display *.cpp -lwiringPi
./TM1637Display
geany project.geany &
```

C#:

```
git clone https://github.com/chirndler/wiringpi.net.sensors.git
cd wiringpi.net.sensors
xbuild /p:Configuration=Release wiringpi.net.sensors.sln
cd bin/Release/
sudo mono wiringpi.net.sensors.sample.exe 3
```


Python:

```
sudo pip3 install wiringpi
git clone https://github.com/depklyon/raspberrypi-python-tm1637.git
cd raspberrypi-python-tm1637
sudo python3 setup.py install
```

```
# -*- coding: utf-8 -*-

from tm1637 import TM1637
from gpiozero import CPUTemperature
from time import sleep

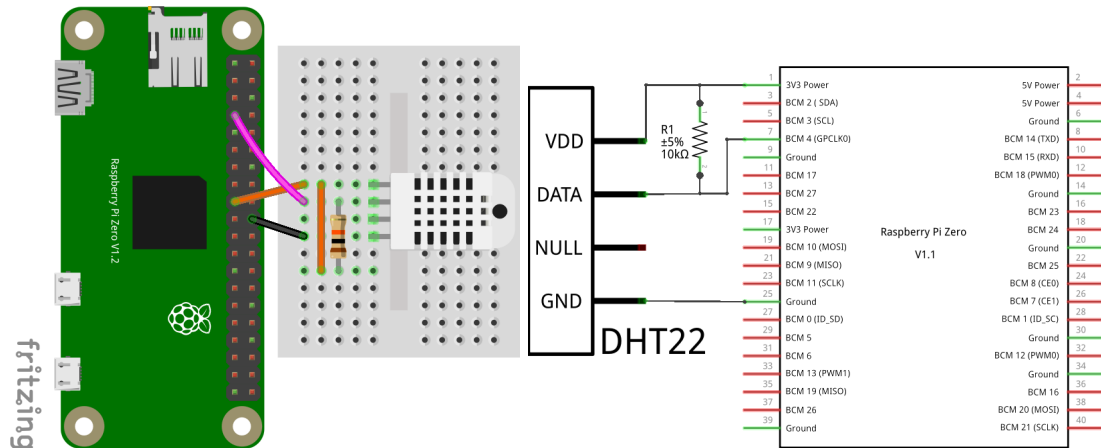
CLK = 10
DIO = 9

cpuTemp = CPUTemperature()
tm = TM1637(clk=CLK, dio=DIO)
# 0 ... off
# 7 ... max
tm.brightness(3)

while True:
    tm.temperature(round(cpuTemp.temperature))
    sleep(1)
```

```
python3 TM1637_Temp.py
```

2.4 Temperatur-/Feuchtesensor DHT22/AM2303



Lies Temperatur und Feuchte aus [Beispiele]
Gib die Werte zyklisch am TM1637 Display aus



C:

```
git clone https://github.com/mstroh76/Sensors-WiringPi.git
cd Sensors-WiringPi/DHT
g++ -o DHT *.cpp -lwiringPi
watch -n 2 ./DHT 4
cd ..
geany DHT.geany &
```

C#:

```
git clone https://github.com/chirndler/wiringpi.net.sensors.git
cd wiringpi.net.sensors
xbuild /p:Configuration=Release wiringpi.net.sensors.sln
cd bin/Release/
sudo mono wiringpi.net.sensors.sample.exe 1
```

Python:

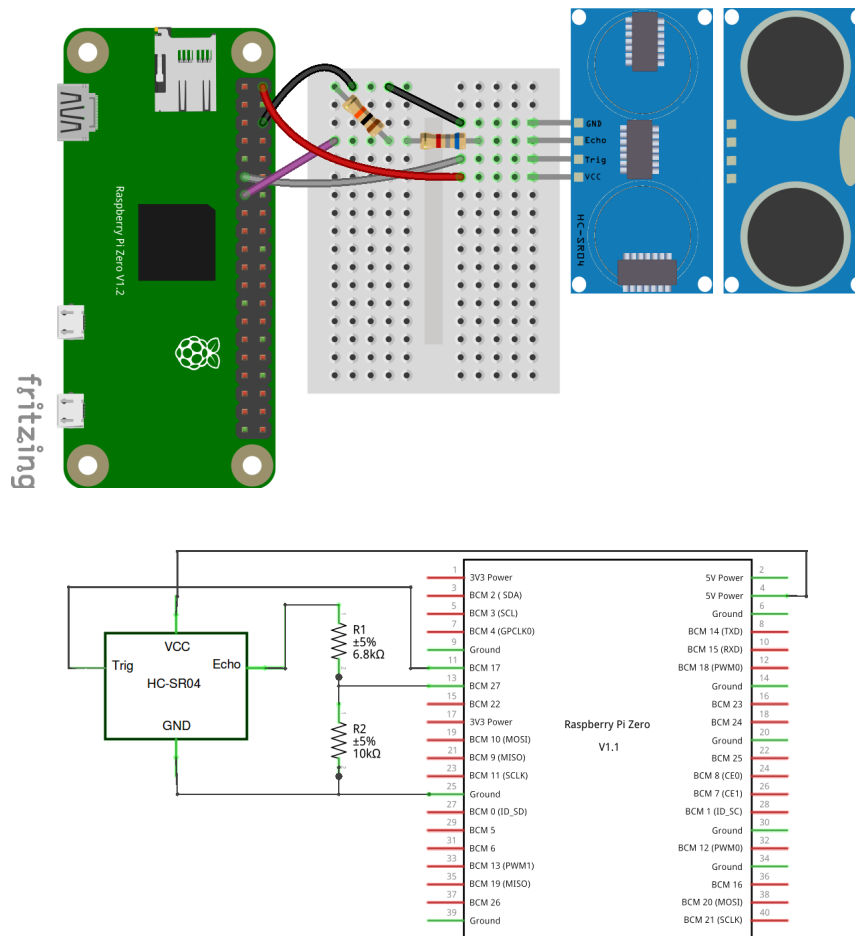
```
git clone https://github.com/jdupl/dhtxx-rpi-python3.git
sudo cp dhtxx-rpi-python3/dhtxx.py /usr/local/lib/python3.5/dist-packages/
```

```
# -*- coding: utf-8 -*-
from time import sleep
from dhtxx import DHT22

dht = DHT22(4)
while True:
    try:
        res = dht.get_result_once()
        print('Temp: ' + '{:.2f}'.format(res[0]) + '°C, Hum: ' + '{:.2f}'.format(res[1]))
    except Exception as e:
        print(e)
    sleep(1)
```

python3 DHT22.py

2.5 Distanzsensor HC-SR04



Miss die Distanz [Beispiele]
 Filtere die gemessene Distanz
 Ermittle die Geschwindigkeit eines bewegten Objektes
 Bewerte die Distanz mit der Ampel: Rot niedriger Abstand, Grün ausreichender Abstand
 Gib den Wert zyklisch am TM1637 Display aus [Beispiel Python]
 Verwende die korrigierte Schallgeschwindigkeit bei aktueller Lufttemperatur vom DHT22



C:

```

git clone https://github.com/mstroh76/Sensors-WiringPi.git
cd Sensors-WiringPi/HC-SR04
g++ -o HC-SR04 *.cpp -lwiringPi
./HC-SR04
cd ..
geany HC-SR04.geany &
  
```

C#:

```

git clone https://github.com/chirndler/wiringpi.net.sensors.git
cd wiringpi.net.sensors
xbuild /p:Configuration=Release wiringpi.net.sensors.sln
cd bin/Release/
sudo mono wiringpi.net.sensors.sample.exe 2
  
```

Python:

```
# -*- coding: utf-8 -*-

from time import sleep
from gpiozero import DistanceSensor
# Be aware not to use a GPIO with a physical pull-up resistor
# like GPIO3, otherwise gpiozero will raise an exception because
# it can't deactivate the pull-up
ds = DistanceSensor(echo=27, trigger=17)

# Measure and print distance periodically
while True:
    print(str(round(ds.distance*100)) + ' cm')
    sleep(1)

# ... Or print distance on 7-segment display
from tm1637 import TM1637
tm = TM1637(clk = 10, dio = 9)
tm.brightness(3)
while True:
    dist = str(round(ds.distance*100))
    tm.write(tm.encode_string(' '*(4-len(dist))+dist))
    sleep(1)
```

python3 HC_SR04.py

Anhang

GPIO's Raspberry Pi Zero



3,3 V	1	2	5 V
I2C1 SDA, GPIO02	3	4	5 V
I2C1 SCL, GPIO03	5	6	GND
GPCLK0, GPIO04	7	8	GPIO14, UART TX
GND	9	10	GPIO15, UART RX
GPIO17	11	12	GPIO18, PWM0 I2S BCK/CLK
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3,3 V	17	18	GPIO24
SPI MOSI, GPIO10	19	20	GND
SPI MISO, GPIO09	21	22	GPIO25
SPI SCLK, GPIO11	23	24	GPIO08, SPI CE0
GND	25	26	GPIO07, SPI CE1
I2C0 SDA, GPIO00 (Reserviert)	27	28	GPIO01, I2C0 SCL (Reserviert)
GPCLK1, GPIO05	29	30	GND
GPCLK2, GPIO06	31	32	GPIO12, PWM0
PWM1, GPIO13	33	34	GND
PWM1, GPIO19 I2S LRCK/SYNC	35	36	GPIO16 I2S DIN
GPIO26	37	38	GPIO20 I2S DATA/DOUT
GND	39	40	GPIO21

Lizenz



Dieses Werk steht unter der Lizenz Creative Commons BY-SA 3.0 (<https://creativecommons.org/licenses/by-sa/3.0/at>). Sie erlaubt ausdrücklich, das Werk zu vervielfältigen, zu verbreiten und öffentlich zugänglich machen. Es ist weiters erlaubt, dieses Werk zu verändern und darauf aufbauend zu erweitern. Es muss allerdings der Urheber genannt werden und die aufbauende Arbeit muss unter der gleichen Lizenz stehen.

Die Anleitung enthält Teile aus anderen E-Book's des Autors Martin Strohmayer, diese können über

Amazon <https://www.amazon.de/-/e/B071HJ6GYJ> und

Google <https://play.google.com/store/books/author?id=Martin+Strohmayer> bezogen werden.

Wenn sie die Arbeit des Autors unterstützen wollen, erwerben Sie das E-Book. Danke!

Es wurden freie (CC0) Grafiken von Openclipart verwendet <https://openclipart.org>.

Schaltpläne und Ansichten der Steckplatine wurden mit Fritzing erstellt <http://fritzing.org>.

Es wurden Fritzing Komponenten von Adafruit <https://github.com/adafruit/Fritzing-Library> und Ricky Ng-Adam und Yihui Xiong <https://github.com/mcauser/seed-fritzing-parts> verwendet. Sie werden ebenfalls unter der CC-BY-SA Lizenz zur Verfügung gestellt.

Haftungsausschluss

Die Benutzung dieser Anleitung und die Umsetzung der darin enthaltenen Informationen erfolgt ausdrücklich auf eigenes Risiko. Haftungsansprüche gegen die Autoren für Schäden materieller oder ideeller Art, die durch die Nutzung oder Nichtnutzung der Informationen bzw. durch die Nutzung fehlerhafter und/oder unvollständiger Informationen verursacht wurden, sind grundsätzlich ausgeschlossen. Rechts- und Schadensersatzansprüche sind daher ausgeschlossen. Das Werk inklusive aller Inhalte wurde unter größter Sorgfalt erarbeitet. Die Autoren übernehmen jedoch keine Gewähr für die Aktualität, Korrektheit, Vollständigkeit und Qualität der bereitgestellten Informationen. Druckfehler und Falschinformationen können nicht vollständig ausgeschlossen werden. Für die Inhalte, der in dieser Anleitung abgedruckten Internetseiten, sind ausschließlich die Betreiber der jeweiligen Internetseiten verantwortlich. Die Autoren haben keinen Einfluss auf Gestaltung und Inhalte fremder Internetseiten. Die Autoren distanzieren sich daher von allen fremden Inhalten.