RpiZeroW Debug-WLANBridge

Ein leicher Einstieg für Micropython-Umsteiger in die C Programmierung mit dem pico-c-sdk

Das Tutorial ist nicht für ...

Micropython geeignet. Wer einen Pico programmieren will, sieht die Entscheidung der Programmiersprache meist auf schnellen Erfolg fokussiert. In solch einer Betrachtung ist micropython unschlagbar und dieses Tutorial belastet diejenigen welche auf Python setzten nur mit Unnötigem

Wer aber die hardwarenahe Programmiersprache C lernen will, und bis jetzt nur wenig bis gar keinen Umgang mit Compilern, Bibliotheken, … zu tun hatte, steht oft vor einem Berg an Konfigurationsarbeit des Entwicklungssystems so das die tatsächlich hardwarenahe Programmierung in C darin untergeht. Die Komplexität der Linux-Entwicklung ist um vielfaches höher als der eines Pico, so ist der Aufwand den gemeine Pythonumsteiger in ein brauchbares Entwicklungssetup investieren müssen, oft difizil. Genau hier setzt dieses Tutorial an, an dessen Ende mit der Geany-IDE ein erstes printf auf der seriellen Konsole und „Hallo Welt“ Blinken kompiliert und auf den Pico übertragen/ausgeführt wird, ohne sich Gedanken darum machen zu müssen. Denn für die eigentliche Programmierung/Entwicklung ist das pico-c-sdk und dessen Studium entscheidend und nicht das Entwicklungssystem. Ihr braucht keine Angst vor dem pico-c-sdk haben, wenn ihr ein paar Beispielprogramme abgetippt habt, könnt ihr das pico-c-sdk schrittweise nach eurem Entwicklungsbedarf einfach „abfrühstücken“. Denn dessen Dokumentation ist wie der von Python präzise systematisch, detailliert und umfänglich aber nicht viel. Zudem obendrein steht eine primitive aber schnelle Ausführungsmaschine mit dem Namen „PIO-Statemaschine“ zur Verfügung, mit der sich alle Protokolle einfach, zeitlich präzise umsetzen lassen die nicht durch vorgefertigte Funktionen des pico-c-sdk abgebildet sind.

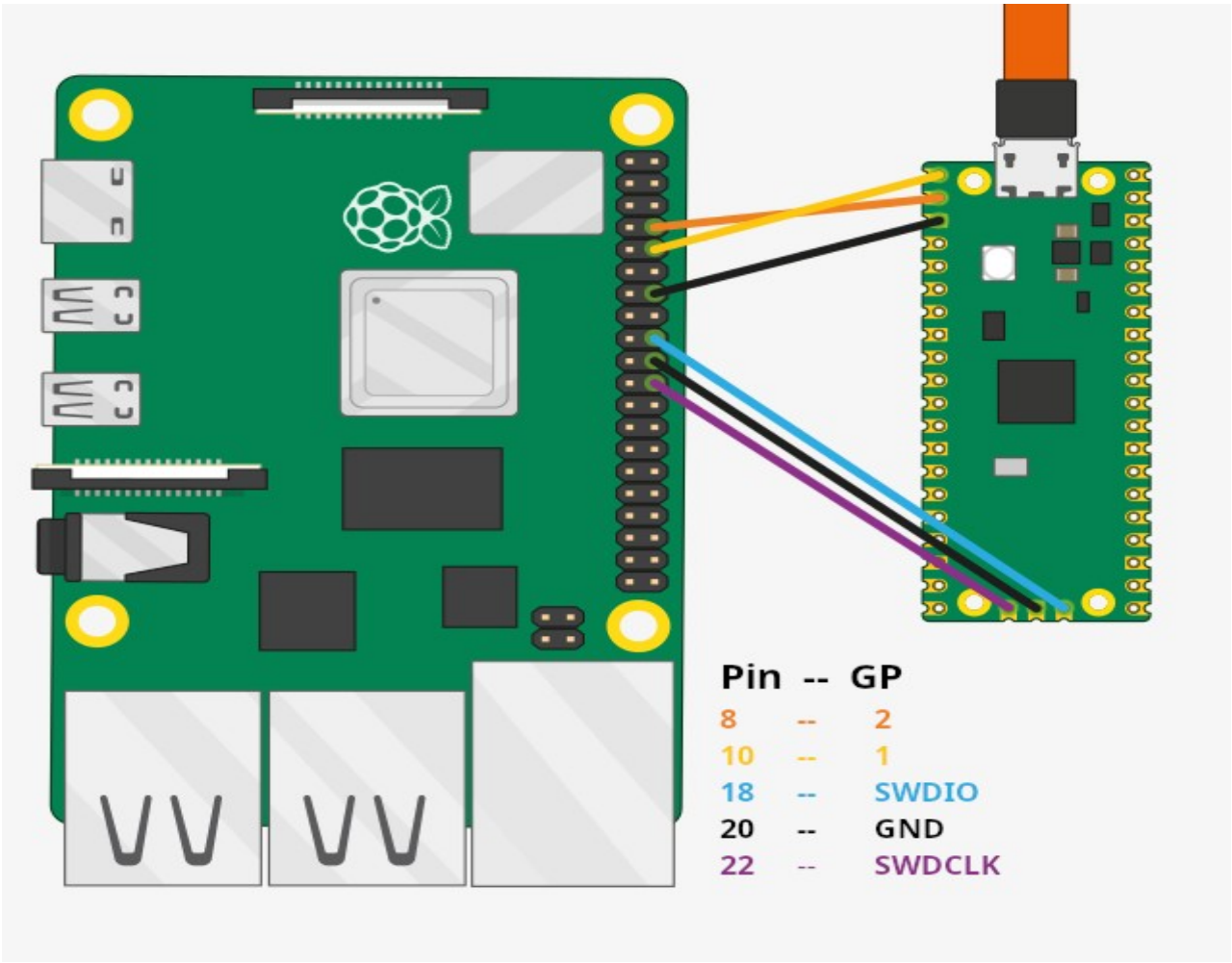
Die Erfahrung zeigt, das ständiges Hantieren mit den GPIO’s des Pico und daran angeschlossener Sensoren/Aktoren am Entwicklungsrechner, eine unpraktikable zeit- und nerven-aufreibende Tätigkeit sind die besonders zu beginn den Entwicklungsprozess ungemein bremsen. In diesem Setup ist ein RPiZeroW mit einem Pico Verbunden, so das diese Beiden gemeinsam als „W-Lan Hardware-Programmier/Test/Debug“ System für Pico-Microcontroller betrachtet sind , welches von einem Dektop-Linuxrechner mit der Geany-IDE programmiert wird.

Voraussetzungen

- W-LAN mit Internetzugang
- Einen RPi mit W-LAN/SSH-Login
- Einen Pi Pico (mit Micro-USB-Kabel zur Spannungsversorgung)
- Einen Dektop-Linuxrechner/Rpi mit W-Lan als Entwicklungsrechner/Programmierumgebung
- Verbindungskabel/JumperWire

Hardware-Verbindungen

Im Bild sehen wir die Verbindungen vom RPi-ZeroW zum Test-Pico.
Interessant ist, das die Spannungsversorgung des Pico im Bild extern über USB ist, da diese in der realen Testphase der Projekte ebenfalls real getestet werden sollte.



Für erste „Gehversuche“ mit der Programmierung des Pico ohne stromhungrige Hardware , kann zusätzlich im Bild oben eine Verbindung von Pin 2 des RpiZeroW zu VBUS des Pico hergestellt werden, so das eine USB-Spannungsversorgung des Pico nicht mehr in Betracht gezogen werden muss, und auch ohne richtig gepolte Sperrdiode dazwischen nicht sollte!

Löten?

Eine weitere Hürde sind die beiden SWD-CKL/IO Kontakte des Pico, da sie selten bis gar nicht mit Stiftleisten verlötet zu erwerben sind, kann man hier auch ganz einfach selbst: Die Loch-Kontaktleiste des Pico von oben in die kurzen Stiftleisten stecken, die nun schief/verkantete Konstruktion aus Pico und Stiftleiste mit Tesaband über das gegenüberliegende Ende des Picos mit der Unterlage/Tisch fixieren, als dann Lötzinndraht kurz auf die Spitze des Lötkolbens heben so das an der Kolbenspitze ein Tropfen hängen bleibt, dann die Lötdraht-Spitze an die vordere Kante zwischen Stiftleiste und Loch des Pico heben, und dann die Kolbenspitze mit dem Tropfen auf die Lötdrahtspitze an der Kante nur leicht andrücken. Das Zinn verläuft ins Loch, Kolben weg, fertig. Das hört sich nicht nur einfach an sondern gelingt auch so. Oft genügt es gar nur die Kolbenspitze mit dem Tropfen, ohne die Lötdrahtspitze leicht an die Kante zwischen Stiftleiste und Loch zu drücken oder „dranheben“ um das Zinn ins Loch fließen zu lassen.

Vorgehensweise/Zielsetzung

Mit der fertig verdrahteteten Hardware werden wir noch ein paar wenige Befehlszeilen im SSH-Terminal des RPiZeroW und dem des Entwicklungsrechners ausführen um, im dringend zum Studuim empfohlenen ../getting-started-with-pico.pdf praktisch ins Kapitel 6 springen zu können wenn wir wollen! Denn vom Ende diesen Tutorials an bis in die Tiefen des pico-c-sdk funktioniert die folgende Debug-Schrittfolge bei Projekten:

1. W-Lan-Programmiergerät (RpiZeroW) mit USB-5V versorgen
2. Einmalig Terminal am Entwicklungsrechner öffnen und „new_project mein_projektName“ eingeben
3. Einmalig F9 (bei mehrseitigen Projekten auch wiederholt) drücken
4. Zu Beginn des Zyklus im Geany-Menue auf „Erstellen-→Seriell“ klicken
5. Code eintippen
6. shift-F9 danch F5 und aufgepopptes Terminal-Fenster schließen
7. Serielle-Debug/ GPIO-Ausgabe des Programms auswerten und im Fehlerfall die Schritte 5 bis 7 wiederholen
8. uf2-Datei auf Zielsystem kopieren

Diese acht Schritte verwenden wir am Ende für ein „Hallo Welt“ Projekt, das uns die umfängliche Funktion des W-LAN Programmiergerät im Zusammenhang mit unserer „Geany-Kofiguration“ auf dem Entwicklungsrechner demonstriert. Und ihr danach eure Zeit dem
https://datasheets.raspberrypi.com/pico/getting-started-with-pico.pdf
widmen könnt um den Pico programmatisch mit Hilfe des
https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf

zu beherrschen. Zuvor noch zwei wichtige aber einfache Schritte:

1. Teilweise-skriptgeführte Konfiguration des Rpi-ZeroW **ca 60 min. Dauer**
2. Skriptgeführte Konfiguration des Dektop-Linux-Enwicklungsrechners

Konfiguration des RPiZeroW

Wichtig: Der RPiZeroW und der Entwicklungsrechner müssen ein Benutzerkonto /home-Verzeichnis gleichen Namens haben unter dem die Befehle auszuführen sind! Beispielhaft lauten diese hier:
Auf dem Entwicklungsrechner `meinName@meinDesktop` und
auf dem RPiZeroW `meinName@meinProgger`
Auch wichtig: Wir führen alle befehle im Stammverzeichnis „/home/meinName/“ aus

Wir sollen auf dem Dektop-Linux Entwicklungsrechner mit dem Benutzer `meinName@meinDesktop` eingelogt sein, und haben ein Terminalfenster geöffnet um einen SSH-Schlüssel zu erzeugen den wir zum RPiZeroW übertragen.
Wenn wir noch keinen „~/.ssh/id_rsa.pub“ Schlüssel haben, geben wir zuerst den Befehl zur Erzeugung des Schlüssels folgend ein:

```
ssh-keygen -t rsa
```

Darauf erscheinen, einfach durch drücken von Enter zu bestätigende Eingabeaufforderungen

Um nun durch Eingabe von:

```
ssh-copy-id -i ~/.ssh/id_rsa.pub meinName@meinProgger
```

vielleicht müssen wir hier erst „yes“ gefolgt von Enter eingeben um zur Passwort-Eingabe des Benutzer `meinName@meinProgger` zu gelangen um den Schlüssel zu übertragen.

Über SSH logen wir uns, bestenfalls in ein neu geflashtes RaspiOS >= Bullsey ohne Desktop und auto-login, mit dem dem Benutzer : `meinName@meinProgger` auf dem RpiZeroW durch folgenden Befehl ein:

```
ssh meinProgger
```

In der darauf erscheinenden Eingabeaufforderung des RpiZeroW geben wir den Befehl unten ein:

```
sudo raspi-config
```

So das wir im angezeigten Menu folgende Auswahl treffen:
In „`Interface-options->Serial Port“` wählen wir zuerst „`no“` keine Shell über die serielle Schnittstelle und aktivieren dann mit „`yes“` den „Hardwarezugriff auf die serielle Schnittstelle“, mit Auswahl von „`oK“` verlassen wir das Menu; Den Reboot führen wir später durch

Wir starten die zeitintensive Konfiguraion mit drei Befehlen:

```
wget https://raw.githubusercontent.com/momefilo/Entwicklungsumgebung/refs/heads/main/install_bridge.sh
```

```
chmod +x install_bridge.sh
```

```
sudo ./install_bridge.sh
```

Während der Ausführung des Skripts erscheinen sehr wenige blau- und lilafarbene Warnmeldungen die nicht stören, solang es keine roten Fehlermeldungen sind. Wir lassen das Terminal-Fenster einfach die Stunde offen und öffnen ein neues Terminal-Fenster auf dem Desktop-Linux um weiter zu machen.

Konfiguration des Dektop-Linux-Entwicklungsrechner

Die Konfiguration Erfordert nur drei Befehle, von denen der letzte ein das mit den beiden zuvor geladene und ausführbar gemachte Skript startet. Das script dauert ca. 10 min und wird mit dem Rechnernamen des RpiZeroW als Parameter aufgerufen, in diesem Beispiel lautet der „meinProgger“ und muss durch euren ersetzt werden.
Wir befinden uns im „/home/meinName“-Verzeichnis des Terminal-Fenster des Desktop-Linux und geben folgende drei Befehle nacheinander ein

```
wget https://raw.githubusercontent.com/momefilo/Entwicklungsumgebung/refs/heads/main/install_desktop.sh
```

```
chmod +x install_desktop.sh
```

```
./install_desktop.sh meinProgger
```

Nach Abschluss der Konfiguration schließen wir das Terminal-Fenster und öffnen ein Neues, das ist wichtig um die neuen Umgebungsvariablen zu laden damit wir mit unserem Ersten Programm beginnen können

„Hallo Welt“ Programm

Erweitertes „Hallo Welt“ Programm

Und nach dem erweiterten „acht Schritte“- Prozess?

Die Ausgangsleistung der GPIO’s und zusätzliche PullUp/Down Wiederstände bei nicht kurzen Busleitungen beachten! Ich habe schon viele Stunden mit Fehlersuche verbracht und nur durch Zufall den Verantwortlichen Wackelkontakt JumperWire/BreadBoard entdeckt, der Sicherheit immer wieder auftritt.