500 € Rastertunnelmikroskop

Dokumentation Elektronik und Programmierung

Armin Manfred Bräuer

Dokumentationsstand: 2. Dezember 2020

Universität Regensburg

Inhaltsverzeichnis

1 Elektronik

1.1 Bauteilauswahl

Bei der Auswahl der Bauteile wurde Wert auf weit verbreitete Hardware gelegt. Das hat den Vorteil, dass die Bauteilbeschaffung für den Nachbau des Projektes einfach erfolgen kann. Ebenfalls wurde speziell Hardware ausgewählt, die voraussichtlich lange erworben werden kann oder alternativ problemlos durch direkte Nachfolger ersetzt werden kann.

Tabelle 1: Bauteilauflistung

Bauteil	**Verwendung**	**Kommentar**
Raspberry Pi 4	Hauptprozessor, User Interface	empfohlen für grafische Oberfläche: 4GB Ram Version
microSD	Massenspeicher für Raspberry Pi	mindestens 8GB, empfohlen: 32GB
USB-C Netzteil	Stromversorgung für Raspberry Pi	mindestens 2,5A, empfohlen: Raspberry Pi offizielles Netzteil
ESP32	Hardwareprozessor	Developmentboard 30pin
microUSB Typ B Netzteil	Stromversorgung für ESP32	kein höherer Strom notwendig, empfohlen: ab 1A
mikroe ADC 8 Click	ADC für Regelkreis, Konvertierung der zum Tunnelstrom proportionalen Spannung	16bit ADC mit I2C Interface, Chip: ADS1115
mikroe DAC 2 Click	DAC für Ausgabe x, y und z	3x 16bit DAC mit SPI Interface, Chip: LTC2601

1.2 Verschaltung der Hardware

Die in Tabelle 1 aufgeführte Hardware wird über die Platine miteinander verbunden.
Dabei ist der Bus zwischen Raspberry Pi und ESP32 als SPI-Bus aufgeführt. Genauer gesagt wird einer der beiden SPI-Busse des ESP32, der **HSPI**, als Kommunikationsschnittstelle genutzt. Zusätzlich zu den Standardsignalen des SPI-Bus wird eine Handshakeverbindung gepflegt, die die Bereitschaft des Coprozessors vermittelt.

Der zweite Bus des ESP32 (**VSPI**)) wird genutzt zur Verbindung zu den DACs. Alle DACs (x, y, z) werden über einen gemeinsamen SPI-Bus, jedoch mit einem separaten Chipselect (CS) angebunden.

Des Weiteren wird der ADC per I2C mit dem ESP32 verschaltet.

1.3 Verbindungsplatine

Die Verbindungsplatine stellt die in Unterabschnitt 1.2 erklärten Verbindungen her.

Diese Platine wurde in Autodesk Eagle designt. Sie bietet vier Mikroe-Sockel, die allerdings aufgrund der CS-Pins nicht frei gewählt werden können. Es ist anzumerken, dass der CS-Port des Sockels 4 nicht mit einem Pin des ESP verbunden ist, weshalb dieser Sockel nur für I2C Geräte (z.B. der verwendete ADC) genutzt werden kann. Die Orientierung der Sockel kann sowohl über die Beschriftung mit dem Modul vergleichen werden als auch über die **TOP**-Markierung.

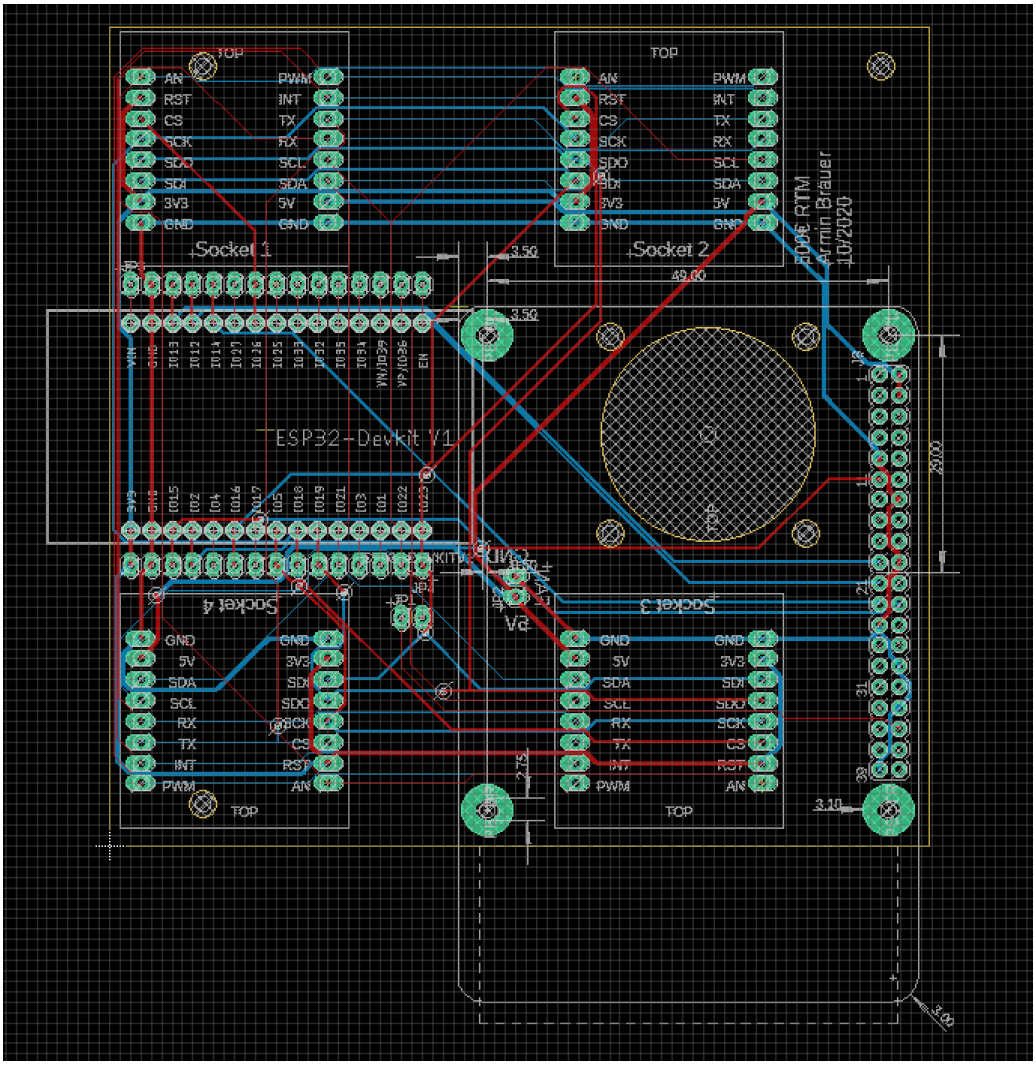

Abbildung 1: Layout der Verbindungsplatine

TOP bezeichnet hier die Oberseite der Module, die meistens durch Schraubanschlüsse identifiziert werden kann. Weiter wird der Raspberry Pi auf die Unterseite gesteckt, der ESP32 wird auf der Oberseite platziert. Zudem ist zur Kühlung des Raspberry Pi eine Aussparung für einen 40x40mm Lüfter mit Anschlüssen auf der Platine vorgesehen. Diese sind direkt mit den 5V- und GND-Pins des Raspberry Pi verbunden. Zusätzlich ist ein Jumper für die Resetansteuerung des ESP durch den Raspberry Pi vorgesehen, die jedoch im aktuellen Stand softwareseitig implementiert ist. Wird diese genutzt, so muss nur an Stelle des *JP1* ein 1x2 Pinheader eingelötet werden, der dann per Jumper geschlossen wird.

2 Programmierung

Nachfolgend wird die Struktur der Programmierung des Raspberry Pi und des ESP32 erläutert. Zudem wird das Protokoll zwischen diesen Geräten thematisiert. Beide Systeme sind in C/C++ programmiert, wodurch der Code relativ leicht auch von Schülern zu verstehen ist, sollten diese Erfahrung in diesen Sprachen haben. An sinnvollen Stellen wurde auf die objektorientierte

Programmierung zurückgegriffen.

2.1 Abstrahierte Struktur

Zu Beginn eines Scanvorgangs wird auf dem Raspberry Pi per Kommandozeile ein Scan initialisiert. Dabei werden die Parameter als Argumente übergeben.

```
1         kI = atof(argv[1]);
2         kP = atof(argv[2]);
3         destinationTunnelCurrentnA = atof(argv[3]);
4         remainingTunnelCurrentDifferencenA = atof(argv[4]);
5         startX = (uint16_t) atoi(argv[5]);
6         startY = (uint16_t) atoi(argv[6]);
7         direction = (bool) atoi(argv[7]);
8         maxX = (uint16_t) atoi(argv[8]);
9         maxY = (uint16_t) atoi(argv[9]);
10        mutiplicatorGridAdc = (uint16_t) atoi(argv[10]);
```

Abbildung 2: Position der Parameter für Kommandozeilenaufruf seitens Raspberry Pi

In Abbildung 2 ist ein Ausschnitt aus dem Code aufgeführt, in dem die Zuweisung der Eingangsparameter zu den eigentlichen Variablen geschieht. In eckigen Klammern zu argv sind die Indizes angegeben, die der Reihenfolge entsprechen. Parameter 0 übergibt den Namen des Executables und wird deshalb nicht verwendet.

Tabelle 2: Parameterbedeutung und akzeptierte Werte

Name	**Bedeutung**	**akzeptierte Werte z.B.**
kI	Z-Reglerparameter kI	123.4
kP	Z-Reglerparameter kP	1234.5
destinationTunnelCurrentnA	Zielwert für Z-Regler, „Auf welchen Tunnelstrom soll geregelt werden “	12.34
remainingTunnelCurrentDifferencenA	Differenz des Tunnelstroms zum Zielwert, ab dem der Strom als ausgeregelt gilt	0.01
startX	Startwert X, normal immer 0	0
startY	Startwert Y, normal immer 0	0
direction	Startrichtung des Scans	0: rechts, 1: links
maxX	Maximalwert für X, für z.B. 200 Punkte: 199	199
maxY	Maximalwert für X, für z.B. 200 Punkte: 199	199
multiplicatorGridAdc	Multiplikator, wie viele DAC-Punkte pro Punkt nötig sind	10

Nach Initialisierung des Handshake-GPIOs als Input mit Pullup und Interruptroutine wird der SPI-Bus seitens des Raspberry Pi konfiguriert und in Betrieb genommen. Der Controller greift dazu auf den **BCM2835**-Chipsatz zurück, der eine Hardwareeinheit für SPI bereitstellt. Mit diesem Schritt ist die Hauptroutine main() des Raspberry Pis beendet. Weitere Schritte folgen nur, sobald der ESP die Handshake-Leitung mit einem Signal belegt. Dieses Prinzip folgt dem **Observerpattern** aus der modernen Programmiermethodik. Nur bei Interesse des Coprozessors wird eine Verbindung und Kommunikation hergestellt. Trotz der Tatsache, dass der Raspberry Pi der Busmaster ist, spielt der ESP32 eine höhere Rolle und nur er kann die Kommunikationen anregen. Dies liegt darin begründet, dass auf dem Raspberry Pi keine zeitkritischen Tätigkeiten ausgeführt werden. So kann dieser unterbrochen werden, was umgekehrt nicht möglich ist.

Nach diesen Schritten sollte ein Reboot des ESP32 erfolgen. Zum momentanen Zeitpunkt erfolgt dies noch über den physischen Button auf dem Gerät selbst, das kann jedoch durch

die, auf der Platine integrierten (näheres siehe Unterabschnitt 1.3), Resetline ersetzt werden, wodurch der Raspberry Pi die Möglichkeit erhält einen Reset am Coprozessor durchzuführen.

Wird der ESP32 gestartet, so wird lediglich ein Init der HSPI-Einheit (siehe Unterabschnitt 1.2) durchgeführt. Dabei werden sämtliche Pins mit ihren Funktionen konfiguriert, SPI-Objekte innerhalb des Frameworks initialisiert und Speicherbereiche für das Senden und Empfangen von Paketen eingerichtet. Anschließend wird ein Task zur Bedienung des Busses erstellt und gestartet. Dieser führt eine Anfrage auf Übertragung der Konfiguration über den HSPI zum Raspberry Pi durch, woraufhin einzeln die Parameter übertragen werden. Sollte eine Übertragung fehlerhaft sein, so wird weiterhin die Anfrage aufrecht erhalten, bis die Übertragung vollständig abgeschlossen wurde. Sollte der ESP32 nie in den Modus des eigentlichen Messens wechseln, so liegt die Vermutung nahe, dass die Kommunikation bezüglich der Parameter nicht erfolgreich abgeschlossen wurde. Debugausgaben per serieller Konsole liefern nähere Informationen zum aktuellen Stand der Übertragung.
Der HSPI-Task ist weiter auch für die Übertagung der gemessenen Daten als ein größeres Paket eingerichtet. Darauf wird jedoch im späteren Verlauf dieses Usecases eingegangen.
Ist die Übertragung abgeschlossen, so werden die Parameter in der seriellen Konsole ausgegeben und der eigentliche Regelalgorithmus wird gestartet. Dieser Vorgang enthält die Initialisierung des VSPI-Busses zu den DACs. Das erfolgt nahezu identisch zur Einrichtung des HSPI. Weiter wird ein Timer mit der Frequenz des ADCs initialisiert. Die Eingabe der Zeit erfolgt in μs, weshalb sich die Zeit mit $\frac{10^6 \mu s}{860 SPS} = 1162 \mu s$ berechnet. Die Einheit SPS steht hierbei für SamplesPerSecond. Um leicht unterhalb der Frequenz des ADCs zu arbeiten wird eine Zeit von 1200 μs gewählt.
Die Prozesse der Regelung (inklusive ADC) und die Kommunikation mit den DACs laufen, trotz der potentiellen Möglichkeite dazu, nicht parallel zueinander ab. Das ist einerseits dem langsamen ADC geschuldet, der die Zykluszeit auf 1162 μs anhebt, obwohl die reine Rechenzeit auf dem Controller mit 300 μs wesentlich kürzer ausfällt. Andererseits sind bei Tests Probleme aufgetreten, da neue Regelzyklen noch nicht mit dem aktuellen Ist-Wert der Regelgröße gearbeitet haben, weil die DAC Kommunikation noch nicht abgeschlossen war, jedoch der Sample-and-Hold Vorgang im ADC schon gestartet wurde.
Liegt nach einigen Regelzyklen der Tunnelstrom im angestrebten Bereich, so wird an die **dataQueue** ein Eintrag mit den x- und y-Koordinaten und dem DAC-Wert der z-Richtung angehängt. Erreicht die Anzahl der gespeicherten Datensätze den konfigurierten Maximalwert, so werden Timer und Regler angehalten und ein Senden der angereihten Datensätze an den Raspberry Pi initiiert. Das Gleiche passiert auch, wenn der letzte Punkt aufgenommen wurde. Zwischen den Scanpunkten wird pro y-Reihe immer mit abwechselnder Richtung die Position verschoben. Das bedeutet, dass es keine größeren Verfahrwege gibt, sondern nur Schritte von einem Pixelabstand verfahren werden. Sind alle Punkte aufgenommen, so werden alle Tasks gelöscht. Die Daten befinden sich nun auf dem Raspberry Pi in einem .csv-File. Für einen neuen Scan muss ein neuer Aufruf seitens des Raspberry Pi getätigt werden.

2.2 Besonderheiten Raspberry Pi

Der Raspberry Pi wird mit dem Standardbetriebssystem Raspbian (Konsole oder mit GUI) betrieben. Dieses bringt bereits viele Programme mit, die für den Buildprozess oder auch das Ausführen des Codes notwendig sind. Lediglich **PiGPIO** und eventuell **make** sind zusätzlich

nötig. Es ist anzumerken, dass bei Möglichkeit der Code für das RTM als root ausgeführt werden sollte (`sudo -i`), da sonst Probleme beim Ansprechen der Hardwareeinheiten nicht ausgeschlossen werden können.

2.3 Betriebssystem ESP32

Der ESP32 wird in diesem Projekt mit der SDK **esp-idf**, die auf dem Betriebssystem **FreeRTOS** basiert, betrieben. Das hat den Vorteil, dass von beiden Kernen des Prozessors Gebrauch gemacht werden kann. Somit können Prozesse parallel zueinander ablaufen. Außerdem bietet die genutzte SDK den Vorteil, dass sie aufgrund ihrer Verbreitung (und der Verbreitung des ESP32) noch längere Zeit unterstützt und gewartet wird.

2.4 Codestruktur ESP32

3 Anleitung: Aufbau der Hardware und Flashen der Software

3.1 Installation und Einrichtung Raspberry Pi

Das Betriebssystem des Raspberry Pi wird mithilfe der Software BalenaEtcher (oder vergleichbaren Programmen) auf die microSD Karte geschrieben. Sobald dieser Prozess beendet ist und der Pi mit dieser Karte startet wird das Dateisystem ausgeweitet, damit der gesamte Platz der Speicherkarte zur Verfügung steht. Anschließend werden die fehlenden Pakete mit den Befehlen `apt update` und `apt install pigpio make` installiert. Des Weiteren muss der Quellcode auf den Raspberry Pi geladen werden. Je nach Situation stehen hier SCP oder git zur Verfügung. Zu beiden Punkten finden sich im Internet zahlreiche Anleitungen zur Anwendung. Mit dem Befehl `make` wird der Inhalt des Ordners zu einer ausführbaren Datei gebaut (siehe Abbildung 3). Nach diesem Schritt ist bereits die Einrichtung abgeschlossen und das Programm kann über einen Aufruf wie in **??** durchgeführt werden.

```
root@rtm-pi:~/git/500-rtm-pi/500-rtm-pi# ls
CMakeCache.txt  cmake_install.cmake  include  log.txt   Makefile     platformio.ini  test
CMakeFiles      CMakeLists.txt       lib      log.txtc  newScan.csv  src
root@rtm-pi:~/git/500-rtm-pi/500-rtm-pi# make
[ 16%] Building CXX object CMakeFiles/rtm_pi.dir/src/communication.cpp.o
[ 33%] Building CXX object CMakeFiles/rtm_pi.dir/src/dataStoring.cpp.o
[ 50%] Building CXX object CMakeFiles/rtm_pi.dir/src/globalVariables.cpp.o
[ 66%] Building CXX object CMakeFiles/rtm_pi.dir/src/main.cpp.o
[ 83%] Building CXX object CMakeFiles/rtm_pi.dir/src/spi.cpp.o
[100%] Linking CXX executable rtm_pi
[100%] Built target rtm_pi
root@rtm-pi:~/git/500-rtm-pi/500-rtm-pi# ls
CMakeCache.txt  cmake_install.cmake  include  log.txt   Makefile     platformio.ini  src
CMakeFiles      CMakeLists.txt       lib      log.txtc  newScan.csv  rtm_pi          test
root@rtm-pi:~/git/500-rtm-pi/500-rtm-pi#
```

Abbildung 3: Buildprozess des Projektes mit Ordnerinhalt vorher und nachher

```
root@rtm-pi:~/git/500-rtm-pi/500-rtm-pi# ./rtm_pi 100 1000 10.0 0.01 0 0 0 199 199 100
Argument 1: 100
Argument 2: 1000
Argument 3: 10.0
Argument 4: 0.01
Argument 5: 0
Argument 6: 0
Argument 7: 0
Argument 8: 199
Argument 9: 199
Argument 10: 100
Correct count of parameters
SPI setup succesfull
```

Abbildung 4: Aufruf des Skripts seitens Raspberry Pi

3.2 Einrichtung ESP32

Die Einrichtung des ESP32 ist aufgrund seiner generellen Struktur etwas schwieriger, wird jedoch durch Entwicklungsumgebungen und deren Erweiterungen so weit wie möglich vereinfacht.

Zum Hochladen des Codes ist hauptsächlich der Editor **Visual Studio Code** erforderlich. Dieser kann kostenlos für alle Betriebssysteme von Microsoft bezogen werden. Zudem muss im installierten VS Code die Extension **PlatformIO** installiert werden. Diese Erweiterung stellt eine Embedded-Entwicklungsumgebung bereit. In PlatformIO muss weiter noch eine Plattform zur Entwicklung von Code für ESP Produkte eingebunden werden, Abbildung 5 dient zur Orientierung.

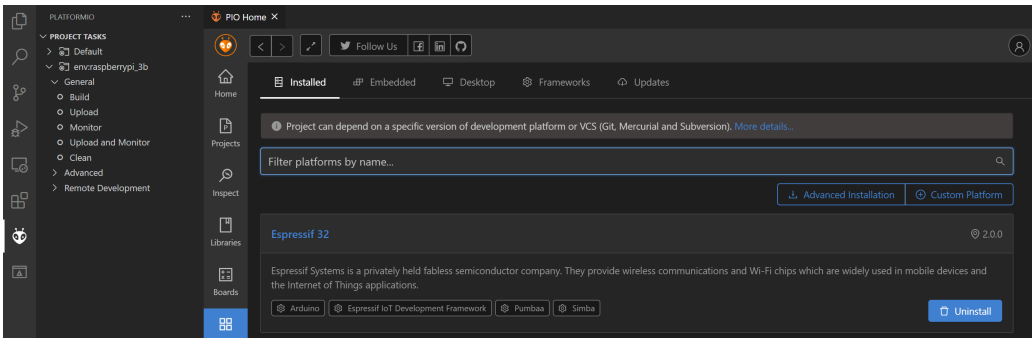

Abbildung 5: Installation der Espressif 32 Plattform zur Entwicklung von ESP-IDF

Nach der Installation der Erweiterung und der Plattform kann das eigentlich Projekt über File -> Open Folder geöffnet werden. In der Datei platformio.ini befindet sich die Konfiguration bezüglich des COM-Ports (Windows) oder des /dev - Mounts (Unix). Der Uploadport muss hier an das entsprechende System angepasst werden. Windowsnutzer finden den Port im Geräte-Manager. Anschließend wird der Code per horizontalem Pfeil in der unteren blauen Leiste auf den Microcontroller geladen. Die Konsolenausgaben werden über das Steckersymbol erreicht.

3.3 Hardwareaufbau

Der Aufbau der Hardware gestaltet sich aufgrund der Platine einfach. Hierbei ist ausschließlich darauf zu achten, dass die Belegung der Stecksockel richtig erfolgt.

Tabelle 3: Stecksockelbelegung

Sockelnummer	**Modul**
1	DAC-X
2	DAC-Y
3	DAC-Z
4	ADC

4 Anhang

Abbildungsverzeichnis