Armin Rohnen

STM32F411 nucleo - MATLAB® Schnittstelle

Schnittstellenbeschreibung für die ersten Schritte der Microcontrollerprogrammierung

November 27, 2021

Hochschule München
Fakultät Maschinenbau, Fahrzeugtechnik, Flugzeugtechnik
Dachauerstraße 98b
80335 München

Vorwort

Dieses Dokument beschreibt die Softwareschnittstelle zwischen dem STM32F411 nucleo Board und MATLAB® für die Steuerungsaufgaben im Pumpenprüfstand und in der labortechnsichen Espressomaschine.

Es wird davon ausgegangen, dass der Leser über Grundkenntnisse in der Programmierung mit MATLAB® und Python verfügt.

Im Softwarestand zum 1. November 2021 verfügt die beschriebene Schnittstelle über Funktionalität zum Ansteuern der Magnetventile, zur Ausgabe von PWM-Signalen sowie zur Ausgabe von zwei Spannungssignale welche als Sollwerte im Spannungsbereich 0 bis 10 Volt verwendet werden. Ebenso wurde die Füllstandserkennung mit zwei Füllstandssensoren sowie die Spannungsmessung aller analogen Eingänge realisiert.

Nicht umgesetzt wurde die Funktionalität zum Ansteuern eines DC-Motortreibers. Hier ist es im aktuellen Projektstand fraglich ob diese Funktionalität zukünftig benötigt wird. Die Entscheidung darüber wird im Zuge der Funktionstest der Getriebepumpe gefällt. Ebenfalls steht derzeit die Ansteuerung eines Displays sowie die Erfassung des Flowmeter-Signals nicht zur Verfügung. Dies wird in späteren Versionen der Schnittstelle realisiert werden.

Die Literaturverweise in diesem Dokuemnt beziehen sich auf die Quellenliste, welche unter $https://wiki.turnus-espresso.de/Quellen\_und\_Dokumente$ zu finden ist.

Planegg, November 2021 *Armin Rohnen*

Inhaltsverzeichnis

Abkürzungsverzeichniss

CR/LF carriage return - line feed, Wagenrücklauf und Zeilenvorschub, Sonderzeichen für das Ende einer Datenzeile

ADC Digital Analog Converter, Digital zu Analog Converter

GPIO general-purpose input/output, allgemeiner bzw. digitaler Ein- / Ausgang

IDE Integrated Development Enviroment, integrierte Entwicklungsumgebung

I2C Inter-Intergrated Circuit, I-Qaudrat-C, serieller Datenbus zur Kommunikation in elektronischen Schaltungen

LED light-emitting diode, Leuchtdiode

MCU Micro Controller Unit, Mikrocontroller

PIN Mikrocontroller-PIN, Elektrischer Anschlusskontakt des Mikrocontroller-Bausteins

PWM Puls Weiten Modulation, ein Rechtecksignal mit definierbarer Taktfrequenz und einstellbarer Teilung, die Information wird hierbei über die Teilung (Tastverhältnis) übermittelt

REPL read-eval-print-loop, lese-auswerte-rückgabe-schleife

SDA Serial Data, I2C Datenleitung

SCL Serial Clock, I2C Taktleitung

UART Universal Asynchronous Receiver Transmitter, Universelle asynchrone Schnittstelle, Digitale serielle Schnittstelle für den Datenaustausch zwischen

Mikrocontrollern und PCs

USB USB-Schnittstelle, Universial Serial Bus, Universelle serielle PC Schnittstelle

Kapitel 1
Schnittstellenbeschreibung

Die Schnittstellenbeschreibung umfasst jenen Programmcode in MicroPython, welcher entweder direkt über eine Entwicklungsumgebung auf dem STM32F411 Board ausgeführt oder via MATLAB® zur Ausführung gebracht wird. Auf eine Darstellung als MATLAB®-Code wird dabei verzichtet, da dies auf Basis der MATLAB®-MicroPython Kommunikation als selbsterklärend betrachtet wird. Eine ausführliche Beschreibung zur MATLAB®-MicroPython Kommunikation sowie zur Erstellung von MATLAB® Apps wird auf [40] verwiesen.

1.1 MATLAB®-MikroPython Kommunikation

MATLAB® verfügt seit der Version 2019b über das Objekt *serialport* über das auf den USB-Serial-Treiber zugegriffen werden kann. Die Anweisung

$$\text{mcu} \;=\; \text{serialport}('\text{COM3}', 115200);$$

stellt die Verbindung von MATLAB® zum Mikrocontroller mit einer Baudrate von 115200 her. Die weiteren EInstellungen 8 Datenbits, keine Parität und ein Stopbit entsprechen der Voreinstellung serieller Schnittstellen und müssen daher nicht weiter eingestellt werden. Zuvor ist die USB-Anschlussbezeichnung des Mikrocontroller zu ermitteln. Dies erfolgt am leichtesten über die Systemsteuerung des PCs. Die Bezeichner dort sind mit den Bezeichner in MATLAB® identisch.

Über die beiden Anweisungen

```
configureTerminator(mcu, 'CR/LF');
configureCallback(mcu, 'terminator', @Datenverarbeitung);
```

ist die Kommunikation zwischen MATLAB® und dem Mikrocontroller vollständig konfiguriert. Jede Eingabe- bzw. Datenzeile wird mit den Sonderzeichen ”CR” für carriage return (Wagenrücklauf) und ”LF” für line feed (Zeilenvorschub) beendet. Die Anweisung *configureTerminator* konfiguriert den Serialport entsprechend.

Für die am PC eingehenden Daten, stellt der virtuelle USB-Serial-Treiber einen Eingangsdatenpuffer bereit. Dieser wird durch die Anweisung *configureCallback* so konfiguriert, dass immer dann, wenn dort eine Datenzeile eingegangen ist, die Funktion *Datenverarbeitung* aufgerufen wird. Die Datenverarbeitung selbst erfolgt In der Funktion. Im einfachsten Falle wird die Datenzeile im Command Window von MATLAB® ausgegeben.

Mit

```
function datenverarbeitung(device,~)
  zeile = readline(device)
end
```

erfolgt dies.

Nachdem die Konfiguration der seriellen Schnittstelle abgeschlossen ist, kann über das Command Window von MATLAB® direkt mit dem Mikrocontroller kommuniziert werden.

1.2 Benötigte Module - Importanweisungen

Für die Initialisierung des Basisboards werden einige Module aus der MicroPython-Bibliothek sowie weitere Classendefinitionen benötigt. Der Import ist exakt in der dargestellten Form erforderlich, da anderenfalls im weiteren Programmcode eine andere Syntax verwendet werden muss.

```
from machine import SoftI2C
from machine import Pin
import mcp23017
from pyb import ADC
from pyb import Timer
import mcp4725
importutime
```

Das Modul *SoftI2C* stellt die Funktionalität der Elektronikschnittstelle *I2C* zur Verfügung. Nur hierdurch können der GPIO-Expander *MCP*23017 und die beiden DACs *MCP*4725 angesprochen werden. Zur Realisierung der Funktionalität der beiden Bauelemente werden die entsprechenden Klassenbibliothelken *mcp*23017 und *mcp*4725 benötigt. Das Modul *Pin* ermöglicht den logischen Zugriff auf PINs des Microcontrollers. Die Module *ADC* und *Timer* werden für die Spannungsmessungen und die PWM-Funktionen benötigt.

1.3 Callback Funktionen für die Interruptbehandlung

Die Impulse der angeschlossenen Flowmeter lösen an dem jeweiligen PIN der MCU einen Interrupt aus. Für die Abarbeitung dieser Interrupts werden die Callback-Funktionen angelegt. Es wird jeweils ein Zeitstempel in Microsekundenauflösung erzeugt.

```
def flow1_callback(Pin) :
  print('flow1 :' +str(utime.ticks_us()))
```

```
def flow2_callback(Pin) :
  print('flow2 :' +str(utime.ticks_us()))
```

1.4 GPIO-Expander - Ansteuerung der Magnetventile

Die Magnetventile in den angeschlossenen Maschinen werden über den GPIO-Expander *MCP*23017 angesteuert. Logisch 1 schaltet des Magnetventil, während Logisch 0 das Magnetventil im Ruhezustand hält. Der *MC*23017 Baustein muss dazu konfiguriert werden. Die erfolgt in drei Schritten. Zunächst wird die zugehörige I2C-Schnittstelle über

$$\text{i2c\_exp} = \text{SoftI2C}(\text{scl} = \text{Pin}('\text{PB6}'), \text{sda} = \text{Pin}('\text{PB7}'), \text{freq} = 400000)$$

definiert und im zweiten Schritt der GPIO-Expander über

$$\begin{aligned}&\text{gpio\_exp} = \text{mcp23017.MCP23017}(\text{i2c\_exp}, 0\text{x}20)\\&\text{gpio\_exp.porta.mode} = 0\text{b}00000000\\&\text{gpio\_exp.portb.mode} = 0\text{b}00000000\end{aligned}$$

initialisiert. Während

$$\begin{aligned}&\text{gpio\_exp.porta.gpio} = 0\text{b}00000000\\&\text{gpio\_exp.portb.gpio} = 0\text{b}00000000\end{aligned}$$

alle PINs des GPIO-Expanders auf 0 setzt. Jeder Ausgang des GPIO-Expanders kann entsprechend er Tabelle 1.1 separat gesetzt werden.

1.5 Analoge Spannungseingänge - Analoge Messwerte

Die analogen Messwerte werden als Spannungen an den jeweilig definierten PINs des Microcontrollers über die internen 12-Bit-ADC bestimmt. Hierzu müssen die entsprechenden PINs als ADC wie folgt angelegt werden:

Tabelle 1.1 Anweisungen zum Schalten der Magnetventile

PIN Bezeichnung	Anweisung
D01	gpio_exp.pin(1, value = 1)
D02	gpio_exp.pin(0, value = 1)
D03	gpio_exp.pin(8, value = 1)
D04	gpio_exp.pin(9, value = 1)
D05	gpio_exp.pin(10, value = 1)
D06	gpio_exp.pin(11, value = 1)
D07	gpio_exp.pin(12, value = 1)
D08	gpio_exp.pin(13, value = 1)
D09	gpio_exp.pin(14, value = 1)
D10	gpio_exp.pin(15, value = 1)

$$\text{T\_Boiler} = \text{ADC}('\text{PA0}')$$
$$\text{T\_Befuellung} = \text{ADC}('\text{PA1}')$$
$$\text{T\_Eingang} = \text{ADC}('\text{PA4}')$$
$$\text{Leitwert} = \text{ADC}('\text{PC2}')$$
$$\text{P\_Gruppe} = \text{ADC}('\text{PC1}')$$
$$\text{Taste} = \text{ADC}('\text{PC3}')$$
$$\text{P\_Boiler} = \text{ADC}('\text{PC0}')$$
$$\text{T\_Zwischenraum} = \text{ADC}('\text{PA5}')$$
$$\text{Gewicht1} = \text{ADC}('\text{PA6}')$$
$$\text{T\_Mischer} = \text{ADC}('\text{PA7}')$$
$$\text{P\_Boiler\_Alt} = \text{ADC}('\text{PB1}')$$
$$\text{Gewicht2} = \text{ADC}('\text{PC2}')$$

Die Zuordnung der analogen Messwerte erfolgt über die Tabelle 1.2.

Für die Durchführung der Spannungsmessungen wurden drei Anweisungen vorbelegt:

1. Temperaturmesswerte
   $print(str(T\_Boiler.read()*3.3/4096)+',' +str(T\_Befuellung.read()*3.3/4096)+',' +str(T\_Eingang.read()*3.3/4096)+',' +str(T\_Zwischenraum.read()*3.3/4096)+',' +str(T\_Mischer.read()*3.3/4096))$
2. Drücke
   $print(str(P\_Gruppe.read()*3.3/4096)+',' +str(P\_Boiler.read()*3.3/4096)+',' +str(P\_Boiler\_Alt.read()*3.3/4096))$
3. Restliche Messwerte
   $print(str(Leitwert.read()*3.3/4096)+',' +str(Taste.read()*3.3/4096)+',' +str(Gewicht1.read()*3.3/4096)+',' +str(Gewicht2.read()*3.3/4096))$

Tabelle 1.2 Zuordnung der analogen Messwerte

Bezeichnung	Messwert
T_Boiler	NTC Temperatur im Boiler
T_Befuellung	NTC Temperatur in der Boilerbefüllung
T_Eingang	Temperatur am Leitwertsensor
Leitwert	Leitwert
P_Gruppe	Druck in der Brühgruppe
Taste	Tastendruck, die gedrückte Taste wird über den Spannungsmesswert identifiziert
P_Boiler	Boilerdruck
T_Zwischenraum	NTC Temperatur im Zwischenraum der Glaszylinder
*Gewicht*1	Waegezelle 1
T_Mischer	Mischwassertemperatur
P_Boiler_Alt	Boilerdruck über AVS-Römer Drucksensor
*Gewicht*2	Waegezelle 2

1.6 Digital Input - Füllstände erkennen

Für die Erkennung von Füllständen ist das Basisboard mit zwei Schaltungen versehen, welche eine Kurzschlussdetektion vornehmen und den Zustand als HIGH-Signal (Kurzschluss) oder LOW-Signal (kein Kurzschluss) an zwei PINS des Microcontrollers anlegen. Über

$$\text{Fuell\_1} = \text{Pin}('\text{PB15}', \text{Pin.IN}, \text{Pin.PULL\_UP})$$
$$\text{Fuell\_2} = \text{Pin}('\text{PC8}', \text{Pin.IN}, \text{Pin.PULL\_UP})$$

wird die Funktionalität der PINs definiert. Die Erkennung erfolgt durch die Abfrageanweisungen

$$\text{Fuell\_1.value}()$$
$$\text{Fuell\_2.value}()$$

1.7 Digital Input - Flowmeter

Das Basisboard kann über digitale Eingänge die Impulssignale eines Flowmeters detektieren. Ein fallende Flnake am Digitaleingang löst einen Interrupt aus.

$$\text{FLOW1} = \text{Pin}('\text{PA12}', \text{Pin.IN})$$
$$\text{FLOW1.irq}(\text{trigger} = \text{Pin.IRQ}_\text{F}\text{ALLING}, \text{handler} = \text{flow1}_\text{c}\text{allback})$$
$$\text{FLOW2} = \text{Pin}('\text{PB5}', \text{Pin.IN})$$
$$\text{FLOW2.irq}(\text{trigger} = \text{Pin.IRQ}_\text{F}\text{ALLING}, \text{handler} = \text{flow2}_\text{c}\text{allback})$$

1.8 0 bis 10 V Steuersignale - Spannungen erzeugen

Für das AVS-Römer Dosierventiel und den Appoldt Phasenschnittregler wird als Stellsignal ein Spannungssignal zwischen 0 und 10 Volt benötigt. Der Microcontroller selbst verfügt nicht über die Möglichkeit Spannungen zu generieren. Dies erfolgt über zwei 12-Bit DAC-Bausteine *MCP*4725. Analog zum GPIO-Expander wird hierzu eine Bibliothek verwendet. Die Bausteine sind über einen weiteren I2C mit dem Microcontroller verbunden.

Die Anweisung

$$\text{i2c\_dac} = \text{SoftI2C}(\text{scl} = \text{Pin}('\text{PB10}'), \text{sda} = \text{Pin}('\text{PB9}'), \text{freq} = 400000)$$

konfiguriert den zugehörigen I2C-Bus. Über

$$\text{dosierventil} = \text{mcp4725.MCP4725}(\text{i2c\_dac}, \text{mcp4725.BUS\_ADDRESS}[1])$$
$$\text{appoldt} = \text{mcp4725.MCP4725}(\text{i2c\_dac}, \text{mcp4725.BUS\_ADDRESS}[0])$$

werden die beiden DACs angelegt. Über die direkt im Anschluss erfolgenden Anweisungen

$$\text{appoldt.write}(0)$$
$$\text{dosierventil.write}(0)$$

wird sichergestellt, dass die jeweilige Spannungsausgabe mit 0 Volt eingestellt ist und kein Fehlverhalten aufkommen kann.

Die Spannungswerte der beiden DACs werden analog der obigen Anweisung für 0 Volt durchgeführt. Der Wertebereich beträgt dabei 0 bis 4095 für den Spannungs-

bereich 0 Volt bis 10 Volt. Die Bestimmung des Wertes erfolgt gemäß

$$Wert = \frac{4095}{10000} \cdot U$$

Es muss ein ganzzahliger Wert übergeben werden.

1.9 Gerätesteuerungen - PWM Signale

Einige Stellelemente, wie z. B. Servomotoren, benötigen PWM Signale zur Positionierung des Stellhebels. Die Leistungsregelung der Heizelemente wird ebenfalls über PWM-Signale realisiert.

Die Erzeugung eines PWM-Signals benötigt bei STM32 Microcontroller einen internen Timer. Je internen Timer stehen bis zu vier Kanäle zur Verfügung, welche alle im gleichen Takt betrieben werden. Allerdings kann nicht beliebig ein Timer mit einem PIN verknüpft werden. Tabelle 1.3 gibt Auskunft über die Zuordnung von PWM-Signal zu Timer und Timerkanal.

Tabelle 1.3 Zuordnung der Timer zu PWM-Signal

Bezeichnung	PIN	Gerät	Frequenz	Timer	Kanal
PWM Dampfventil	PB8	Servo	50 Hz	4	3
PWM Entnahmerohr	PC6	Servo	50 Hz	3	1
PWM Licht	PC7	LED	50 Hz	3	2
PWM Dosierventil	PC9	Servo	50 Hz	3	4
PWM Tassenwaermer	PA9	SSR	8 Hz	1	2
PWM Boiler	PA11	SSR	8 Hz	1	4

Es werden drei Timer mit den zugehörigen Timerfrequenzen über die Anweisungen

$$\begin{aligned} \text{timer1} &= \text{Timer}(1, \text{freq} = 8) \\ \text{timer3} &= \text{Timer}(3, \text{freq} = 50) \\ \text{timer4} &= \text{Timer}(4, \text{freq} = 50) \end{aligned}$$

angelegt. Im weiteren erfolgt über die Anweisungspaare $timerX.channel$ und $PWM.pulse\_width\_percent(0)$ die Initialisierung des jeweiligen PWM-Signals und die Voreinstellung mit 0% Impulsbreite, also kein Signal.

PWM_Dampf = timer4.channel(4, mode = Timer.PWM, pin = Pin('PB8'))
PWM_Dampf.pulse_width_percent(0)
PWM_Entnahme = timer3.channel(1, mode = Timer.PWM, pin = Pin('PC6'))
PWM_Entnahme.pulse_width_percent(0)
PWM_Licht = timer3.channel(2, mode = Timer.PWM, pin = Pin('PC7'))
PWM_Licht.pulse_width_percent(0)
PWM_Dosierventil = timer3.channel(4, mode = Timer.PWM, pin = Pin('PC9'))
PWM_Dosierventil.pulse_width_percent(0)
PWM_Tassen = timer1.channel(2, mode = Timer.PWM, pin = Pin('PA9'))
PWM_Tassen.pulse_width_percent(0)
PWM_Boiler = timer1.channel(4, mode = Timer.PWM, pin = Pin('PA11'))
PWM_Boiler.pulse_width_percent(0)

Die benötigte Impulsbreite des jeweiligen PWM-Signals wird z. B. durch die Anweisung $PWM\_Boiler.pulse\_width_percent(wert)$. Wobei *wert* einen Betrag zwischen 0 und 100 annehmen kann.