universität wien

Fakultät für Physik
Elektronik-Praktikum

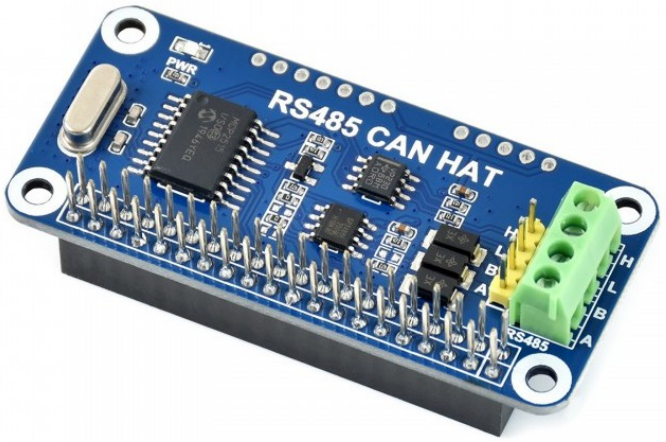
Bussysteme

RS485 / CAN Platine

Wir verwenden folgende Platine:

https://www.waveshare.com/rs485-can-hat.htm

Die Anschlüsse A und B sind für den RS485 Bus, H und L für CAN vorgesehen
Alle Teilnehmer sind bereits über CAN verbunden



RS485

Ein Temperatur / Luftfeuchtesensor soll über die RS485 Schnittstelle abgefragt werden.
Verwende dazu das Programm im Anhang.

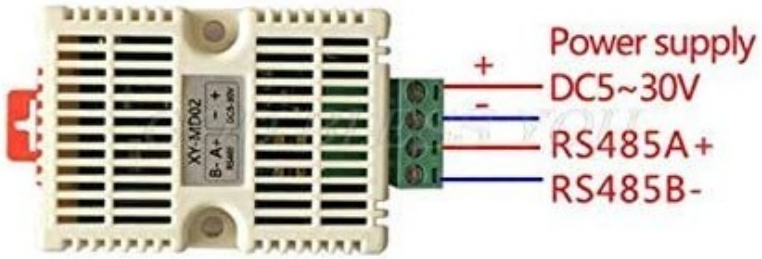
Aufbau 1Pkt
Programm 1Pkt

Das Modul XY-MD02 ist mit einem SHT20 Sensor bestückt.

Die Spannungsversorgung (5V, GND) erfolgt hier vom Raspi. Bei Platzierung des Sensors in größerer Distanz kann eine externe Spannungsversorgung notwendig sein (Leitungslänge bei RS485 bis 1000m)

Wir verwenden die Bibliothek ‚minimalmodbus' für den Datentransfer

https://minimalmodbus.readthedocs.io/en/stable/



RS485 Programm für XY-MD02 Temperatur

Übertragung bei read_register(register, Dezimalstellen, Function-code)

Lese die Temperatur aus Register1 mit 1 Kommastelle, Function-Code=4
temperature=instrument.read_register(1,1,4) # Register mit der Temperatur einlesen

Aus Datenblatt:

ModBus Command							
Master Read Temperature Command Frame(0x04)							
Device Address	Function Code	Starting Address Hi	Starting Address Li	Quantity Hi	Quantity Li	CRC Hi	CRC Li
0x01	0x04	0x00	0x01	0x00	0x01	0x60	0x0A
Response Temperature Value from Slave							
Device Address	Function Code	Bytes	Temp Hi	Tmep Li	CRC Hi	CRC Li	
0x01	0x04	0x02	0x01	0x31	0x79	0x74	

For example:
Temperature value=0x131, converted to a decimal 305 , so the actual temperature value = 305/10 = 30.5℃
Note: Temperature is signed hexadecimal number, temperature value = 0xFF33, converted to a decimal - 205, so the actual temperature = -20.5 ℃

RS485 Programm für XY-MD02 Feuchte

Übertragung bei read_register(register, Dezimalstellen, Function-code)

Lese Feuchte aus Register 2 mit 1 Kommastelle, Function-Code=4
humidity=instrument.read_register(2,1,4) # Register mit der Feuchtigkeit einlesen

Aus Datenblatt:

Master Read Humidity Command Frame(0x04)							
Device Address	Function Code	Starting Address Hi	Starting Address Li	Quantity Hi	Quantity Li	CRC Hi	CRC Li
0x01	0x04	0x00	0x02	0x00	0x01	0x90	0x0A

Response Humidity Value from Slave						
Device Address	Function Code	Bytes	Humidiyt Hi	Humidity Li	CRC Hi	CRC Li
0x01	0x04	0x02	0x02	0x22	0xD1	0xBA
For example: Humidity Value = 0x222, converted to a decimal 546, so actual humidity value = 546/10 = 54.6%RH						

RS485 Programm für XY-MD02


```
import minimalmodbus
import time

# Konfigurieren und Starten der Schnittstelle
# (Serielle Schnittstelle muß auch aktiviert sein [Startmenü/Einstellungen/Raspberry-Pi-Konfiguration/Schnittstellen/"Serial Port"])
instrument =minimalmodbus.Instrument('/dev/serial0',1) # 1 ist die Geräteadresse
instrument.serial.baudrate=9600
instrument.serial.timeout=0.5

while True:
    temperature=0
    humidity=0
    try:
        temperature=instrument.read_register(1,1,4) # Register mit der Temperatur einlesen
    except minimalmodbus.NoResponseError:
        print("Modbus timeout")            # Fehlermeldung
    try:
        humidity=instrument.read_register(2,1,4) # Register mit der Feuchtigkeit einlesen
    except minimalmodbus.NoResponseError:
        print("modbus timeout")        # Fehlermeldung
    #Ausgabe
    print('T=',temperature,'Hum=',humidity,'%RH')
    time.sleep(1)
```

CAN Daten senden

Schreib die Temperatur- und Feuchtigkeitswerte auf den CAN-Bus 2Pkt

Das Format soll wie folgt definiert werden:

Arbitration ID: Tischnummer (1-6)

Daten:
Byte1: Temperatur Vorkomma
Byte2: Temperatur Nachkomma
Byte3: Luftfeuchte (ohne Komma)
Byte4 bis 8: 0 (es werden immer 8 Bytes übertragen)

CAN Daten senden Beispielprogramm


```
import os
import can
import time

# CAN konfigurieren und starten
os.system('sudo ip link set can0 type can bitrate 100000')
os.system('sudo ifconfig can0 up')

can0 = can.ThreadSafeBus(channel = 'can0', bustype = 'socketcan_ctypes')

# Adresse zur Arbitrierung speichern
ARB_ADRESSE = 0x007 # hier die Tischnummer verwenden

# Ausführen einer Schleife zum Versenden von 256 CAN-Botschaften im 1-Sekunden-Abstand
for i in range(0,256,1):
    msg = can.Message(arbitration_id=ARB_ADRESSE, data=[0,1,2,3,4,5,6,i], extended_id=False)
    can0.send(msg)
    print(i)
    time.sleep(1)

print ("Ende")

# Bei Bedarf can stoppen
#os.system('sudo ifconfig can0 down')
```

CAN Daten lesen

Lies vom CAN-Bus die Daten laut nachstehendem Programm 2Pkt

Zum Testen senden wir laufend folgende Message:

Arbitration ID=0

Byte1= 30
Byte2= 5 30,5°C, 40%RH
Byte3= 40
Byte4-8: 0

CAN Daten lesen Beispiel


```
import os
import can
import wiringpi as wpi

# CAN-Bus konfigurieren und starten
# (Eintrag in /boot/config.txt muss vorhanden sein)
os.system('sudo ip link set can0 type can bitrate 100000')
os.system('sudo ifconfig can0 up')

can0 = can.ThreadSafeBus(channel = 'can0', bustype = 'socketcan_ctypes')

while True:
    # Warten auf den Empfang einer CAN-Botschaft, wird 3 Sekunden lang keine Botschaft empfangen, erfolgt eine Meldung "Timeout"
    msg = can0.recv(3.0) # Timeout von 3s
    if msg is None:
        print('Timeout, no message') # wenn lange keine Botschaft empfangen wurde --> Timeout
    else:
        # Ausgabe der empfangenen Daten
        msg_daten = msg.data
        print (msg_daten)
        # Ausgabe der arbitration_id und der CAN-Botschaft
        arbid = msg.arbitration_id
        print (arbid)
        print (msg)

#Beenden
os.system('sudo ifconfig can0 down')
```

Darstellung der Temperatur/Feuchtewerte

Programmiere mittels GUI ein Fenster, in dem laufend die Temperatur- und Luftfeuchtewerte aller Teilnehmer (0-6) in Textfeldern dargestellt werden. Verwende zur Zuordnung die arbitration ID.

3Pkt

Nachdem hier keine Eingabe notwendig ist, brauchen wir auch kein threading, wir können daher das Fenster mit

event, values = window.read(timeout=5)

aufrufen

Darstellung der Temperatur/Feuchtewerte


```
import PySimpleGUI as sg

T1=20
RH1=35
T2=22
RH2=40

layout = [
        [sg.Text("Tisch"), sg.Text("Temp:"),sg.Text("RH:")],
        [sg.Text("    1    "), sg.Text("",size=(6,1),key="-T1-"),sg.Text("", size=(8,1), key="-RH1-")],
        [sg.Text("    2    "), sg.Text("",size=(6,1),key="-T2-"),sg.Text("", size=(8,1), key="-RH2-")]
        ]
window = sg.Window('T/RH', layout)

while True:
    event, values = window.read(timeout=5)
    if event == sg.WIN_CLOSED:
        break
    window['-RH1-'].update(str(RH1))
    window['-T1-'].update(str(T1))
    window['-RH2-'].update(str(RH2))
    window['-T2-'].update(str(T2))
window.close()
```

Fernschaltung 1

Schließe dazu wieder die Lampe über die Relaisplatine am Raspi an.

Nun müssen wir uns auf ein Message-Protokoll für die Übertragung einigen. Der Vorschlag ist:

Arbitration ID: eigene Tischnummer (0-6)

Byte1: Tischnummer des Teilnehmers, dessen Lampe geschaltet werden soll
Byte2: 0 für aus, 255 für an
Byte3-8: 0

Wir senden zum Testen ein Steuersignal für alle Teilnehmer laut obigem Übertragungsprotokoll

Die eingehenden Messages müssen nun überprüft werden, ob ich angesprochen bin (Byte1= meine Tischnummer).
Danach je nach Inhalt von Byte2 die Lampe ein- oder ausschalten

Aufbau Lampenschaltung	1Pkt
Programm	3Pkt

Fernschaltung 2

Nun soll jeder Teilnehmer die Lampe eines anderen schalten können. 4 Pkt

Wir verwenden wieder die grafische Oberfläche mit threading.

In einem Thread soll permanent überprüft werden ob eine Message für mich bestimmt ist. Dann je nach empfangenen Daten die Lampe ein/ausschalten

In einem Fenster soll der anzusprechende Teilnehmer und dessen Lampenstatus eingegeben werden (Texteingabe oder Radiobuttons…)
Über die Schaltfläche ‚SEND' wird dann die Nachricht auf den Bus geschrieben

Beispielprogramm zum Download auf MOODLE