UNIVERSITÄT PADERBORN

Projekt – Ampelsteuerung

mit dem Nucleo-L053R8

Die Bearbeitungszeit beträgt drei Wochen. Die Abgabe der fertigen Programme ist am **29.11.2023** (In Panda bis vor der Übung um **13:59 Uhr** hochladen). Der Quellcode ist zu kommentieren und externe Quellen sind zu kennzeichnen! Halten Sie sich beim Programmieren an den *Style Guide* (pmi-doc-styleguide.pdf)

Vorbereitung

Trennen Sie den USB Stecker vom Nucleo Board, so dass keine Spannung anliegt. Stecken Sie nun das Display auf das Nucleo Board. Achten Sie auf die richtige richtige Polung und Kontaktierung. Stecken Sie anschließend die *Bob* Platine rechts unter das Nucleo Board. Verbinden Sie anschließend die *Alice* und die *Carlos* Platine mit *Bob*.

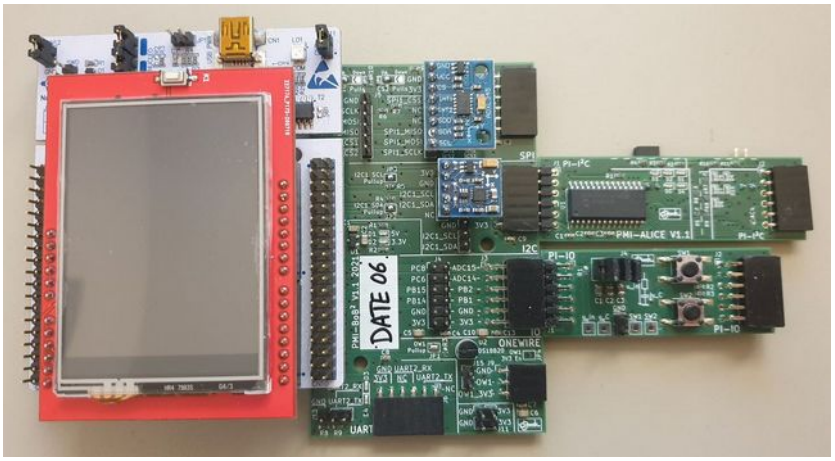

Abbildung 1: Zusammengesteckte Platinen

Aufgabe 1 - zyklisches Lauflicht

Da das Display schon zu viele GPIO Ports belegt, muss für die LEDs der Ampel ein sogenannter Port-Expander (MCP23017) Baustein benutzt werden. Dieser bindet zwei zusätzliche 8-Bit Ports via I^2C Interface an den Mikrocontroller an.

Aufgabe 1 a)

Schreiben Sie ein **eigenes Modul** (mcp23017.h und mcp23017.c), um den MCP23017 anzusteuern, damit dieser eventuell in einem anderen Projekt eingesetzt werden kann. Nutzen Sie zur Kommunikation das Modul ***i2c.h*** bzw ***i2c_hw.c*** aus dem Software Template. Benutzen Sie außerdem *defines* für die Adressen der Register des MCP23017s. Implementieren sie die Funktionen **`init_mcp23017()`**, **`read_mcp23017()`** und **`write_mcp23017`**() mit entsprechenden Parametern.

Aufgabe 1 b)

Benutzen Sie für diese Aufgaben Interrupts, um die Taster auszuwerten!
Entfernen Sie überflüssige Funktionsaufrufe und
entsprechenden Sourcecode aus dem Software Template!

Nutzen Sie dann dieses Modul , um die LEDs auf der Erweiterungsplatine *Alice* zyklisch aufleuchten zu lassen. Jede LED soll für 50 ms leuchten. Vernachlässigen Sie die grüne Straßen LED rechts (siehe Abbildung 2).

Hinweis: Nachdem Daten zum MCP23017 gesendet wurden, muss etwas gewartet werden, damit die Daten vom Chip verarbeitet werden können. Benutzen Sie die Funktion **`nop30()`** zum Warten.

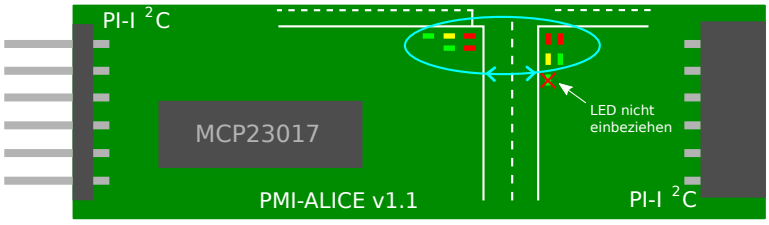

Abbildung 2: Alice Platine Mit LEDs

Benutzen Sie die Taster auf der *Carlos* Erweiterungsplatine, um das Lauflicht zu beeinflussen:

- Bei Betätigung des oberen Tasters (SW1) soll das Lauflicht gestartet bzw. angehalten werden. Ist das Lauflicht angehalten, so sollen keine LEDs leuchten.
- Wird der untere Taster (SW2) gedrückt, so soll sich die Richtung ändern.

Hinweis: Der Taster SW1 ist an Port PB2 und SW2 ist an Port PB1 angeschlossen, Pullup-Widerstände sind hardwareseitig verbaut (siehe pmi-doc-quickinfo.pdf Seite 7).
Achtung: Diese Taster prellen, also müssen diese softwareseitig entprellt werden!

Aufgabe 2: Ampelkreuzung mit Haupt- und Nebenstraße

In dieser Aufgabe soll eine Steuerung für eine Ampelkreuzung (Abbildung 3) mit Haupt- und Nebenstraßen implementiert werden. Die Hauptstraße wird bevorzugt behandelt, d.h. die Ampel leuchtet standardmäßig grün. Daraus folgt, dass die Nebenstraße standardmäßig rot leuchtet. Eine Ampelzyklus folgt der zeitlichen Abfolge aus Abbildung 4.

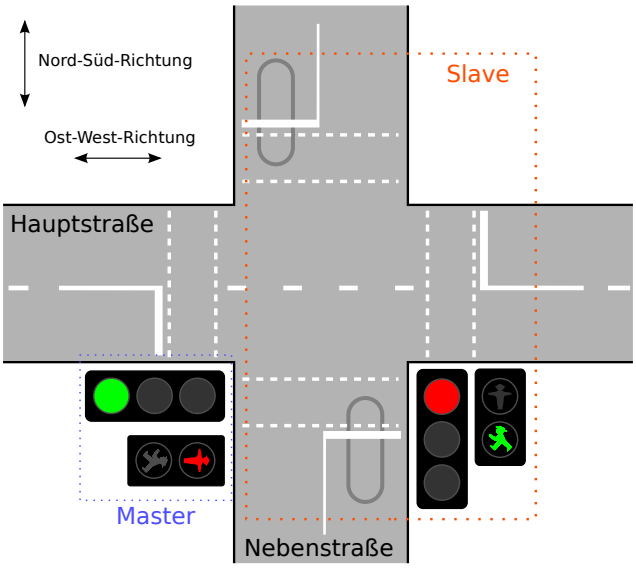

Abbildung 3: Ampelkreuzung

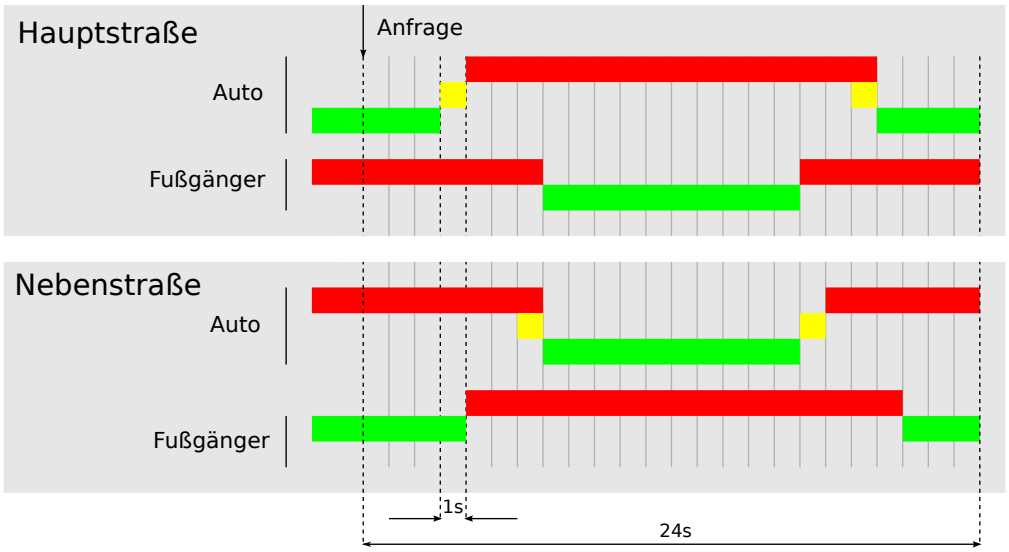

Abbildung 4: Ampelzyklus für Haupt- und Nebenstraße

Aufgabe 2 a)

Das aufgesteckte Display hat eine Auflösung von 320 x 240 Pixel. Für die Ansteuerung wird das Modul *ili9341* vom Software Template benötigt. Dieses Modul verfügt über die grundlegenden Funktionen für die Darstellung einzelner Pixels oder Text. Es können aber keine Schwarz-Weiß-Bitmaps, also Bilder, die aus einer Anordnung von Pixeln bestehen, dargestellt werden.

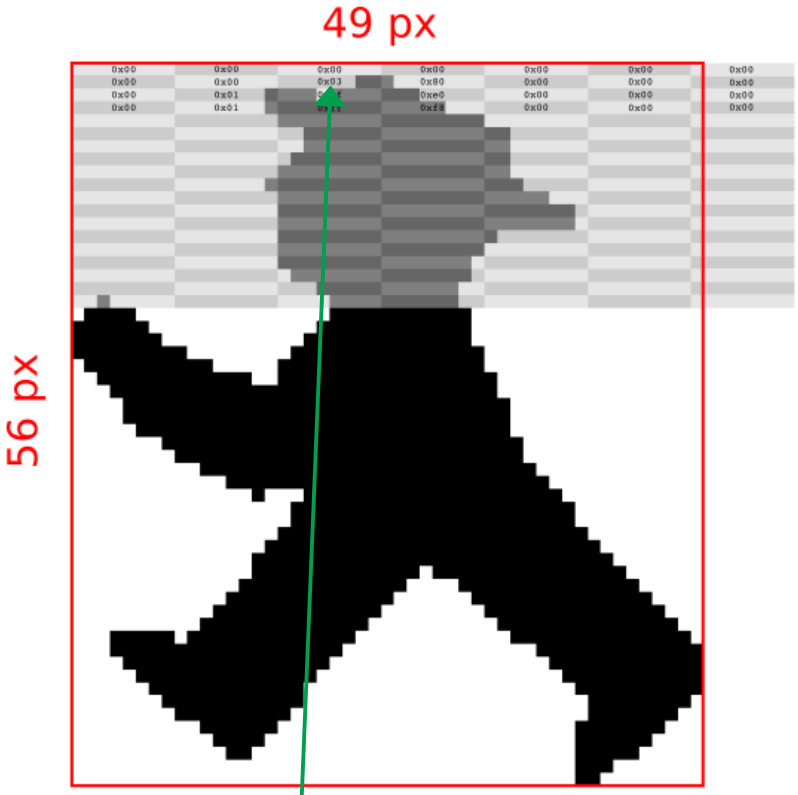

Abbildung 5: Bitmap Beispiel

Erweitern Sie das Modul *ili9341* (ili9341.h und ili9341.c) um eine Funktion, welche ein Array von Pixeldaten, beginnend an einer Koordinate (x,y), darstellt. Die Angaben über die Breite und Höhe (w, h) des Bildes müssen dabei mit angegeben werden, sowie die Farbe des Vordergrundes als auch die Farbe das Hintergrundes. Nutzen Sie die bereits vorhandenen Funktionen aus dem ili9341 Modul.

Ergänzen Sie also folgende Funktionsdeklaration:

```
/**
* @brief ?
* @param ?
*/
void ili9341_draw_bmp_h(uint16_t x, uint16_t y, uint16_t w, uint16_t h, uint8_t* bmp_data,
 uint16_t color, uint16_t bgcolor);
```

Das Bitmap Array *bmp_data*, welches benötigt wird, ist die Bit - Repräsentation des Bildes. Diese Bits werden horizontal zu je einem Byte zusammengefasst.

Das Beispiel Bitmap in Abbildung 5 hat die Größe von 49x56 Pixel. Der Anfang dieses Bitmaps als uint8_t Array lautet:

```
const uint8_t go_49x56 [] = {
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x80, 0x00, 0x00, 0x00, 0x00, 0x01,
   0xef, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x01, 0xff, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0xff,
...};
```

Aufgabe 2 b)

Stellen Sie das Bild *bmp_go_49x56.bmp* (bmp_go_49x56.h) mit Ihrer neuen Funktion an der Position x=20, y=20 auf dem Display dar. Als Hintergrundfarbe soll **schwarz** gewählt werden, als Vordergrundfarbe **grün**.

Aufgabe 2 c)

Nun geht es an die eigentliche Ampel. Dazu wird eine Steuerung benötigt, welche die LEDs in der richtige Reihenfolge zum richtigen Zeitpunkt ansteuert. Außerdem muss die Steuerung noch auf externe Ereignisse wie z.B. Taster reagieren können.

Ein Hilfsmittel zur Planung der Steuerung ist eine *Zustandsübergangstabelle*, in der die Ampelzustände festgelegt werden und die Bedingungen, wann ein Zustandswechsel in welchen Zustand erfolgt.

Zustandsname	Buchstabe Hauptstraße	Buchstabe Nebenstraße	Bedingung	Wartezeit	nächster Zustand
MS_G	G	g	Taster abfragen	3	MS_Y
MS_Y	Y	g	time=0	1	MS_R
MS_R	R	R	time=0	2	...
...	...		...	…	...

Tabelle 1: Schema der Zustandsübergangstabelle, Abkürzungen siehe Fußnote[1]

Die Bedeutung der Buchstaben für Haupt- und Nebenstraße entnehmen sie nachfolgender Tabelle:

Buchstabe	Verkehrsampel	Fußgängerampel
O	aus	aus
R	**rot**	**rot**
g	**rot**	**grün**
Y	**gelb**	**rot**
r	aus	**rot**
G	**grün**	**rot**
n	**grün**	aus
D	**rot** + **gelb**	**rot**

Tabelle 2: Buchstabe-LED Codierung

Erstellen Sie die Zustandsübergangstabelle für die Ampelsteuerung, wie in Tabelle 1 angefangen. Nehmen Sie diese Tabelle als Ausgangspunkt für Ihre eigene. Daraus soll mit Hilfe der Buchstaben aus Tabelle 2 ersichtlich werden, wann welche LED für wie lange leuchtet und welche Bedingung

1 TL=Traffic Lights → Verkehrsampel, PL=Pedestrian Lights → Fußgängerampel, MS=Main Street → Hauptstraße, SS=Side Street → Nebenstraße, G=Green, Y=Yellow, R=Red

für einen Zustandswechsel erfüllt sein muss. Folgen Sie für das Timing dem Ampelzyklus aus Abbildung 4. **Legen sie die Zustandsübergangstabelle der Abgabe als PDF bei!**

Aufgaben 2 d)

Benutzen Sie für diese Aufgaben Interrupts, um die Taster auszuwerten!
Entfernen Sie überflüssige Funktionsaufrufe und
entsprechenden Sourcecode aus dem Software Template!

Implementieren Sie die Steuerung. Nutzen Sie für die Hauptstraßenampel die 5 LEDs auf der linken Seite der *Alice* Erweiterungsplatine und für die Nebenstraßenampel die rechten LEDs (Abbildung 6). Geben Sie außerdem die Zustände der Fußgängerampel der Hauptstraße als Grafik auf dem Display aus. Benutzen Sie die beiden Dateien *stop_51x56.bmp* und *go_49x56.bmp* respektive *bmp_stop_51x56.h* und *bmp_go_49x56.h* als Grafik für die Fußgängerampel mit den entsprechenden Farben.

Fährt ein Auto an die Ampel der Nebenstraße, so signalisiert dies eine Induktionsschleife in der Straße und eine Grün-Phase für das Auto wird angefordert.

Sollte ein Fußgänger die Hauptstraße überqueren wollen, so muss er an der Ampel die Grünphase für die Fußgängerampel durch Betätigen des Tasters anfordern.

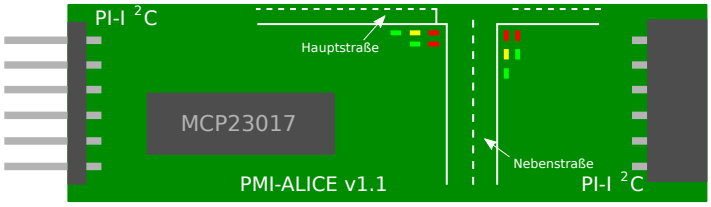

Abbildung 6: Alice Erweiterungsplatine

Die Induktionsschleife soll durch den unteren Taster der *Carlos* Erweiterungsplatine ausgelöst werden, die Anfrage für die Fußgängerampel über die Hauptstraße soll durch den oberen Taster erfolgen. **Achtung:** Der Zyklus darf nicht automatisch wieder starten, wenn während der Ausführung erneut Tasten gedrückt wurden!

Zur Umsetzung:

Definieren Sie einen neuen Aufzählungsdatentyp `state_t`, der die Zustandsnamen aus ihrer in Aufgabenteil 2c) erstellten Tabelle enthält:

```
typedef enum {MS_G=0, MS_Y, MS_R, …} state_t;
```

Achten Sie darauf, dass die Reihenfolge der Zustände in der *enum* Definition gleich sein muss mit der Reihenfolge der Zustände in der Tabelle. Die Tabelle selbst wird in C als Array einer Struktur abgebildet:

```
typedef struct {
    char    led_character_ms; // main street led character
    char    led_character_ss; // side street led character
    uint8_t condition;
    uint8_t wait_time;
    state_t next_state;
} tl_state_t;
state_t state = <INITIAL_STATE>;
tl_state_t state_table[???] = {
/* leds_ms   leds_ss condition?
   |         |       |  waittime in s
   |         |       |  |
   |         |       |  |  next state
   |         |       |  |  |      */
 {'G',      'g',     1, 3, MS_Y}, // next state → "Main Street Yellow", 3 seconds after button press
 {'Y',      'g',     0, 1, MS_R}, // next state → "Main Street Red" after 1 socond
  ...
};
```

Für den eigentlichen Zustandswechsel muss jetzt der Inhalt der Array Einträge `state_table` in einer Schleife ausgewertet und der Folgezustand gesetzt werden. Die Zustände der LEDs werden durch die Buchstaben der Haupt- und Nebenstraße vom Eintrag aus dem Array gesetzt.
Nutzen Sie für die Umrechnung der Buchstaben in die entsprechenden Bitmuster für die LEDs die `switch/case` Anweisung!
Hinweis: https://www.mikrocontroller.net/articles/Statemachine#Umsetzung_in_Tabellenform

Aufgabe 3 – Kommunikation mit 2 Boards

Nun sollen zwei *alice* Entwicklungsplatinen wie in Abbildung 7 verbunden werden, so dass eine wirkliche Ampelkreuzung mit 4 Ampeln entsteht. Damit es nicht zu Synchronisierungsproblemen kommt, soll die untere *alice* Platine mit dem Nucleo Board die Hauptsteuerung sein, die obere *alice* Platine mit einem weiteren Nucleo Board soll nur Befehle vom unteren Nucleo Board entgegen nehmen.

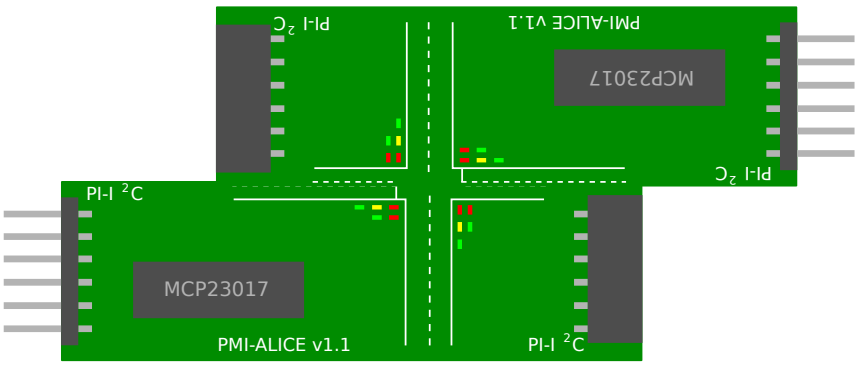

Abbildung 7: 2 verbundene Alice Entwicklungsplatinen

Die *Hauptsteuerung* (untere Platine) beinhaltet die eigentliche Steuerung (siehe Aufgabe 2) und gibt an, welche LEDs die obere *alice* Platine aktivieren soll. Außerdem muss sie auch von der oberen Platine erfahren, ob die Induktionsschleife oder der Taster ausgelöst wurde. Dazu sendet die untere Platine über die UART Schnittstelle sekündlich den Status der LEDs und erwartet als Antwort darauf, ob der Taster oder die Induktionsschleife ausgelöst hat.
Die obere Platine wiederum nimmt nur Befehle von der *Hauptsteuerung* entgegen und setzt diese um. Außerdem wertet sie den Zustand der Induktionsschleife und des Tasters aus und gibt diesen Zustand der *Hauptsteuerung* auf die sekündliche Anfrage zurück.
Bei einem Kommunikationsabbruch soll die Hauptstraße auf grün geschaltet werden, und die Ampel der Nebenstraße soll sekündlich gelb blinken. Dies gilt sowohl für die untere, als auch für die obere Platine.
Funktioniert die Kommunikation wieder soll der normale Betrieb fortgesetzt werden.

Aufgabe 3 a)

Die UART Implementierung, die dem Software Template beiliegt, ist eine sogenannte *blockierende* Implementierung: Bei einem lesenden Zugriff auf die serielle Schnittstelle wird so lange gewartet, bis auch wirklich Daten ankommen. Sollte die Kommunikation gestört sein, so kommen keine Daten mehr an und das Programm würde dann "hängen" bleiben - es blockiert also.
Erstellen Sie auf Grundlage des Software Templates der UART Schnittstelle ein neues Modul *uart_irq*. Initialisieren Sie die Schnittstelle so, dass bei ankommenden Daten ein Interrupt ausgelöst wird. Der Interrupt selber soll in der Funktion `void USART2_IRQHandler(void)` abgearbeitet werden.
Hinweis: Ähnlich wie bei der Konfiguration eines Port Interrups muss auch der UART Interrupt beim NVIC angemeldet und aktiviert werden. Die entsprechende Definition der Interrupt Nummer lautet `USART2_IRQn`.

Aufgabe 3 b)

Erstellen Sie ein Art Sequenzdiagramm für die Kommunikation basierend auf der nachfolgenden Kommunikationsbeschreibung. Zur Darstellung des Sequenzdiagramms kann folgender Link benutzen werden: https://sequencediagram.org. Achten Sie darauf, dass mit Hilfe des Sequenzdiagramms die Kommunikation, das Protokoll und das Timing nachvollzogen werden kann: Wer sendet was? Timing (Timeout) ersichtlich? Wer fragt und wer antwortet auf was?
Es soll folgende Kommunikation über die UART Schnittstelle erfolgen:

Die untere Platine sendet sekündlich **2 Buchstaben** gefolgt von ↵ (ENTER, ASCII Wert 13) zur oberen Platine. Der erste Buchstabe zeigt an, welche LEDs auf der Hauptstraßenampel leuchten sollen, der zweite, welche LEDs auf der Nebenstraßenampel leuchten sollen. Nutzen sie die Buchstabencodierung aus Tabelle 2. Die obere Platine sendet daraufhin den Zustand des Tasters bzw. den Zustand der Induktionsschleife als **ein Buchstabe** gefolgt von einem ↵ zurück:
y↵→Taster gedrückt bzw. Induktionsschleife hat ausgelöst, n↵→keine Anfrage.

Beispiel 1:
Anfrage: Gg↵ → Hauptstraße hat grün, Nebenstraße hat rot
Antwort: n↵ → keine Anfrage
Beispiel 2:
Anfrage: GY↵ → Hauptstraße hat grün, Nebenstraße hat gelb
Antwort: y↵ → Taster gedrückt bzw. Induktionsschleife hat ausgelöst

Aufgabe 3 c)

Implementieren Sie die beiden Steuerungen für die obere und die untere *alice* Platine in **einem** *Visual Studio Code* Projekt. Nutzen Sie dafür die Möglichkeit der Präprozessor Anweisungen **#define**, **#ifdef**, **#ifndef** und **#endif**.

Als Grundlage der *Hauptsteuerung* soll die Aufgabe 2 dienen. Die Funktionen der einzelnen Nukleo Boards kann getestet werden, indem z.B. das Terminal des PCs als Gegenstelle genutzt wird. Außerdem liegen dem Projekt noch zwei Python Skripte zum Testen bei:

Die untere Platine bzw. Haupsteuerung kann mit dem Skript *test-main-control.py* getestet werden, die obere Platine mit dem Skript *test-slave-control.py*.

Die beiden Skripte senden entsprechende Daten über die UART Schnittstelle vom PC zum Nukleo Board. Gegebenenfalls muss die Zeile `ser.port = '/dev/ttyACM0'` für die jeweilige Schnittstelle am PC angepasst werden.

Viel Spaß!