Dokumentation Mikrokontrolleraufgabe

1. **Verwendete Peripherie**

*I2C-Kommunikation*

Für die Kommunikation mit den Sensoren wurde die I2C1 Schnittstelle im Fast Mode (400 khz) verwendet. SDA wird an PB7 und SCL an PB6 angeschlossen. I2C-Kommunikation findet ausschließlich mit den Blocking-Funktionen statt; eine Interrupt-gesteuerte I2C-Übertragung ist nicht implementiert.

*Timer*

Es werden die Timer TIM 3 und TIM 16 verwendet. TIM 3 übernimmt die PWM-Generation für die beiden Pins PC8 und PC9, an denen die grüne bzw. blaue LED angeschlossen ist. Mit diesem PWM Signal wird die Helligkeit der beiden LEDs mittels des Befehls \_\_HAL\_TIM\_SET\_COMPARE(&htim3,TIM\_CHANNEL\_4,WERT) gesteuert. TIM 3 ist folgendermaßen konfiguriert:

* Prescaler: 0
* Counter Mode: Up
* Counter Period: 511
* Internal Clock Division: No Division
* Auto-reload preload: Enable
* Tim 3 gloabl interrupt: Disabled
* Slave Mode: Disable
* Trigger Source: Disable
* Clock Source: Internal Clock
* Channel 1&2: Disable
* Channel 3&4: PWM Generation CH3/4

**Hinweis:** Bei der Code-Generation von Timer 3 tritt in der Cube IDE ein Bug in stm32f0xx\_hal\_msp.c (Zeile 202) auf: Es wird fälschlicherweise „GPIO\_InitStruct.Alternate = GPIO\_AF0\_TIM3;” generiert, obwohl es GPIO\_AF1\_TIM3 heißen müsste. Der Fehler muss jedes Mal, wenn der Code neu generiert oder geändert wird, manuell behoben werden. Nur GPIO\_AF1\_TIM3 ist nämlich in stm32f0xx\_hal\_gpio\_ex.h definiert.

TIM 16 ist ein Timer, der deutlich langsamer hochzählt als TIM3. Er wird verwendet, um ein regelmäßiges Interrupt auszulösen, mit dem die LEDs periodisch an- bzw. ausgeschaltet werden, wenn die Abweichung des Sensors vom Nullwert kleiner als 5° ist. Dieser Interrupt ist einer von zwei Interrupts im Programm. TIM 16 ist folgendermaßen konfiguriert:

* Prescaler: 47999
* Counter Mode: Up
* Counter Period: 249
* Internal Clock Division: No Division
* Auto-reload preload: Enable
* TIM 16 global interrupt: Enabled
* Channel 1: Disable

1. **Programmdurchlauf vor der Endlosschleife**

In der Datei main.c werden zwei globale Variablen des Typs volatile bool initialisiert. Diese beiden Variablen, Tara und LEDistAN werden während der Interrupts geändert. Die Funktion int main(void) beginnt mit der Initialisierung der Sensor-Rohdatenvariablen, einiger anderer Variablen, der Initialisierung der HAL, System Clock und der Peripherien. Anschließend wird der PWM Timer gestartet, die Pulsweite aber vorerst auf Null gesetzt, um die LEDs auszuschalten.

Anschließend wird die Funktion InitialisiereGyro() aufgerufen, die in der Datei „Gyro.c“ zu finden ist. Die Funktion beginnt damit, den FXAS21002C-Sensor in den Standby-Modus zu versetzen, da dies im Datenblatt empfohlen wird, wenn gewisse Register beschrieben werden sollen. Anschließend werden CTRL\_REG0 und CTRL\_REG1 konfiguriert. Die gewählte Konfiguration ist wie folgt:

CTRL\_REG0:

* LPF cutoff frequency: [01] für 16Hz bei ODR = 100 Hz
* High-pass filter enable: [0] für OFF
* Full-Scale-Range selection: [10] -> Range = +-500dps; Sensitivität = 15,625 mdps/LSB

CTRL\_REG1:

* Reset: [0] für NEIN
* Self Test: [0] für OFF
* Output Data Rate: [011] für 100 Hz
* Active Mode: [1] für ON

Nach jeder Registerkonfiguration wird dasselbe Register ausgelesen und geprüft, ob der gewünschte Wert übernommen wurde. Ist das nicht der Fall, bricht die Initialisierungsfunktion mit einem Fehler ab. Anschließend wird die Device ID abgefragt. Ist auch diese korrekt, ist die Initialisierung erfolgreich und die Initialisierung des FXOS8700CQ Sensors startet mit der Funktion InitialisiereFXOS8700CQ(). Es werden folgende Register konfiguriert: M\_CTRL\_REG1, M\_CTRL\_REG2, CTRL\_REG1. Anfangs wird auch hier der Sensor in Standby versetzt und zum Schluss mit der Konfiguration von CTRL\_REG1 wieder aktiviert. Wie bei der Initialisierung des FXAS21002C wird ebenfalls jedes konfigurierte Register ausgelesen und kontrolliert. Sind alle Register korrekt gesetzt und die Device ID abgefragt und überprüft, gibt die Funktion ebenfalls den Wert *true* zurück. Die Konfiguration des FXOS8700CQ-Sensors ist unten angegeben:

M\_CTRL\_REG1:

* Auto Calibration: [1] für ON
* One-shot magnetic reset: 0 für OFF
* Oversample ratio: [011] für OSR = 16 bei 6,25 Hz ODR
* Hybrid Mode setting: [11] für Hybrid Mode

M\_CTRL\_REG2:

* Hybrid-Autoincrement-Mode: [1], um beim Burst-Read von Register 0x06 auf Register 0x33 zu springen
* Magnetic sensor reset (degaussing) Frequenz: [00] für Standardwert

CTRL\_REG1:

* Output data rate selection: [101] für ODR = 12,5 Hz in mag-only-mode bzw. 6,25 Hz in hybrid mode
* Inoise: [1] für Reduced noise mode ON
* Fast Read Mode: [0] für OFF
* Avtive Mode: [1] für ON

Wenn beide Initialisierungsfunktionen erfolgreich durchlaufen sind, blinkt die grüne LED 3x um die Initialisierung des Gyroskops anzuzeigen. Danach blinkt die blaue LED dreimal, um die Initialisierung des FXOS8700CQ anzuzeigen.

1. **Endlosschleife**

In der Endlosschleife werden folgende Schritte immer wiederholt:

1. Die Drehraten werden ausgelesen
2. Der FXOS8700CQ wird ausgelesen
3. Es wird geprüft, ob ein Tara-Interrupt seit letztem Schleifendurchlauf vorliegt. Ein solcher Interrupt wird ausgelöst, wenn der an PA0 angeschlossene blaue Knopf gedrückt wird. Ist dies der Fall, wird der aktuelle Azimut mit den gerade ausgelesenen Sensordaten berechnet und in der vorzeichenbehafteten Einheit [° Abweichung von Norden] in einer double-Variable gespeichert.
4. Die aktuelle Abweichung von der Tara-Ausrichtung wird berechnet. Die Ausrichtung bezieht sich auf die **positive x-Achse** des Sensors. Wurde der blaue Knopf noch nicht gedrückt, ist die Tara-Ausrichtung mit 0° Abweichung initialisiert; es wird also der reine Azimut berechnet. Die Funktion **BerechneAusrichtung**(int16\_t \*x\_axis\_Mag, int16\_t \*y\_axis\_Mag, **double** \*TaraHeading) gibt die aktuelle Abweichung von der gewählten Referenz in Milligrad an.
5. Es wird überprüft, ob die Abweichung kleiner gleich 5000 Milligrad ist. Ist das der Fall, wird TIM16 gestartet und anschließend überprüft, ob sich der Sensor im oder gegen den Uhrzeigersinn um die z-Achse dreht. Die entsprechende LED wird dann angeschaltet, wenn die volatile bool Variable LEDistAN, die sich immer dann ändert, wenn TIM16 ein Interrupt auslöst, den Wert *true* hat. Sobald TIM16 jedoch hochgezählt hat, wird der Interrupt die Variable LEDistAN auf *false* setzen, was zu einem Abschalten der LED führt. Dieser periodische An- und Abschaltvorgang führt zu einem Blinken der entsprechenden LED. Ist die Abweichung jedoch größer als 5000 Milligrad, wird der Timer TIM16 wieder gestoppt.
6. Die anschließende if-else-Abfrage steuert das Verhalten der LEDs für den Fall, dass die Abweichung größer als 5000 Milligrad ist. In diesem Fall soll die LED nicht blinken, sondern in der Helligkeit mit der Drehgeschwindigkeit variieren. Es wird geprüft, ob die z-Achse positive Werte (Drehung gegen den Uhrzeigersinn), oder negative Werte (Drehung IM Uhrzeigersinn) annimmt. Abschließend wird die Pulsweite der jeweiligen LED linear mit der Drehgeschwindigkeit des Sensors verändert und die Endlosschleife beginnt von Neuem.

