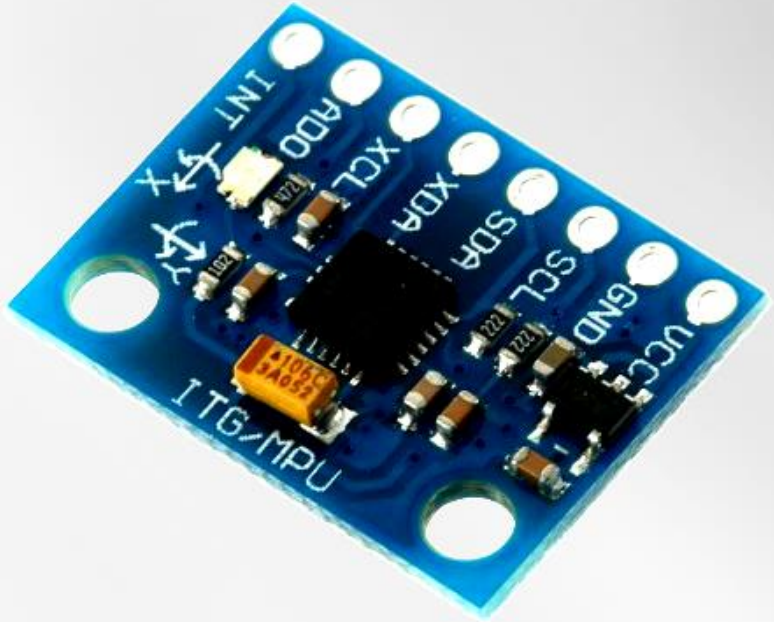
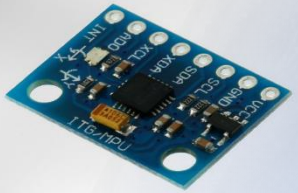
Sensorik (IMU)

Einführung

Inertial Measurement Unit: MPU6050

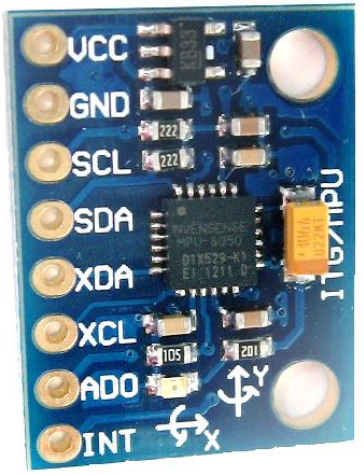


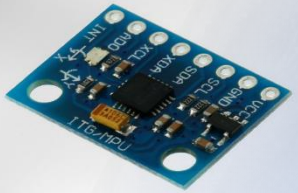


Inhalt

Umfang: ca. 1-3 Zeitstunden

- Sensorik
- Lagesensorik: IMU (Intertial Measurement Unit)
- Interface: TWI (Two Wire Interface)
- Aufgaben





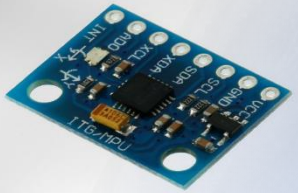
Sensorik

Sensorik:

- Sensorik ist die Anwendung von technischen Bauteilen (Sensoren) zur Messung.
- Ein Sensor ist ein technisches Bauteil, das physikalische oder chemische Eigenschaften erfasst.
- Die Erfassung bzw. Messung erfolgt mittels physikalischer oder chemischer Effekte und wird dazu in ein elektrisches Signal umgewandelt.
- Beispiele: Temperatursensor, Beschleunigungssensor, Drehratensensor

Einige wesentliche Aspekte:

- Messgröße: Was wird gemessen? z.B. Temperatur
- Messeinheit: In welcher Einheit wird gemessen? z.B. °C
- Skalierungsfaktor: Teil der Kalibrierung. Oftmals existiert ein linearer Skalierungsfaktor zwischen „Rohwert“ und „Endwert“. Dies ist eine lineare / dynamische Abweichung.
- Offset / Bias: Weiterer Teil der Kalibrierung. Neben dem Skalierungsfaktor existiert oftmals eine stationäre Abweichung, d.h. fester Offset / Bias.
- Auflösung: Mit welcher Auflösung wird gemessen, d.h. welche kleinsten Änderungen können dargestellt werden.
- Wiederholrate: Wie oft können Messungen erfolgen? Im Allgemeinen gilt: Je schneller, umso besser



Sensorik

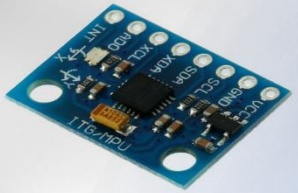
Kalibrierung:

- Mit Kalibrieren ist das Erfassen von Abweichungen und korrekte Einstellen eines Sensors.

- Solche Abweichungen sind oftmals nicht zufällig, sondern systematisch. Oftmals gilt für die Abweichungen folgende Formel: m = s * r + b, dabei gilt:
  m ist der gültige (kalibrierte) Messwert.
  s ist der lineare Skalierungsfaktor.
  r ist der Rohwert (nicht kalibrierte Messwert).
  b ist der Bias / Offset. Dies entspricht der Nullpunktabweichung.

- Die Formel besagt, dass und wie der gültige Messwert unter Verwendung von s (Skalierungsfaktor) und b (Offset) aus dem Rohwert bestimmt werden kann.

- Wichtige Voraussetzung: s und b sind (zumindest zeitweise) konstant



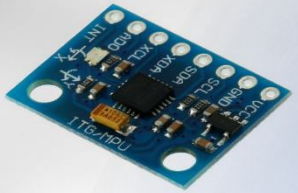
Lagesensorik

Lagesensorik sind Sensoren, die die Lage im Raum erfassen. Unter Lage wird (hier) die Orientierung und nicht die Position verstanden. Klassische Lagesensoren, die auch in jedem Quadrokopter zum Einsatz kommen, sind:

- Gyroskop / Gyrometer: Ein Drehratensensor und der wichtigste Sensor jedes Quadrokopters. Es existiert kein Quadrokopter, ohne Gyroskop. Das Gyroskop misst Drehbewegungen (Rotationen). Der Lageregler benötigt permanent diese Information, um den Quadrokopter stabil in der Luft zu halten. Bei Ausfall der Lagesensorik / Lageregelung stürzt der Quadrokopter unweigerlich ab. Ohne Gyroskop + Regelung ist ein Flug nicht möglich.
- Accelerometer: Ein Beschleunigungssensor und der zweitwichtigste Sensor jedes Quadrokopters. Der Gyroskop akkumuliert systembedingt, d.h. prinzipiell, immer kleine Fehler, wodurch die berechnete Lage mit der Zeit von der wahren Lage abweicht. Diese Abweichung nennt man Drift. Der Beschleunigungssensor misst Beschleunigungen, darunter auch die Gravitation der Erde. Aus der Gravitation der Erde lässt sich die horizontale Lage bestimmen. Damit lässt sich der Drift kompensieren und die Lage über Lage Zeit korrekt bestimmen.

Weitere Sensoren:

- Magnetometer: Oftmals kommen noch Magnetsensoren zum Einsatz, mit denen analog zum Accelerometer der Drift kompensiert werden kann. Denn der Beschleunigungssensor kann Drehungen um die eigene Achse nicht erfassen und daher auch nicht kompensieren, wohl aber das Magnetometer. Es funktioniert vergleichbar einem Kompass.



Lagesensorik / IMU

IMU = Inertial Measurement Unit = Inertiale Messeinheit

Definition:

Eine IMU ist die Kombination mehrerer Inertialsensoren (Trägheitssensoren). Üblich ist ein triaxialer Aufbau aus Beschleunigungs- (Accelerometer) und Drehratensensoren (Gyroskope). Die IMU ist die Sensoreinheit eines Inertialen Navigationssystemes, welches Positions- und Orientierungsänderung erfasst. Dies kann als Input der Orientierungs-darstellung zur Verwendung in einer Lageregelung hergenommen werden.

Positionsbestimmung (1D, vereinfacht):

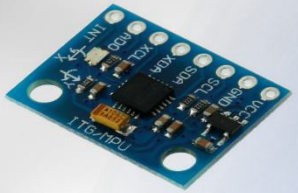
$$P = \int \int a$$

Ungenau wegen doppelter Integration.
Nur für sehr kurze Zeiten brauchbar.
Meist wird GPS hinzugezogen.

Orientierungsbestimmung (1D, vereinfacht):

$$\varphi = \int \omega$$

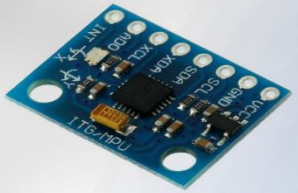
Genauigkeit ok, aber Driftproblem.
Meist werden weitere Sensoren hinzugefügt.



IMU (Intertial Measurement Unit)

IMU ITG/MPU

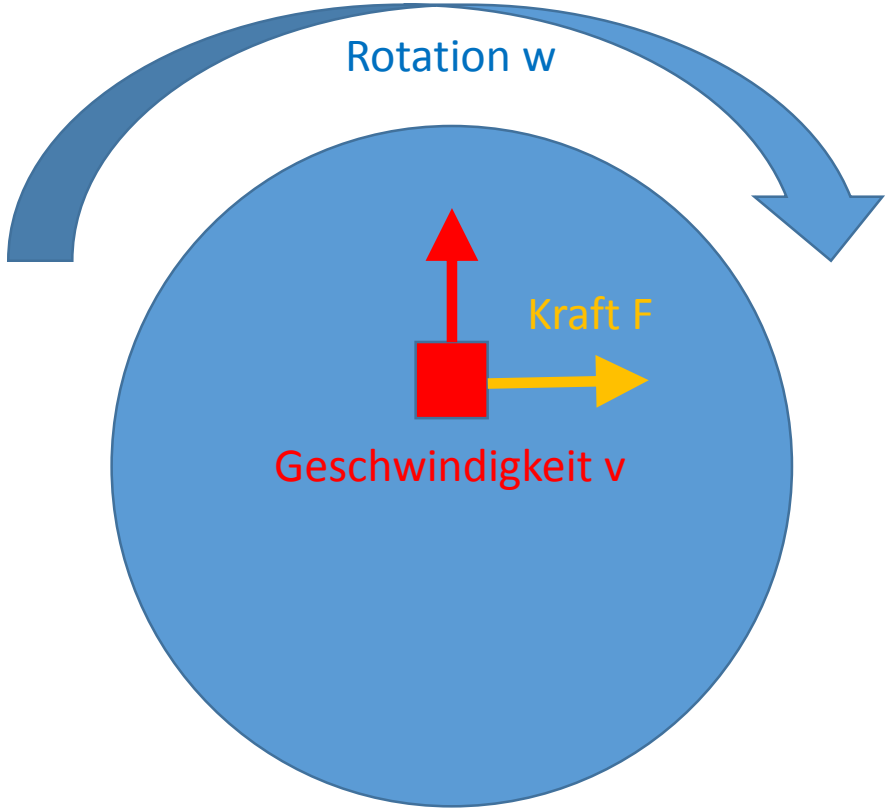
- MPU6050
- Ein Chip mit Gyroskope + Accelerometer
- Ansteuerung per TWI
- Datenblätter:
  - IMU_MPU_6000+6050_RegisterMap.pdf
  - IMU_MPU_6000+6050_Sepcs.pdf
- Bevor der Sensor verwendet werden kann, muss er konfiguriert und kalibriert werden

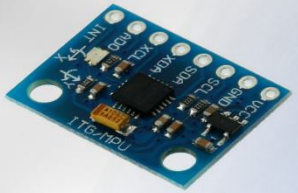


IMU (Intertial Measurement Unit)

Funktionsprinzip Gyroskop (Gyro):

- Messprinzip Corioliskraft (analog zur Lorenzkraft):
  - Bewegte Teilchen (v) (Ursache)
  - gleichzeitige Rotation (w) (Vermittlung)
    -> Wirkung: Kraft (F) senkrecht zu Ursache und Vermittlung
  - Kraft steht Senkrecht auf Drehachse und Bewegung
  - Tritt auf in rotierenden Bezugssystemen (w) bei gleichzeitiger Bewegung (v)
  - F = - 2 · m · ω · v
- MEMS: Mikro-Elektro-Mechanisch
- Triaxial: x,y,z = 3 DOF: Degree of Freedom, Freiheitsgrade („unabhängige Richtungen“)
- Drehratensensor: Misst Drehgeschwindigkeiten
- Messung: Kapazitiv

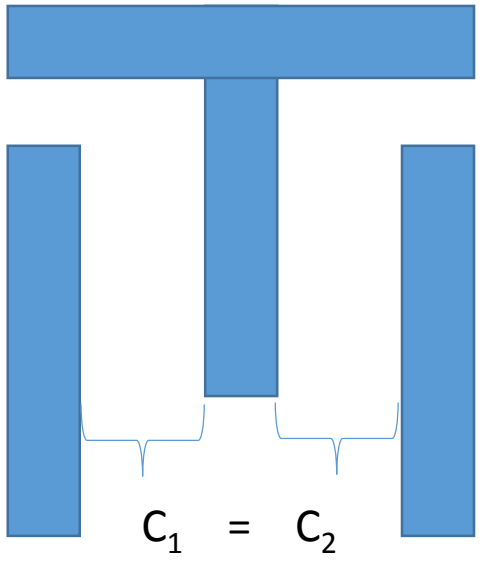


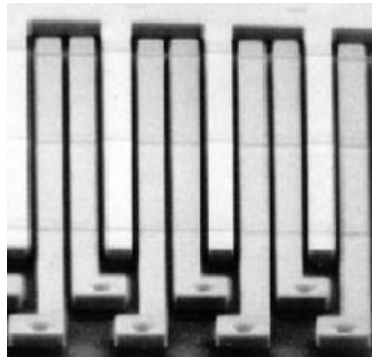


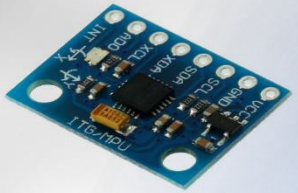
IMU (Intertial Measurement Unit)

<u>Funktionsprinzip Accelerometer (Acc)</u>:

- Messprinzip Feder-Masse Prinzip
  - Zwei Kammreihen
  - Aufgrund von Trägheit ändern sich die Abstände
  - Messung der Abstandsänderung kapazitiv
- MEMS: Mikro-Elektro-Mechanisch
- Triaxial (x,y,z = 3 DOF)
- Beschleunigungssensor: Misst Beschleunigungen (translatorisch)



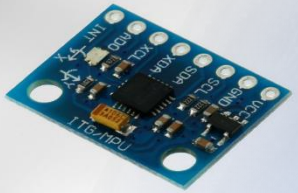




TWI (Two Wire Interface)

TWI

- Serieller Master-Slave-Bus: Byte für Byte
- Auch I²C oder I2C (inter-Integrated Circuit) genannt
- 2 Busleitungen: SCL (serial clock), SDA (serial data line)
- Vorteile: günstig, wenige Verdrahtungen
- Nachteile: störanfällig, geringe Leitungslänge
- Taktmodi: Standard bis 100 kHz, Fast Mode bis 400 kHz
- Übertragungsrate: max. 3,4 Mbit/s (relativ langsam)



TWI (Two Wire Interface)

TWI Datenaustausch

- Startsignal
- 1 Byte Adresse (7bit) + R/W (1bit = 1 für Write)
- ACK (Acknowledge / Bestätigung vom Slave)
- Byteweise Datenpakete je quittiert mit ACK
- Stopsignal

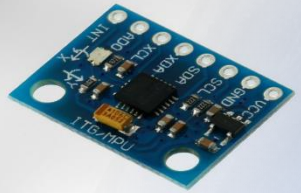
- Die genannte Adresse ist die I²C/TWI Slave Adresse des angesprochen Teilnehmers.

Implementierung:

- Verwende TWI Framework

Datenpakete:

- Beim AVR Framework mind. 2 Bytes
- 1. Byte **Kommando** bzw. Adresse, für R/W
- 2. Bytes + Weitere sind **Daten** (R/W)
- Bei Burstmode werden mehrere Bytes von einer Anfangsadresse angefangen übertragen.

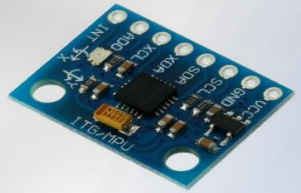


TWI (Two Wire Interface)

TWI Framework

- Verwendet Struktur twi_package_t
- chip = TWI Slave Adresse
- addr = **Kommando**
- addr_length = Länge des Kommandos in Bytes
  Üblicherweise = 1
- buffer = Pointer auf Speicherbereich der **Daten**
  - Send / Write (W): Diese Daten werden versendet
  - Receive / Read (R): Hierhin werden die empfangenen Daten geschrieben
- length = Anzahl Bytes die gesendet / empfangen werden

```
typedef struct
{
  char chip;// TWI Slave Address

  unsigned int addr;// Command Byte to be sent

  int addr_length;// Amount of Command Bytes

  void *buffer;// Buffer for Data (Send/Receive)

  unsigned int length;// Amount of Data Bytes
} twi_package_t;
```



TWI (Two Wire Interface)

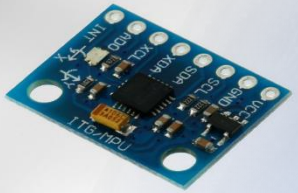
TWI Framework Beispiel

```
char data_received_mpu_acc[6];
twi_package_t packet_mpu6000_acc;

packet_mpu6000_acc.chip = 0x69;
packet_mpu6000_acc.addr_length = 1;
packet_mpu6000_acc.length = 6;
packet_mpu6000_acc.addr = 0x3B;
packet_mpu6000_acc.buffer = data_received_mpu_acc;

read_from_twi(&packet_mpu6000_acc, 0);
```

Danach stehen die 6 Daten-Bytes aus dem Sensorspeicher beginnend mit Adresse 0x3B in data_received_mpu_acc

Bei einem Fehler wird eine Nachricht ausgegeben. Anhand des zweiten Parameters, hier 0, kann diese Fehlernachricht zugeordnet werden.

```
Lesen / Daten empfangen:
read_from_twi(const twi_package_t *twi, int error_id);

Schreiben /  Daten senden:
write_to_twi(const twi_package_t *twi);
```

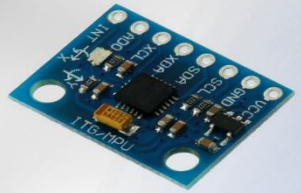


TWI (Two Wire Interface)

IMU konfigurieren

- Es werden 5 Bytes zur Konfiguration geschrieben: Die empfohlenen Einstellung stehen in Tabelle.
- Schreibe (write_to_twi) also 5x 1 Byte entsprechend der Tabelle.
- Beachte: buffer ist ein Pointer auf die unten stehenden Daten. Verwende auch einen Pointer!
- TWI Adresse: chip = **0x69** Alternativ: chip = 0x68
- Wie der Sensor sonst noch konfiguriert werden kann, steht im Datenblatt (Für Profis).

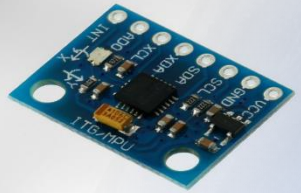
Konfigurations-Kommando (addr)	Konfigurations-Daten (buffer)	Funktion
0x6B	0x03	Power Mode (z-Referenz)
0x19	0x09	Samplerate 10ms
0x1A	0x04	1kHz DLPF mit 19ms Delay
0x1B	0x10	+/- 1000°/s
0x1C	0x08	+/- 4g



TWI (Two Wire Interface)

IMU auslesen

- Jeder der beiden Sensoren (Accelerometer und Gyroskop) verfügt über Register, in denen die Konfigurationsparameter und Messwerte stehen. Einige Register können verändert werden.
- Jeder der beiden Sensoren verfügt über 6 Bytes an Messwerten: Zwei Bytes bilden eine Messung. Es erfolgt eine Messung für jede der drei Raum-Achsen: x,y und z.
- Ein Paket kann wie folgt angelegt werden: twi_package_t packet_mpu6000_gyro;
- Diese sechs Bytes stehen in einem Register und müssen alle gemeinsam, im sogenannten Burst-Mode, ausgelesen werden. Der Burst-Mode wird immer dann automatisch aktiviert, wenn mehr als 1 Wert auf einmal ausgelesen wird. Dazu ist length auf 6 zu setzen: packet_mpu6000_gyro.lenght = 6;
- Erfolgt das Auslesen nicht im Burst-Mode, so können sich die Sensorwerte in der Zwischenzeit ändern. Dies führt zu erheblichen Fehlern aufgrund der Darstellung im Zweierkomplement.
- Die Register-Adresse (addr), aus denen jeweils die 6 Bytes ausgelesen werden, lauten:
  Accelerometer: 0x3B packet_mpu6000_acc.addr = 0x3B
  Gyroskop: 0x43 packet_mpu6000_gyro.addr = 0x43
- Es muss je ein Speicher (buffer) angelegt werden, wo die Messwerte abgelegt werden können, z.B. :
  char acc_messwerte[6];
- Der Buffer kann nun wie folgt in das TWI-Paket übernommen werden:
  packet_mpu6000_gyro.buffer = acc_messwerte;



Aufgaben

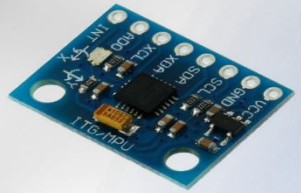
Verwende das EMQ Framework:

- Mitgeliefert als Library mit Open-Source-Teil
- Extra für das Quadrokopter-Labor entwickelt
- Enthält Treiber und Standard-Funktionen für
  - Hardware: CPU, IRQ, TC, Remote, ADC, USART, TWI, Delay, Motoren, LED
  - Software:
    - Grundlagen zum einfachen Starten
    - Funktionsvorlagen zum Ausfüllen
- Details siehe Softwaredokumentation: EMQ_Framework.pdf
- Das EMQ Framework verwendet kein Betriebssystem
- Der gesamte Code läuft in einer Endlosschleife (while)

Ein paar allgemeine Hinweise

EMQ Datentypen

- In der Datei EMQ_Interface_Data.h stehen einige, verfügbare EMQ Datentypen
- Die Datentypen mit dummy sollen später mit sinnvollen Inhalt gefüllt werden
- Alle anderen Datentypen **NICHT** modifizieren.
- Die Systemzeit steht in der Variable tc_ticks [ms].



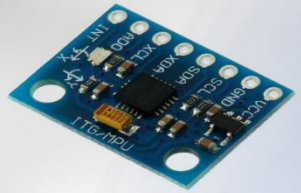
Aufgaben

Notwendige Hardware:

- EVK1100
- QCS / QCSF
- Mikro USB Kabel für Strom und zum Flashen
- USART Kabel für Kommunikation: RS232 auf PC (RS232 oder USB)

Notwendige Software:

- AVR Studio 32 installiert (mit Tool Chain und FLIP Treiber)
- EMQ Framework (Code)



Aufgaben

In dieser Übung wird das Modul IMU mit den beiden Dateien imu.c und imu.h erarbeitet.

Aufgabe 1: Überblick verschaffen

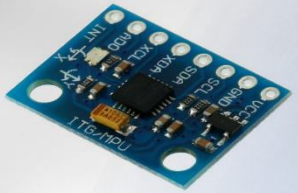
Machen Sie sich kurz mit dem EMQ Framework vertraut: Wichtig ist, die wesentlichen Quellcode-Dateien (MAIN, API, LIBRARY) zu verstehen. Auch die Programm-Struktur und der Programmfluss – insbesondere für das IMU-Modul - sollte prinzipiell nachvollzogen werden. Stellen Sie Fragen, wenn Ihnen etwas nicht klar ist.

Aufgabe 2: Sensoren konfigurieren

Konfigurieren Sie die beide Sensoren. Dazu sind 5 Bytes per Funktionsaufruf write_to_twi() zu schreiben, d.h. write_to_twi() ist 5 mal auszuführen: Stets natürlich mit unterschiedlichen Parametern, d.h. der Paket-Parameter wird modifiziert. Die Modifikationen des Parameters stehen in der Tabelle auf Folie 14. Zwischen jedem Paket soll 200ms gewartet werden, um dem Sensor Zeit zu geben, die Konfigurationen vorzunehmen: Dies erfolgt durch folgenden Funktionsaufruf: wait(200);

Füllen Sie dazu die Funktion **my_imu_init()** in **imu.c** aus. Verwenden Sie die Werte aus der Tabelle in Folie 14 sowie das Framework aus Folie 12 und 13.

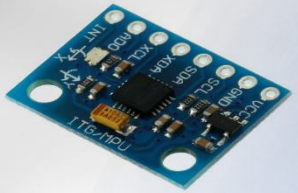
!!! Auf der folgenden Folie steht zum Nachsehen die Lösung für das erste Konfigurations-Byte !!!



Aufgaben

Wir konfigurieren die IMU - Hier wird Power Mode und z-Referenz eingestellt:

```
int status = 0;                                          // Unsere Variable zum Speichern von Fehlern.
twi_package_t packet_mpu6000;                            // Wir legen ein Packet an.

packet_mpu6000.chip = 0x69;                              // Wir legen die TWI Adresse fest: Folie 14
packet_mpu6000.addr_length = 1;                          // Wir legen die Adresse-Länge fest: Immer 1
packet_mpu6000.length = 1;                               // Wir wollen 1 Byte schreiben

packet_mpu6000.addr = 0x6B;                              // Das Kommando / Register, das wir brauchen: Folie 14, Zeile 1
const unsigned char setup_data_mpu_pm_[1] = { 0x03};     // Daten, die wir schreiben wollen: Folie 14, Zeile 1
                                                         // Das Framework erwartet einen Pointer, daher legen wir eine
                                                             Variable an.
packet_mpu6000.buffer = (void*) setup_data_mpu_pm_;      // Wir übergeben unsere Daten

status += write_to_twi(&packet_mpu6000);                 // Wir schreiben, d.h. senden das Paket.
                                                         // Der Rückgabewert ist 0, wenn alles ok. Falls ein Fehler beim
                                                             Schreiben auftritt, ist in status der Fehlercode.
wait(200); //Warte  kurz
```



Aufgaben

Aufgabe 3: Programmstruktur ergänzen

In jedem Zeitschritt sollen der Accelerometer und der Gyroskop Messwerte ausgeben (read) und diese konditioniert (cond) werden. Füllen Sie die Funktion **my_imu_read()**, die später von my_imu_run() alle 10ms aufgerufen werden soll und die Sensoren regelmäßig auslesen und konditionieren soll, soweit wie möglich. Ignorieren Sie zunächst den Parameter. Diese Funktion soll andere Funktionen des Moduls (imu.c) verwenden. Überlegen Sie sich, welche nötig sind.
Tipp: Es sind 4 Funktionsaufrufe nötig.

Aufgabe 4a: Sensoren auslesen

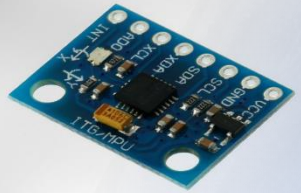
Nun sollen die Sensoren ausgelesen werden, indem Sie die Funktionen **my_acc_read_MPU6000()** und **my_gyro_read_MPU6000()** ausfüllen. Mit dem Kommando read_from_twi() können die 6 Bytes eines Sensors ausgelesen werden (vgl. Folie 15). Dazu müssen Sie ein TWI-Paket (Folie 12) zum Auslesen des Accelerometers anlegen und konfigurieren sowie eines für das Gyroskop.

Aufgabe 4b: Sensoren auslesen

Jeder Sensor liefert pro Achse zwei Bytes: Ein High- und ein Low-Byte. Die beiden Bytes müssen zu einer 2-Byte-Zahl (short) zusammengefasst werden. Dies erfolgt durch den Shift-Operator und den bitweisen Oder-Operator:

```
short acc_raw[3];                                          // Speicher für Rohwerte, global / permanent gespeichert
acc_raw[0] = acc_messwerte[1] | (acc_messwerte[0] << 8);   // Zwei gelesene Bytes zusammenfassen
```

Führen Sie diese Operation für beide Sensoren und für je alle drei Achsen aus. Geben Sie die Daten per USART aus.



Aufgaben

Aufgabe 5a: Sensoren konditionieren

Als nächstes sind die Sensoren zu konditionieren. Dies ist Teil der Kalibrierung. Konditionieren meint, dass die Rohdaten der Sensoren in korrekt skalierte Werte umgewandelt werden. Dies erfolgt mittels Division der Messung durch den Skalierungsfaktor. Die Skalierungsfaktoren lauten 8.192 für den Acceleromter und 32.8 für das Gyroskop.

Für die Konditionierung stehen zwei leere Funktionen (my_acc_cond(), my_gyro_cond()) bereit, für jeden Sensor eine, die genutzt werden sollen.

Bedenken Sie, dass für Divisionen die Gleitkommadarstellung benötigt wird. Daher sind die Messungen in double zu Type-Casten, zu dividieren und das Ergebnis ebenfalls in einem double zu speichern, wie folgend beschrieben:

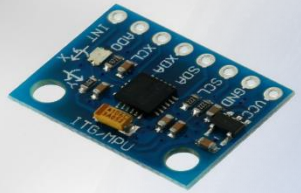
```
double acc_cond[3];                    // Konditionierte Werte, werden z.B. global / permanent gespeichert
acc_cond[0] = (double)acc_raw[0] / 8.192;
```

Aufgabe 5b: Sensoren konditionieren

Die konditionierten (z.B. globalen) Werte sollen an den Parameter der Funktion my_imu_read() übergeben werden:

```
imu_values->acc.x = acc_cond[0];
```

Nun liest die Funktion my_imu_read() Rohwerte aus, die bereits konditioniert / skaliert sind, d.h. teilweise kalibriert.



Aufgaben

Aufgabe 6: Sensoren kalibrieren

Kalibrieren Sie die Sensoren, wobei in einer For-Schleife zweitausend mal die Sensorwerte auszulesen sind und darüber der Mittelwert zu bilden ist. Dies soll in der Funktion **my_imu_calibrate()** erfolgen, wobei die Funktion my_imu_read() zum Auslesen der Sensoren zu verwenden ist. Damit der Gravitationsvektor dabei nicht herauskalibriert wird, ist am Ende auf das Bias der z-Achse der Wert 1000 zu addieren. Speichern Sie das ermittelte Bias (6 Werte: Ein Wert pro Sensor und Achse) in globalen Variablen als double.

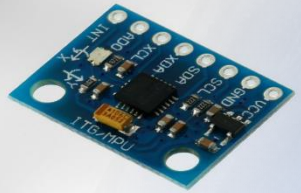
Aufgabe 7: Sensoren kalibrieren

Füllen Sie die Funktion **my_imu_run()**, so dass die Funktion Rohwerte mittels **my_imu_read(imu_values_raw)** ausliest und von den Rohwerten das Bias abzieht, um gültige, kalibrierte Messungen zu generieren. Dazu sind folgende Daten anzulegen:

```
imu_data_raw imu_values_raw;          // Für die Rohdaten aus my_imu_read
vector acc_values;                    // Für die kalibrierten Daten
```

Das Bias kann z.B. wie folgt abgezogen werden:

```
acc_values.x = imu_values_raw.acc.x - acc_bias[0];
```



Aufgaben

Aufgabe 8a: Sensorwerte weiterverarbeiten (Winkel)
Führen Sie in **my_imu_run()** die eindimensionale Integration für die Messungen des Gyroskops durch, indem Sie die drei Achsen unabhängig voneinander integrieren, d.h. aufaddieren und mit der Zeit skalieren:
imu_values->rpy.x = imu_values->rpy.x + gyro_values.x / 100.0;
Damit erhalten Sie einen Winkel. Bei einer 90° Drehung sollten schließlich auch 90° herauskommen.

Aufgabe 8b: Sensorwerte anzeigen (Winkel)
Im Display sollen die drei Winkel angezeigt werden. Modifizieren Sie dazu die renew_display() Funktion.

Aufgabe 8c: Sensorwerte weiterverarbeiten (Winkelgeschwindigkeit)
Zudem müssen wir für den Regler die Winkelgeschwindigkeiten an imu_values übergeben:
imu_values->rpy_angular_rate.x = gyro_values.x;