IMU WiFi-Sensorboard - BETA

Karsten Schäfer[1,3], Bernhardt Schäfer[1,2], Markus Mroch[1,3] und Philipp Schneider[1,2]

[1] *Universität Stuttgart*
[2] *Institut für Navigation*
[3] *Institut für Sportwissenschaften*

October 24, 2017

Das Sensorboard ermöglicht es Beschleunigungen und Drehraten des BMI055 IMU Sensors (3.2) kabellos, über ein WiFi-Netzwerk zu übertragen. Die Stromversorgung erfolgt durch einen Lithium-Polymer Akku und lässt so einen Messbetrieb über mehrere Stunden zu. Der Ladestand des Akkus kann ebenfalls ausgelesen und übertragen werden. Als *Microcontroler* und WiFi-Modul dient ein ESP-12F (3.1). Als Lade- und Programmierschnittstelle dient ein *Micro USB Anschluss* 2 LEDs können als Statusanzeige verwendet werden eine weitere dient als Ladeindikator.

In diesem Dokument soll gezeigt werden, wie Programme auf das Board mit Hilfe der Arduino IDE aufgespielt werden können. Ein Programmbeispiel 2 zur Übertragung der Messdaten an MATLAB liegt vor.

Contents

1 Aufbau

Das Sensorboard wurde entworfen um zwei verschiedene MEMS IMU Sensoren benutzen zu können. Die hier beschrieben Variante ist ausschließlich mit dem Bosch BMI055 [1] bestückt.

1.1 Übersicht

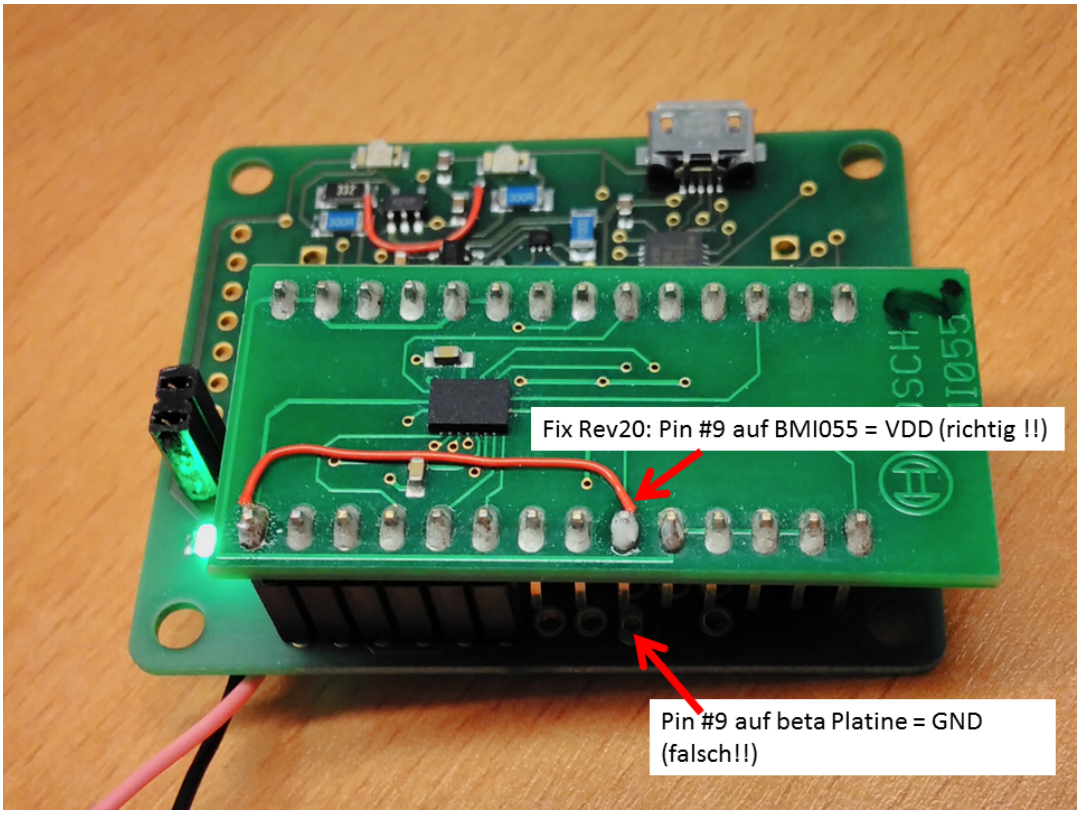

Abb. 1: *Übersicht mit nachträglichen Verbindungen*

1.2 Schaltplan

Abbildung 2 zeigt den Schaltplan des Sensorboards. Zu beachten ist, dass der Sensor *ICM2061* nicht bestückt ist.
I^2C **Betrieb** GPIO4 = SDA und GPIO5 =SCL

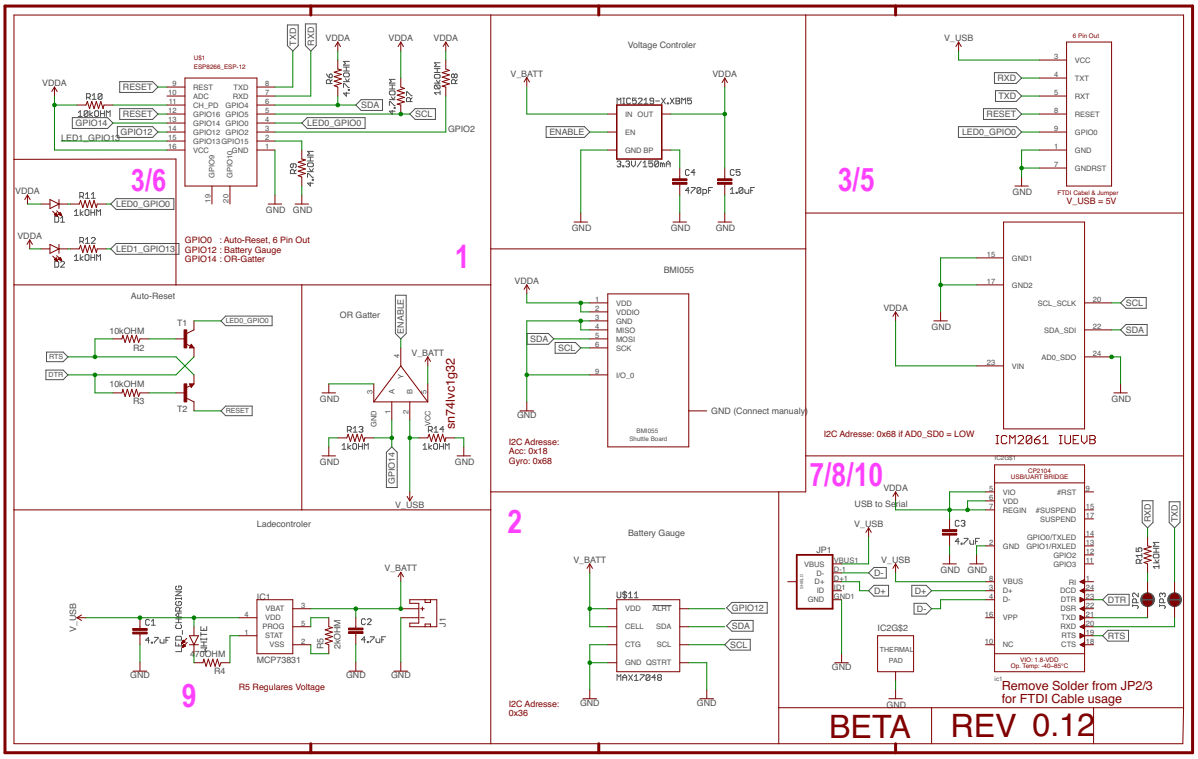

Abb. 2: *Schaltplan des Sensorboards (ICM2061 ist nicht bestückt)*

1.3 Platine

Abbildung 3 zeigt die Abmessungen und den Aufbau der Platine

	Bezeichnung
1	ESP-12F (3.1)
2	Fuelegauge (3.3)
3	LED D2 (3.4)
4	Reset (3.7)
5	FTDI Anschluss (3.6)
6	LED D1 (3.4)
7	Micro USB Anschluss
8	CP2104 (3.5)
9	LED D3 (3.4)
10	Solderbridge (3.5)

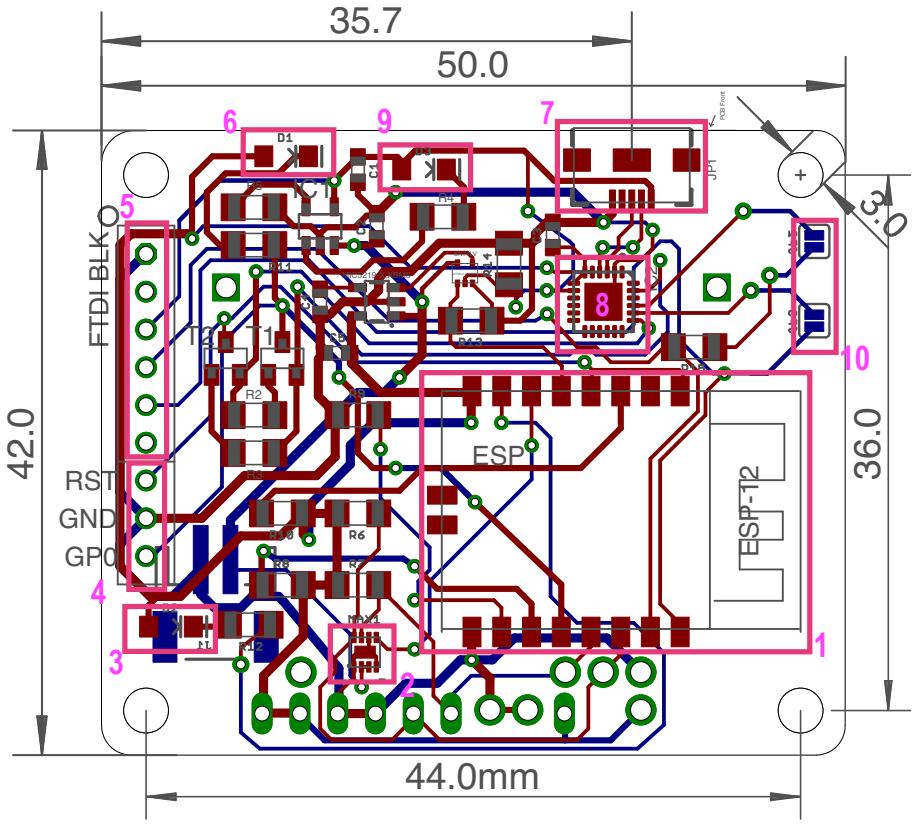

Abb. 3: *Platinen Layout ohne Sensor.* ROT *=Oben,* Blau *=Unten,* Grün *=Durchkontaktierung*

2 Beispielanwendung

Im Folgenden sollen ein paar Beispiel Anwendungen gezeigt werden. Um den Code auf das Sensorboard zu *Flashen* wird die Arduino IDE benötigt. Für Arduino Projekte gibt es eine große *Community*, sodass die meisten Probleme durch einer kurzen Internetrecherche lösbar sind.

2.1 LED Test

Der folgende Arduino Sketch testet die Funktion der LEDs. Die Pins GPIO0 GPIO2 und GPIO13 dienen hierbei als Sink für die LED, d.h. LOW = An.
GPI014 muss für den Batteriebetrieb HIGH sein.

```
#define LED0 0
#define LED1 2
#define LED2 13
#define GPIO14_OR 14


void setup() {
  pinMode(LED0, OUTPUT);
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  pinMode(GPIO14_OR, OUTPUT);
  digitalWrite(GPIO14_OR, HIGH);      // always on
  Serial.begin(115200);
}

// the loop function runs over and over again forever
void loop() {
  Serial.println(ESP.getChipId());
  Serial.println("all low");
  digitalWrite(LED0, LOW);
  digitalWrite(LED1, LOW);
  digitalWrite(LED2, LOW);
  delay(200); // wartet fuer 200 Millisekunden
  Serial.println("high 0");
  digitalWrite(LED0, HIGH);
  digitalWrite(LED1, LOW);
  digitalWrite(LED2, LOW);
  delay(200);
  Serial.println("high 1");
  digitalWrite(LED0, HIGH);
  digitalWrite(LED1, HIGH);
  digitalWrite(LED2, LOW);
  delay(200);
  Serial.println("high 2");
  digitalWrite(LED0, HIGH);
  digitalWrite(LED1, HIGH);
  digitalWrite(LED2, HIGH);
  delay(200);

}
```

2.2 I²C Scanner

Dieser Sketch kann verwendet werden um nach über den I^2C-Bus angeschlossenen Sensoren zu Suchen. Im Serial Monitor der ARDUINO IDE sollten folgende Adressen gefunden werden 0x18, 0x68 und 0x36 das sind Beschleunigungssensor & Gyroskop (3.2) und Fuelgauge (3.3).

```

    //
    // This sketch tests the standard 7-bit addresses
    // Devices with higher bit address might not be seen properly.
    //

    #include <Wire.h> // i2c libary


    void setup()
    {
```

```
12      Wire.begin(5,4); // i2c Bus :  GPIO5 =SCL und GPIO4 = SDA
13
14
15
16      Serial.begin(9600);           // Serial Monitor mit 9699 Baut Rate
17      while (!Serial);             // Leonardo: wait for serial monitor
18      Serial.println("\nI2C Scanner");
19    }
20
21
22    void loop()
23    {
24      byte error, address;
25      int nDevices;
26
27      Serial.println("Scanning...");
28
29      nDevices = 0;
30      for(address = 1; address < 127; address++ )
31      {
32        // The i2c_scanner uses the return value of
33        // the Write.endTransmisstion to see if
34        // a device did acknowledge to the address.
35        Wire.beginTransmission(address);
36        error = Wire.endTransmission();
37
38        if (error == 0)
39        {
40          Serial.print("I2C device found at address 0x");
41          if (address<16)
42            Serial.print("0");
43          Serial.print(address,HEX);
44          Serial.println("  !");
45
46          nDevices++;
47        }
48        else if (error==4)
49        {
50          Serial.print("Unknow error at address 0x");
51          if (address<16)
52            Serial.print("0");
53          Serial.println(address,HEX);
54        }
55      }
56      if (nDevices == 0)
57        Serial.println("No I2C devices found\n");
58      else
59        Serial.println("done\n");
60
61      delay(5000);           // wait 5 seconds for next scan
62    }
```

2.3 Übertragen des Signals via Wi-Fi

Eine Möglichkeit die Messdaten auf einen Computer mit Matlab zu übertragen soll hier gezeigt werden. Der Computer und der Sensor müssen imselben WiFi Netzwerk sein (z.B. WiFi-Hotspot mit Handy). Um den Sensor mit dem gewünschten Netz zu verbinden müssen im Sketch SSID und Passwort (Zeile 27-28) angepasst werden. Nach einem Reset wird über den Seriellen Monitor der Status der Verbindung angegeben. Ist der Sensor verbunden, leuchtet die grüne LED. Das Matlab file sollte die eingehenden UDP Verbindungen Verarbeiten.

2.3.1 Übertragungsstruktur

Jede Messung besteht aus 16 Byte. int32 (4 Byte) für die Zeit int16 (2 Byte) für die Beschleunigung jeder Achse und int16 (2 Byte) für die Drehrate jeder Achse.
Es werden immer 10 Messungen gemeinsam mit der Signalstärke des Wi-Fis (int32 4 Byte) und dem Ladezustand der Batterie (int8 1 Byte) übertragen, sodass das gesammte Packet $16 \cdot 10 + 5 = 165$ Byte lang ist.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...	161	162	163	164	165
t_1	t_2	t_3	t_4	ax_1	ax_2	ay_1	ay_2	az_1	az_2	ax_1	gx_2	gy_1	gy_2	gz_1	gz_2	...	$wifi_1$	$wifi_2$	$wifi_3$	$wifi_4$	bat

Die Werte werden im Matlab Skript (Zeile 40-50) zusammengesetzt.

2.3.2 Matlab Code

```
clear all
clear all
close all
% Variablen initialisieren und Fenster oeffnen
port=7000; %<--- Hier eigenen Port definiern (Wie in Arduino Skript)
figure
text(0.5,0.5,strcat('Waiting for input on Port:',num2str(port)));
drawnow
dataraw=zeros(1,165,'uint8');
messung=zeros(1000,7,'int32');
% UTP Port defenieren
udpr = dsp.UDPReceiver('LocalIPPort',port);
%Auf Daten von Port warten und holen, wenn vorhanden
while true
dataraw=zeros(1,165,'uint8');
while size(dataraw,1)<2
dataraw = udpr();
end
%Ladezustand und Signalstaerke entpacken
perc=dataraw(165);
wifi=typecast(uint8(dataraw(end-4:end-1)), 'int32');
%10 Messungen a 16byte =160x1 Vektoer in  10x16 Array umformen
data=reshape(dataraw(1:160),16,10)';
%Bytes wieder zu Werten zusammensetzen
for i=1:10;
time(i) = typecast(uint8(data(i,1:4)), 'int32');
acc(i,1) = typecast(uint8(data(i,5:6 )), 'int16');
acc(i,2) = typecast(uint8(data(i,7:8 )), 'int16');
acc(i,3) = typecast(uint8(data(i,9:10)), 'int16');
gyro(i,1) = typecast(uint8(data(i,11:12)), 'int16');
gyro(i,2) = typecast(uint8(data(i,13:14)), 'int16');
gyro(i,3) = typecast(uint8(data(i,15:16)), 'int16');
end
%Die aeltesten 10 Messungen loeschen und die Neuen 10 hinzufuegen
messung(1:10,:)=[];
messung(end+1:end+10,:)=[time' acc gyro];
%% Darstellung
%Beschleunigung
subplot(3,1,1)
title('Beschleunigung')
hold on
    p1=plot(messung(:,1),messung(:,2),'r');
    p2=plot(messung(:,1),messung(:,3),'g');
    p3=plot(messung(:,1),messung(:,4),'b');
axis([messung(1,1), messung(end,1), -5000 5000]);
%Drehraten
subplot(3,1,2)
title('Drehraten')
hold on
    p1=plot(messung(:,1),messung(:,5),'r');
    p2=plot(messung(:,1),messung(:,6),'g');
    p3=plot(messung(:,1),messung(:,7),'b');
axis([messung(1,1), messung(end,1), -5000 5000]);
%Wifi und Signalstaerke
subplot(3,1,3)
title('Telemetrie')
hold on
b1=bar([1 2], [2*(wifi+100)  perc]);
xticklabels({'wifi','battery'})
ylabel('%')
%Zeichnen der Plots
drawnow
    delete(p1);
    delete(p2);
    delete(p3);
    delete(b1);
end
```

2.3.3 Arduino Sketch

Dieser Sketch

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>
#include <stdio.h>

#include "Wire.h"
#include "I2Cdev.h"
#include "MAX17048.h"
#include "BMI055.h"
// BMI055 default I2C address is 0x18 and 0x68
BMI055 imu;
// MAX17048 default I2C address is 0x36
```

```
12 MAX17048 lipo;

14 WiFiUDP Udp;

16 #define LED_R    0        // red
17 #define LED_G    13       // green
18 #define LED_B    2        // blue onboard ESP8266-12F
19 #define ON_PIN   14       // GPIO14 muss high, sonst Notaus

21 #define SSID   "iPippo"
22 #define PW     "hallowelt"
23 #define TARGET_IP "172.20.10.4"
24 #define TARGET_PORT 7000
25 #define MESSAGE_SIZE 255

27 uint8_t data[32000];
28 uint16_t counter = 0;

30 bool error = false;
31 uint32_t errorT;

33 union ByteSplit2 {
34   int16_t in;
35   struct S {
36     byte b1, b2;
37   } out;
38 };
39 union ByteSplit4 {
40   uint32_t in;
41   struct S {
42     byte b1, b2, b3, b4;
43   } out;
44 };

46 void setup() {
47   Wire.begin(5, 4);
48   Serial.begin(115200);
49   Serial.println("booting...");
50   Serial.print("Connecting to "SSID);
51   WiFi.begin(SSID, PW);
52   while (WiFi.status() != WL_CONNECTED) {
53     delay(500);
54     Serial.print(".");
55   }
56   Serial.println();
57   Serial.print("Connected, IP address: ");
58   Serial.println(WiFi.localIP());

60   imu.initialize();

62   if (!imu.testConnection()) {
63     Serial.println("IMU test failed");
64     delay(100);
65     ESP.restart();
66   }

68   // Config Bandwidth and Range has to be adjusted to your expected measurments
69   imu.setRangeAcc(BMI055_ACC_RANGE_16G);
70   imu.setRangeGyr(BMI055_GYR_RANGE_2000);

72   imu.setBandwidthAcc(BMI055_ACC_BW_500);
73   imu.setBandwidthGyr(BMI055_GYR_BW_523);

75   lipo.initialize();

77   pinMode(LED_R, OUTPUT);
78   pinMode(LED_G, OUTPUT);
79   pinMode(LED_B, OUTPUT);
80   pinMode(ON_PIN, OUTPUT);
81   digitalWrite(LED_R, HIGH);
82   digitalWrite(LED_G, LOW);
83   digitalWrite(LED_B, HIGH);
84   digitalWrite(ON_PIN, HIGH);
85 }
```

```
87  void loop() {
88    ByteSplit4 t;
89    t.in = (uint32_t)millis();
90    //  t.in = 12313;
91    data[counter++] = t.out.b1;
92    data[counter++] = t.out.b2;
93    data[counter++] = t.out.b3;
94    data[counter++] = t.out.b4;

96    imu.getAcceleration(&data[counter++], &data[counter++], &data[counter++], &data[counter++], &
        data[counter++], &data[counter++]);


99    imu.getTurnRate(&data[counter++], &data[counter++], &data[counter++], &data[counter++], &data[
        counter++], &data[counter++]);

101   if (counter >= 160) {//160 equals 10x ( 2byte*6sensors + 4byte time )
102     sendBuffer();
103     counter = 0;
104   }
105 }

107 bool sendBuffer() {
108   int8_t perc = (int)(lipo.getSOC()    * MAX17048_SOC_SCALE);
109   uint8_t vbatt = (int)(lipo.getVCELL() * MAX17048_VCELL_SCALE * 10);
110   bool okay = true;

112   ByteSplit4 s;
113   s.in = WiFi.RSSI();
114   data[counter++] = s.out.b1;
115   data[counter++] = s.out.b2;
116   data[counter++] = s.out.b3;
117   data[counter++] = s.out.b4;
118   data[counter++] = perc;

120   for ( uint8_t iTry = 0; iTry < 10; iTry++) {
121     okay = sendUDP(data, counter, TARGET_PORT);
122     if (okay == true)
123       break;
124     delay(1);
125   }
126   if (okay == false) {
127     delay(0);
128     byte msg[32] = "finnaly failed to send message\n";
129     sendUDP(msg, 32, 7001);
130     Serial.println("finnaly failed to send message\n");
131     digitalWrite(LED_R, LOW);
132     error = true;
133     errorT = (uint32_t)millis();
134     delay(0);
135   }
136   else if (error == true && ((uint32_t)millis() - errorT) >= 1000) {
137     byte msg[50] = "finnaly failed to send message\n";
138     sendUDP(msg, 32, 7001);
139     error = false;
140     digitalWrite(LED_R, HIGH);
141   }
142   counter = 0;
143 }

145 bool sendUDP(byte* test, uint8_t msgSize, uint16_t port) {
146   if ( Udp.beginPacket(TARGET_IP, port) != 1 ) {
147     return false;
148   }
149   Udp.write(test, msgSize);
150   if ( Udp.endPacket() != 1) {
151     return false;
152   }
153   return true;
154 }
```

3 Komponenten

3.1 ESP-12F

Das ESP8266 Modul ist ein Ultra-low-Power-32-Bit-Mikrocontroller. Integriertes WLAN ermöglicht die Kommunikation nach Außen. Die Programmierung kann z.B. über die ARDUINO-IDE vorgenommen werden. Die Sensoren (IMU, Batteriestand) sind über einen I^2C-Bus angeschlossen.

WiFi Protocles 802.11 b/g/n
Frequency Range 2.4GHz-2.5GHz (2400M-2483.5M)
Security WPA/WPA2 WEP/TKIP/AES
Stromverbrauch 80 mA
Network Protocols IPv4, TCP/UDP/HTTP/FTP

3.2 BMI055

Der BOSCH BMI055 ist eine Inertiale Messeinheit (IMU) und somit eine räumliche Kombination von Beschleunigungssensoren und Drehratensensoren. Die Messungen erfolgen in x-, y- und z- Richtung und realisieren so 6 Freiheitsgrade (6DoF).

Accelerometer

I2C Adresse 0x18
Auflösung 12bit
Messbereich Programmierbar: ±2g/±4g/±8g/±16g
Stromverbrauch 130 μA

Gyroskop

I2C Adresse 0x68
Auflösung 16bit
Messbereich Programmierbar: ±125°/s to ±2000°/s
Stromverbrauch 5mA

3.3 Fuelgauge

Der Batteriestandsanzeiger MAX17048 bietet die Möglichkeit den Ladestand eines LiPo-Akkus (1 Zelle, 3.7V) in Prozent abzufragen. Interne wird die Spannung in eine einem Model in Prozent umgerechnet, dabei werden spezifische Effekte der Batterie, wie Temperatur und Spannungskurve berücksichtigt.

I2C Adresse 0x36
Precision ±7.5mV
Stromverbrauch <23μA

3.4 LEDs

Das Board verfügt über 3 LEDs von denen 2 über die GPIOs des ESP8266 Moduls frei programmierbar sind. Eine LED ist als Ladeindikator vorgesehen.

Table 1: *LEDs*

Name	Anschluss	Farbe
D1	GPIO0	???
D2	GPIO13	???
D3	Ladestand	???

LED D1 blinkt wenn Kommunikation über die UART Schnittstelle stattfindet.

3.5 CP2104

CP2104 Chip als USB/UART Bridge für die Programmierung. Treiber sind für alle gängigen Betriebssystem vorhanden. RXD und TXD können ggf. manuell getrennt werden (Siehe *Solderbridge* Abb. 3).

3.6 FTDI Anschluss

Anschluss für 5 Volt FTDI Kabel (GND, CTS, VCC, TXD, RXD, RTS). Kann auch zum laden des Akkus verwendet werden.

3.7 Reset

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

4 Anhang

4.1 Bibliotheken

4.1.1 BMI055

```
1  // BMI055 I2C Class source
2  // 2013/11/05 by Bernhardt Schaefer, boen82@gmail.com
3  // Datasheet: http://www.bosch-sensortec.com/en/homepage/products_3/6_axis_sensors_2/
       inertial_measurement_unit_1/bmi055_1/bmi055
4  // http://ae-bst.resource.bosch.com/media/products/dokumente/bmi055/BST-BMI055-DS000-06.pdf
5  // Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
6  //
7  // Changelog:
8  //     2013/11/05 - initial release

10 /* ============================================
11 I2Cdev device library code is placed under the MIT license
12 Copyright (c) 2011 Jeff Rowberg

14 Permission is hereby granted, free of charge, to any person obtaining a copy
15 of this software and associated documentation files (the "Software"), to deal
16 in the Software without restriction, including without limitation the rights
17 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
18 copies of the Software, and to permit persons to whom the Software is
19 furnished to do so, subject to the following conditions:

21 The above copyright notice and this permission notice shall be included in
22 all copies or substantial portions of the Software.

24 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
25 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
26 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
27 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
28 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
29 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
30 THE SOFTWARE.
31 ===============================================
32 */

34 #include "BMI055.h"

36 /*
37  Default Constructor
38  */
39 BMI055::BMI055() {
40   devAddrAcc  = BMI055_ACC_ADDRESS;
```

```
41    devAddrGyr  = BMI055_GYR_ADDRESS;
42  }
43
44  /*
45   Specific Address Constructor
46   */
47  BMI055::BMI055(uint8_t addressAcc, uint8_t addressGyro) {
48    devAddrAcc  = addressAcc;
49    devAddrGyr  = addressGyro;
50  }
51
52  /*
53  Initialize with max Range for Acc and Gyro
54   */
55  void BMI055::initialize() {
56    setRangeAcc(BMI055_ACC_RANGE_16G);
57    setRangeGyr(BMI055_GYR_RANGE_2000);
58    // setBandwidth();
59  }
60
61  /*
62   Verify the I2C connection.
63   Make sure the device is connected and responds as expected.
64   True if connection is valid, false otherwise.
65   This registers are used to verify the identity of the device
66   BMI055_ACC_BGW_CHIPID    = 0xFA
67   BMI055_GYR_CHIP_ID   = 0x0F
68   */
69  bool BMI055::testConnection() {
70    return ((getDeviceIDAcc() == 0xFA) && (getDeviceIDGyr() == 0x0F));
71  }
72
73  uint8_t BMI055::getDeviceIDAcc() {
74    I2Cdev::readByte(devAddrAcc, BMI055_ACC_BGW_CHIPID, buffer);
75    return buffer[0];
76  }
77
78  uint8_t BMI055::getDeviceIDGyr() {
79    I2Cdev::readByte(devAddrGyr, BMI055_GYR_CHIP_ID, buffer);
80    return buffer[0];
81  }
82
83  /*
84  Set Acc Full Range
85   0011 = +-2g
86   0101 = +-4g
87   1000 = +-8g
88   1100 = +-16g
89   */
90  void BMI055::setRangeAcc(uint8_t range) {
91    I2Cdev::writeBits(devAddrAcc, BMI055_ACC_PMU_RANGE, BMI055_ACC_RANGE_BIT,
        BMI055_ACC_RANGE_LENGTH, range);
92  }
93
94  /*
95  Set Gyr Full Range
96   000 = +-2000Â°/s
97   001 = +-1000Â°/s
98   010 = +- 500Â°/s
99   011 = +- 250Â°/s
100  100 = +- 125Â°/s
101  */
102 void BMI055::setRangeGyr(uint8_t range) {
103   I2Cdev::writeBits(devAddrGyr, BMI055_GYR_RANGE, BMI055_GYR_RANGE_BIT, BMI055_GYR_RANGE_LENGTH,
        range);
104 }
105
106 void BMI055::setBandwidthAcc(uint8_t bandwidth) {
107   I2Cdev::writeBits(devAddrAcc, BMI055_ACC_PMU_BW, BMI055_ACC_BW_BIT, BMI055_ACC_BW_LENGTH,
        bandwidth);
108 }
109
110 void BMI055::setBandwidthGyr(uint8_t bandwidth) {
```

```
111   I2Cdev::writeBits(devAddrGyr, BMI055_GYR_BW, BMI055_GYR_BW_BIT, BMI055_GYR_BW_LENGTH, bandwidth
        );
112 }

114 uint8_t BMI055::getBandwidthAcc() {
115   I2Cdev::readBits(devAddrAcc, BMI055_ACC_PMU_BW, BMI055_ACC_BW_BIT, BMI055_ACC_BW_LENGTH, buffer
        );
116   return buffer[0];
117 }

119 uint8_t BMI055::getBandwidthGyr() {
120   I2Cdev::readBits(devAddrGyr, BMI055_GYR_BW, BMI055_GYR_BW_BIT, BMI055_GYR_BW_LENGTH, buffer);
121   return buffer[0];
122 }

124 uint8_t BMI055::getRangeAcc() {
125   I2Cdev::readBits(devAddrAcc, BMI055_ACC_PMU_RANGE, BMI055_ACC_RANGE_BIT,
        BMI055_ACC_RANGE_LENGTH, buffer);
126   return buffer[0];
127 }

129 uint8_t BMI055::getRangeGyr() {
130   I2Cdev::readBits(devAddrGyr, BMI055_GYR_RANGE, BMI055_GYR_RANGE_BIT, BMI055_GYR_RANGE_LENGTH,
        buffer);
131   return buffer[0];
132 }

134 void BMI055::getAcceleration(uint8_t* axLB, uint8_t* axHB, uint8_t* ayLB, uint8_t* ayHB, uint8_t*
        azLB, uint8_t* azHB) {
135   I2Cdev::readBytes(devAddrAcc, BMI055_ACC_ACCD_X_LSB, 6, buffer);
136   *axLB=(uint8_t)buffer[0];
137   *axHB=(uint8_t)buffer[1];

139   *ayLB=(uint8_t)buffer[2];
140   *ayHB=(uint8_t)buffer[3];

142   *azLB=(uint8_t)buffer[4];
143   *azHB=(uint8_t)buffer[5];

145 }

147 // // TODO: mask out unused bits? signed integer behaves strange when bitshifted: [1111 1111 1111
        0001] >>4 = [1111 1111 1111 1111]
148 //
149 // int16_t BMI055::getAccelerationX() {
150 //   I2Cdev::readBytes(devAddrAcc, BMI055_ACC_ACCD_X_LSB, 2, buffer);
151 //   return ((((int16_t)buffer[1]) << 8) | (int16_t)buffer[0]  >> 4);
152 // }
153 //
154 // int16_t BMI055::getAccelerationY() {
155 //   I2Cdev::readBytes(devAddrAcc, BMI055_ACC_ACCD_Y_LSB, 2, buffer);
156 //   return ((((int16_t)buffer[1]) << 8) | (int16_t)buffer[0]  >> 4);
157 // }
158 //
159 // int16_t BMI055::getAccelerationZ() {
160 //   I2Cdev::readBytes(devAddrAcc, BMI055_ACC_ACCD_Z_LSB, 2, buffer);
161 //   return ((((int16_t)buffer[1]) << 8) | (int16_t)buffer[0]  >> 4);
162 // }

164 void BMI055::getTurnRate(uint8_t* gxLB,uint8_t* gxHB, uint8_t* gyLB,uint8_t* gyHB, uint8_t* gzLB,
        uint8_t* gzHB) {
165   I2Cdev::readBytes(devAddrGyr, BMI055_GYR_RATE_X_LSB, 6, buffer);
166   *gxLB=(uint8_t)buffer[0];
167   *gxHB=(uint8_t)buffer[1];

169   *gyLB=(uint8_t)buffer[2];
170   *gyHB=(int8_t)buffer[3];

172   *gzLB=(uint8_t)buffer[4];
173   *gzHB=(uint8_t)buffer[5];
174 }

176 // int16_t BMI055::getTurnRateX() {
177 //   I2Cdev::readBytes(devAddrGyr, BMI055_GYR_RATE_X_LSB, 2, buffer);
```

```
178 //    return ((((int16_t)buffer[1]) << 8) | buffer[0]);
179 // }
180 //
181 // int16_t BMI055::getTurnRateY() {
182 //    I2Cdev::readBytes(devAddrGyr, BMI055_GYR_RATE_Y_LSB, 2, buffer);
183 //    return ((((int16_t)buffer[1]) << 8) | buffer[0]);
184 // }
185 //
186 // int16_t BMI055::getTurnRateZ() {
187 //    I2Cdev::readBytes(devAddrGyr, BMI055_GYR_RATE_Z_LSB, 2, buffer);
188 //    return ((((int16_t)buffer[1]) << 8) | buffer[0]);
189 // }
```

4.1.2 BMI055.h

```
1  // BMI055 I2C Class header
2  // 2013/11/05 by Bernhardt Schaefer, boen82@gmail.com
3  // Datasheet: http://www.bosch-sensortec.com/en/homepage/products_3/6_axis_sensors_2/
       inertial_measurement_unit_1/bmi055_1/bmi055
4  // http://ae-bst.resource.bosch.com/media/products/dokumente/bmi055/BST-BMI055-DS000-06.pdf
5  // Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
6  //
7  // Changelog:
8  //     2013/11/05 - initial release

10 /* ============================================
11 I2Cdev device library code is placed under the MIT license
12 Copyright (c) 2011 Jeff Rowberg

14 Permission is hereby granted, free of charge, to any person obtaining a copy
15 of this software and associated documentation files (the "Software"), to deal
16 in the Software without restriction, including without limitation the rights
17 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
18 copies of the Software, and to permit persons to whom the Software is
19 furnished to do so, subject to the following conditions:

21 The above copyright notice and this permission notice shall be included in
22 all copies or substantial portions of the Software.

24 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
25 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
26 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
27 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
28 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
29 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
30 THE SOFTWARE.
31 ===============================================
32 */

34 #ifndef _BMI055_H_
35 #define _BMI055_H_

37 #include "I2Cdev.h"

39 #define BMI055_ACC_ADDRESS                  0x18    // default I2C address acc
40 #define BMI055_GYR_ADDRESS          0x68    // default I2C address gyro

42 #define BMI055_ACC_ADDRESS_alt              0x19    // alternative I2C address acc
43 #define BMI055_GYR_ADDRESS_alt              0x69    // alternative I2C address gyro

45 #define BMI055_ACC_BGW_CHIPID             0x00  // Fixed value B11111010 or 0xFA

47 #define BMI055_ACC_ACCD_X_LSB       0x02
48 #define BMI055_ACC_ACCD_X_MSB     0x03
49 #define BMI055_ACC_ACCD_Y_LSB       0x04
50 #define BMI055_ACC_ACCD_Y_MSB     0x05
51 #define BMI055_ACC_ACCD_Z_LSB       0x06
52 #define BMI055_ACC_ACCD_Z_MSB     0x07
53 #define BMI055_ACC_ACCD_TEMP      0x08
54 #define BMI055_ACC_INT_STATUS_0     0x09
55 #define BMI055_ACC_INT_STATUS_1     0x0A
56 #define BMI055_ACC_INT_STAUTS_2     0x0B
57 #define BMI055_ACC_INT_STATUS_3     0x0C
```

```
59 #define BMI055_ACC_FOFO_STATUS        0x0E
60 #define BMI055_ACC_PMU_RANGE        0x0F
61 #define BMI055_ACC_PMU_BW       0x10
62 #define BMI055_ACC_PMU_LPW        0x11
63 #define BMI055_ACC_PMU_LOW_POWER      0x12
64 #define BMI055_ACC_ACCD_HBW       0x13
65 #define BMI055_ACC_BGW_SOFTRESET      0x14

67 #define BMI055_ACC_INT_EN_0       0x16
68 #define BMI055_ACC_INT_EN_1       0x17
69 #define BMI055_ACC_INT_EN_2       0x18
70 #define BMI055_ACC_INT_MAP_0        0x19
71 #define BMI055_ACC_INT_MAP_1        0x1A
72 #define BMI055_ACC_INT_MAP_2        0x1B
73 #define BMI055_ACC_INT_SRC        0x1E
74 #define BMI055_ACC_INT_OUT_CTRL       0x20
75 #define BMI055_ACC_INT_RST_LATCH      0x21
76 #define BMI055_ACC_INT_0        0x22
77 #define BMI055_ACC_INT_1        0x23
78 #define BMI055_ACC_INT_2        0x24
79 #define BMI055_ACC_INT_3        0x25
80 #define BMI055_ACC_INT_4        0x26
81 #define BMI055_ACC_INT_5        0x27
82 #define BMI055_ACC_INT_6        0x28
83 #define BMI055_ACC_INT_7        0x29
84 #define BMI055_ACC_INT_8        0x2A
85 #define BMI055_ACC_INT_9        0x2B
86 #define BMI055_ACC_INT_A        0x2C
87 #define BMI055_ACC_INT_B        0x2D
88 #define BMI055_ACC_INT_C        0x2E
89 #define BMI055_ACC_INT_D        0x2F
90 #define BMI055_ACC_FIFO_CONFIG_0      0x30
91 #define BMI055_ACC_PMU_SELF_TEST      0x32
92 #define BMI055_ACC_TRIM_NVM_CTRL      0x33
93 #define BMI055_ACC_BGW_SPI3_WDT       0x34
94 #define BMI055_ACC_OFC_CTRL       0x36
95 #define BMI055_ACC_OFC_SETTING        0x37
96 #define BMI055_ACC_OFC_OFFSET_X       0x38
97 #define BMI055_ACC_OFC_OFFSET_Y       0x39
98 #define BMI055_ACC_OFC_OFFSET_Z       0x3A
99 #define BMI055_ACC_TRIM_GP0       0x3B
100 #define BMI055_ACC_TRIM_GP1       0x3C
101 #define BMI055_ACC_FIFO_CONFIG_1      0x3E
102 #define BMI055_ACC_FIFO_DATA        0x3F

104 #define BMI055_ACC_RANGE_BIT        3
105 #define BMI055_ACC_RANGE_LENGTH       4
106 #define BMI055_ACC_BW_BIT       4
107 #define BMI055_ACC_BW_LENGTH        5

109 #define BMI055_ACC_RANGE_2G       3    // B0011
110 #define BMI055_ACC_RANGE_4G       5    // B0101
111 #define BMI055_ACC_RANGE_8G       8    // B1000
112 #define BMI055_ACC_RANGE_16G        12     // B1100

114 #define BMI055_ACC_BW_BIT                       4
115 #define BMI055_ACC_BW_LENGTH                    5

117 #define BMI055_ACC_BW_7                         8               // 01000b'7.81 Hz
118 #define BMI055_ACC_BW_15                        9               //'01001b 15.63 Hz
119 #define BMI055_ACC_BW_31                        10              // 01010b 31.25 Hz
120 #define BMI055_ACC_BW_62                        11              //'01011b 62.5 Hz
121 #define BMI055_ACC_BW_125                       12              //'01100b 125 Hz
122 #define BMI055_ACC_BW_250                       13              //'01101b 250 Hz
123 #define BMI055_ACC_BW_500                       14              //'01110b 500 Hz
124 #define BMI055_ACC_BW_1000                      15              //'01111b'1000 Hz


127 #define BMI055_ACC_DATA_HIGH_BW_BIT     7
128 #define BMI055_ACC_DATA_HIGH_BW_LENGTH      1
129 #define BMI055_ACC_DATA_HIGH_BW                 1

131 // GYRO REGISTER
```

```
133 #define BMI055_GYR_CHIP_ID       0x00  // fixed value 0x0F
134 #define BMI055_GYR_RATE_X_LSB      0x02
135 #define BMI055_GYR_RATE_X_MSB      0x03
136 #define BMI055_GYR_RATE_Y_LSB      0x04
137 #define BMI055_GYR_RATE_Y_MSB      0x05
138 #define BMI055_GYR_RATE_Z_LSB      0x06
139 #define BMI055_GYR_RATE_Z_MSB      0x07

141 #define BMI055_GYR_INT_STATUS_0      0x09
142 #define BMI055_GYR_INT_STATUS_1      0x0A
143 #define BMI055_GYR_INT_STAUTS_2      0x0B
144 #define BMI055_GYR_INT_STATUS_3      0x0C

146 #define BMI055_GYR_FIFO_STATUS       0x0E
147 #define BMI055_GYR_RANGE        0x0F
148 #define BMI055_GYR_BW         0x10
149 #define BMI055_GYR_LPM1                          0x11
150 #define BMI055_GYR_LPM2                          0x12
151 #define BMI055_GYR_RATE_HBW                      0x13
152 #define BMI055_GYR_BGW_SOFTRESET                 0x14
153 #define BMI055_GYR_INT_EN_0                      0x15
154 #define BMI055_GYR_INT_EN_1                      0x16
155 #define BMI055_GYR_INT_MAP_0                     0x17
156 #define BMI055_GYR_INT_MAP_1                     0x18
157 #define BMI055_GYR_INT_MAP_2                     0x19
158 #define BMI055_GYR_INT_DATA_SOURCE               0x1A
159 #define BMI055_GYR_MOT_THRES_DATA_SOURCE         0x1B
160 #define BMI055_GYR_MOT_INT_EN                    0x1C

162 #define BMI055_GYR_FIFO_WM_EN                    0x1E

164 #define BMI055_GYR_INT_RST_LATCH                 0x21
165 #define BMI055_GYR_HIGH_TH_X                     0x22
166 #define BMI055_GYR_HIGH_DUR_X                    0x23
167 #define BMI055_GYR_HIGH_TH_Y                     0x24
168 #define BMI055_GYR_HIGH_DUR_Y                    0x25
169 #define BMI055_GYR_HIGH_TH_Z                     0x26
170 #define BMI055_GYR_HIGH_DUR_Z                    0x27

172 #define BMI055_GYR_SOC                           0x31
173 #define BMI055_GYR_A_FOC                         0x32
174 #define BMI055_GYR_TRIM_NVM_CTRL                 0x33
175 #define BMI055_GYR_BGW_SPI3_WDT                  0x34

177 #define BMI055_GYR_OFC1                          0x36
178 #define BMI055_GYR_OFC2                          0x37
179 #define BMI055_GYR_OFC3                          0x38
180 #define BMI055_GYR_OFC4                          0x39
181 #define BMI055_GYR_TRIM_GP0                      0x3A
182 #define BMI055_GYR_TRIM_GP1                      0x3B
183 #define BMI055_GYR_BIST                          0x3C
184 #define BMI055_GYR_FIFO_CONFIG_0                 0x3D
185 #define BMI055_GYR_FIFO_CONFIG_1                 0x3E
186 #define BMI055_GYR_FIFO_DATA                     0x3F

188 // GYR RANGE

190 #define BMI055_GYR_RANGE_BIT       2
191 #define BMI055_GYR_RANGE_LENGTH      3

193 #define BMI055_GYR_RANGE_2000      0   // B000
194 #define BMI055_GYR_RANGE_1000      1   // B001
195 #define BMI055_GYR_RANGE_500       2   // B010
196 #define BMI055_GYR_RANGE_250       3   // B011
197 #define BMI055_GYR_RANGE_125       4   // B100

199 // GYR BW
200 #define BMI055_GYR_BW_BIT                        3
201 #define BMI055_GYR_BW_LENGTH                     4

203 #define BMI055_GYR_BW_32                         7                //  0111   20 100 Hz BW 32 Hz
204 #define BMI055_GYR_BW_64                         6                // â 0110   10 200 Hz BW 64 Hz
205 #define BMI055_GYR_BW_12                         5                // â 0101   20 100 Hz BW 12 Hz
206 #define BMI055_GYR_BW_23                         4                // â 0100   10 200 Hz BW 23 Hz
```

```
207 #define BMI055_GYR_BW_47                    3           // â 0011  5  400 Hz BW 47 Hz
208 #define BMI055_GYR_BW_116                   2           // â 0010  2 1000 Hz BW 116 Hz
209 #define BMI055_GYR_BW_230                   1           // â 0001  0 2000 Hz BW 230 Hz
210 #define BMI055_GYR_BW_523                   0           // â 0000  0 2000 Hz Unfiltered
      (523Hz)


213 #define BMI055_GYR_BW_BIT                   3
214 #define BMI055_GYR_BW_LENGTH                4
215 #define BMI055_GYR_BW_32HZ                  7           // B0111
216 #define BMI055_GYR_BW_64HZ                  6           // B0111
217 #define BMI055_GYR_BW_12HZ                  5           // B0110
218 #define BMI055_GYR_BW_23HZ                  4           // B0101
219 #define BMI055_GYR_BW_47HZ                  3           // B0011
220 #define BMI055_GYR_BW_116HZ                 2           // B0010
221 #define BMI055_GYR_BW_230HZ                 1           // B0001
222 #define BMI055_GYR_BW_523HZ                 0           // B0000

224 // TODO

226 // class

228 class BMI055 {
229 public:
230   BMI055();
231   BMI055(uint8_t addressAcc, uint8_t addressGyro);

233   void initialize();
234   bool testConnection();

236   // CHIP_ID register
237   uint8_t getDeviceIDAcc();
238   uint8_t getDeviceIDGyr();

240   // AXIS registers
241   void getAcceleration(uint8_t* ,uint8_t* , uint8_t* ,uint8_t* , uint8_t* ,uint8_t* );
242   int16_t getAccelerationX();
243   int16_t getAccelerationY();
244   int16_t getAccelerationZ();

246   void getTurnRate(uint8_t* ,uint8_t* , uint8_t* ,uint8_t* , uint8_t* ,uint8_t* );
247   int16_t getTurnRateX();
248   int16_t getTurnRateY();
249   int16_t getTurnRateZ();

251   // Range & Bandwidth Registers
252   uint8_t getRangeAcc();
253   void setRangeAcc(uint8_t range);
254   uint8_t getBandwidthAcc();
255   void setBandwidthAcc(uint8_t bandwidth);
256   uint8_t getRangeGyr();
257   void setRangeGyr(uint8_t range);
258   uint8_t getBandwidthGyr();
259   void setBandwidthGyr(uint8_t bandwidth);

261 private:
262   uint8_t devAddrAcc;
263   uint8_t devAddrGyr;
264   uint8_t buffer[6];
265 };

267 #endif /* _BMI055_H_ */
```

4.1.3 I2Cdev

https://github.com/jrowberg/i2cdevlib/blob/master/Arduino/I2Cdev/I2Cdev.h

4.1.4 MAX17048

https://github.com/libpropeller/libpropeller/blob/master/libpropeller/max17048/max17048.h

Datenblätter

[1] *Bosch Sensortec shuttle-board BMI055 is a PCB with a BMI055 Inertial Measurement Unit (IMU) mounted on it.*

[2] *ESP-12F WiFi Module*.