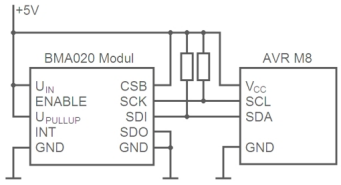
{jcomments on}

Bei ELV gibt es seit einiger Zeit ein 3-Achsen Beschleunigungssensormodul basierend auf dem BMA020 von Bosch Sensortec. Der
MEMS
-Sensor verfügt über drei wählbare Messbereiche von +/-2, +/-4 und +/-8 g. Die Messung erfolgt mit einer Bandbreite von bis zu 1.5 kHz - ebenfalls einstellbar. Praktisch an diesem Modul ist zum einen, dass die Platine bereits bestückt ist. Der Sensor im LGA Gehäuse wäre nämlich nur schwer per Hand zu löten. Außerdem befinden sich ein 2.5V Spannungsregler und Signalpegelewandler auf der Platine, das Modul kann also an 5V betrieben und über I2C oder SPI direkt an einen Mikrocontroller mit 5V-Logik angebunden werden. Mit 5.95 € ist das Modul natürlich außerdem recht günstig. Weil ich schon immer mal mit Beschleunigungssensoren experimentieren wollte, konnte ich nicht widerstehen und habe mir dieses Modul bestellt. Im folgenden möchte ich kurz beschreiben, wie die I2C-Kommunikation mit einem AVR mittels BASCOM realisiert werden kann.

Beschaltung | Die Beschaltung des Sensormoduls für den Betrieb am I2C-Bus stellt sich naturgemäß simpel dar. Ich habe den Sensor mit meinem miniAVR-Modul verbunden, auf dem ein Mega8 verbaut ist. Natürlich kann auch jeder andere µC aus der 8bit AVR Familie verwendet werden. Die folgende Abbildung zeigt den Aufbau. Die beiden Widerstände sind die Pullups, die der I2C-Standard fordert und dementsprechend 4.7k groß.



Kommunikation über BASCOM | Das folgende Stückchen Code zeigt, wie man in BASCOM mit dem Sensor kommuniziert:

{codecitation class="brush: plain; gutter: true;toolbar: false" width="509px"}' kleines beispielprogramm zum auslesen des bma020
' malte ahlers 2010
' weitere infos auf malteahlers.de
'
'
' compiler steuerung
$regfile = "m8def.dat"
$crystal = 8000000
$framesize = 64
$swstack = 64
$hwstack = 64
$baud = 9600
$lib "i2c_twi.lbx"
'
' hardware konfigurieren
' - taster
Config Pinb.6 = Input
Portb.6 = 1
Config Pinb.7 = Input
Portb.7 = 1
Config Pind.5 = Input
Portd.5 = 1
Taster1 Alias Pinb.6
Taster2 Alias Pinb.7
Taster3 Alias Pind.5

```
' - leds
Config Portd.7 = Output
Config Portd.6 = Output
Config Portb.0 = Output
Led_gl Alias Portd.7
Led_rt Alias Portd.6
Led_gr Alias Portb.0
' - i2c
Config Twi = 400000
Config Scl = Portc.5
Config Sda = Portc.4
'
' variablen dimensionieren
Dim V(6) As Byte
Dim Ax As Integer At V(1) Overlay
Dim Ay As Integer At V(3) Overlay
Dim Az As Integer At V(5) Overlay
Dim Acc_cntr As Byte
Dim I As Byte
Dim Acc_range As Byte
Dim D2g As Single
Dim S As Single
'
' konstanten
Const Acc_w = &H70
Const Acc_r = &H71
'
' subs
Declare Sub Set_acc_range(byval Range As Byte)
'
' hauptprogramm
Print "acc test!"
Wait 1
'
I2cinit
'
Call Set_acc_range(0)
' messbereich:
' Set_acc_range(0) -> +/- 2g (default)
' Set_acc_range(1) -> +/- 4g
' Set_acc_range(2) -> +/- 8g
'
Do
 I2cstart
 '
 ' sensor adressieren
```

```
 I2cwbyte Acc_w
 '
 ' acc datenregister adressieren
 I2cwbyte &H02
 '
 I2cstop
 '
 I2cstart
 '
 ' daten lesen
 I2cwbyte Acc_r
 '
 For I = 1 To 5
 I2crbyte V(i) , Ack
 Next I
 I2crbyte V(6) , Nack
 '
 I2cstop
 '
 ' format konvertieren
 Ax = Ax / 64
 Ay = Ay / 64
 Az = Az / 64
 '
 ' in beschleunigungen umrechnen & ausgeben
 S = Ax * D2g
 Print "a(x)= " ; S ;
 S = Ay * D2g
 Print "  a(y)= " ; S ;
 S = Az * D2g
 Print "  a(z)= " ; S
 '
 Toggle Led_rt
 Waitms 10
 '
Loop
'
End
'
Sub Set_acc_range(byval Range As Byte)
 I2cstart
 '
 ' sensor adressieren
 I2cwbyte Acc_w
 '
 ' kontrollregister adressieren
```

```
 I2cwbyte &H14
 '
 I2cstop
 '
 I2cstart
 ' kontrollregister lesen
 I2cwbyte Acc_r
 I2crbyte Acc_cntr , Nack
 '
 I2cstop
 '
 ' kontrollregister manipulieren
 Acc_cntr.3 = Range.0
 Acc_cntr.4 = Range.1
 '
 I2cstart
 '
 ' kontrollregister zurückschreiben
 I2cwbyte Acc_w
 I2cwbyte &H14
 I2cwbyte Acc_cntr
 '
 I2cstop
 '
 ' umrechnungsfaktor neu berechnen
 D2g = 2 ^ Range
 D2g = D2g * 2
 D2g = D2g / 512
 '
End Sub{/codecitation}
```

Code zum Download: bma020_02.bas

Ich beschreibe an dieser Stelle einige Aspekte des Codes, die im engeren Sinne mit dem Sensor zu tun haben. Weil die TWI-Hardware des Mega8 für die I2C Kommunikation verwendet werden soll, ist dem Compiler die Verwendung der entsprechenden Library vorzuschreiben ('$lib = "i2c_twi.lbx"'). Dementsprechend muss dann definiert werden, welche Pins des Controllers SCL und SDA sind, das geschieht mittels der enstprechenden 'Config' Befehle. Die fest eingestellte I2C-Adresse des BMA020 zum Schreiben ist &H70. Das LSB der Adresse gibt beim I2C-Standard ja an, ob gelesen oder geschrieben werden soll, der Lesezugriff erfolgt also über &H71. Diese Adressen werden als Konstanten definiert.

Bevor es darum gehen soll, wie man die Beschleunigungsdaten vom Sensor ausliest, noch eine kurze Erläuterung zum Datenformat und wie damit in meinem Code umgegangen wird. Betrachten wir zunächst nur den Wert der x-Beschleunigung. Da die Auflösung des Sensors 10 bit beträgt, ist ein Messwert auf zwei 8 Bit Datenregister verteilt. Die Bits 0 - 7 des Register &H03 enthalten den höherwertigen Anteil der 10 bits, d. h. bit 7 ist das MSB der Daten. Die Bits 7 - 6 von &H02 enthalten den niederwertigen Anteil, Bit 6 ist also das LSB der Daten. Nun ist wichtig zu wissen, dass die Beschleunigungswerte im Zweierkomplement codiert sind, weil sie natürlich vorzeichenbehaftet sind. Um sich die Umwandlung in einen Dezimalwert per Hand zu ersparen, soll der Compiler dazu verwendet werden. Der Datentyp Integer ist wie allgemein üblich auch bei BASCOM im Zweierkomplement organisiert. Da allerdings ein Integer 16 bit groß ist, ist das MSB (das Vorzeichenbit) Bit 15 (das erste Bit einer Variablen nenne ich Bit 0, das sechzehnte eines Integers also Bit 15). Das MSB eines Beschleunigungsdatums ist aber wie wir gesehen haben dessen zehntes Bit. Wir schieben darum einfach die Beschleunigungsdaten um sechs Stellen nach links. Das kommt zwar einer Multiplikation mit 2^6 = 64 gleich, die kann ja aber im Weiteren durch Division wieder rückgängig gemacht werden. Das Entscheidende an diesem Schritt ist, dass somit das Vorzeichenbit an der richtigen Stelle ist und die Daten korrekt als Zweierkomplement interpretiert werden können. Nun machen wir es uns besonders einfach und sparen uns das explizite Schieben der Bits, indem wir einen kleinen Trick anwenden. BASCOM bietet ja die Möglichkeit, mit 'Dim... At... Overlay' Variablen überschneidend zu definieren. Und das tun wir hier. Wir legen die Integer Variable 'Ax' so über das Bytearray 'V' (in dem byteweise unsere Sensordaten stehen), dass das MSB der Sensordaten dem MSB des Integers entspricht, was schlicht und einfach der Fall ist, wenn 'Ax' bei 'V(1)' beginnt und sich bis 'V(2)' erstreckt. Man braucht jetzt nur noch das implizite Linksschieben durch eine Division durch 64 (was einem Schieben um sechs Bits nach rechts entspricht) rückgängig machen, schon hat man den Messwert im korrekten Format. Das Entsprechende tut man natürlich auch für die Messwerte der anderen beiden Achsen. Nun ist noch wünschenswert, den Digitalwert in physikalische Einheiten von g umzurechnen. Das geschieht über den Faktor 'D2g,' der abhängig vom gewählten Messbereich ist. Beim default Messbereich +/- 2g ist 'D2g' dann natürlich 4g / 2^{10} Bit.

Der Ablauf der I2C Lese- und Schreibvorgänge sind im Datenblatt des BMA020 auf den Seiten 34 und 35 beschrieben. Hält man sich an den Ablauf, kann garnichts schiefgehen. Für einen Lesezugriff erzeugt man eine Startbedingung und überträgt die Sensoradresse, danach die Adresse des zu lesenden Registers (bei &H02 beginnen die Daten) und erzeugt abschließend eine Stoppbedingung. Nach einer erneuten Startbedingung und dem Schreiben der Leseadresse kann man sich die sechs Datenbytes in Folge abholen. Nach dem sechsten Byte signalisiert ein 'Nack' das Ende der Datenübertragung, danach wird mit einer Stoppbedingung terminiert. Nun kann die oben beschriebene Umrechnung der Daten erfolgen, danach werden sie in obigem Beispielprogramm per RS232 ausgegeben.