

INTER-INTEGRATED CIRCUIT

Datenaustausch des Beschleunigungssensors
ADXL345 C

Inhaltsverzeichnis

Aufgabenstellung.....	2
Anforderungsanalyse	3
Designüberlegung.....	4
Team	5
Technologiebeschreibung	6
Aufwandsabschätzung und Arbeitszeitaufzeichnung	7
Arbeitsdurchführung	8
Testbericht.....	12
Conclusio	13
Quellenangabe	14

Aufgabenstellung

Erstellen Sie ein Projekt, wo der ADXL345 über den I2C-Bus die Informationen an den uC sendet. Geben Sie diese drei Achsen auf der seriellen Schnittstelle (UART) zur Kontrolle aus.

Anforderungsanalyse

- **Jegliche erhaltene Informationen sollen via UART auf der Konsole ausgegeben werden**
Die Konfiguration des UART-Outputs ist von Nöten
- **Die benötigten Register, die im Register-Map des Beschleunigungssensor-Datenblattes zu finden sind, müssen in der Applikation vordefiniert werden.**
Die Register, die in der Register-Map beschrieben wurden, als globale Variablen deklarieren
- **Für den Datenaustausch muss eine Verbindung zwischen dem Mikrocontroller und dem Beschleunigungssensor bestehen**
Nach Datenblätter der Hardware arbeiten und die richtigen Ports miteinander verbinden
- **Der Mikrocontroller soll die Daten des Beschleunigungssensors mittels I²C erhalten**
Das Anwenden jeglicher notwendiger Funktionen, die in der Driver-Library vorhanden sind
- **Bei den erhaltenen Daten handelt es sich um Rohdaten, die mittels einer Bitshift-Operation angepasst werden müssen**
Die Definition und den Einsatz einer Methode, die diesen Bitshift-Vorgang durchführt

Designüberlegung

Die geschriebene Applikation soll in Header-Files zerlegt werden, um die Erweiterung des Codes zu gewährleisten. Wenn zum Beispiel ein Upgrade erfolgt und weitere Register verwendet werden müssen, dann sollen diese in einem eigenen Header-File als Globale Variable deklariert werden.

Die notwendigen Funktionen der Driver-Library sollen in richtiger Reihenfolge in Methoden Einsatz finden.

readProcess:

```
I2CMasterSlaveAddrSet()  
I2CMasterDataPut()  
I2CMasterControl()  
while(I2CMasterBusy())  
I2CMasterSlaveAddrSet()  
I2CMasterControl()  
while(I2CMasterBusy())  
I2CMasterDataGet
```

writeProcess:

```
I2CMasterSlaveAddrSet(..., ..., true/false)
```

Relativ ähnlich zum Lesevorgang, nur muss der Schreibvorgang als ein solcher definiert werden (true für einen Lesevorgang und false für einen Schreibvorgang).

Team

Wir (Hannah Siegel, Wolfgang Mair, Christian Janeczek und Andreas Vogt) haben beschlossen uns nach der Schule am Samstag den 10.1.2015 in der Schule zu treffen und diese Aufgabe als Team zu lösen. Da uns nur ein Beschleunigungssensor zur Verfügung gestellt wurde und unsere Kenntnis für eine Einzelarbeit nicht ausreichend war, haben wir in einem Team für diese Aufgabe gearbeitet. Obwohl wir diese Aufgabe als Team gelöst haben, bedeutet dies nicht, dass wir den Code voneinander kopiert haben. Unsere Zusammenarbeit hat mittels dem Teilen der Informationen, die man sich erarbeitet hat, funktioniert. Wir hoffen, dass diese Teambildung keine Probleme aufwirft. Wir sind außerdem bereit zu beweisen, dass jeder von uns seinen Teil beigetragen hat, um diese Aufgabe zu lösen. Jedes Gruppenmitglied hat seinen Code selbst entwickelt und seine eigenen Kommentare/Dokumentation geschrieben.

Technologiebeschreibung

Der Begriff I²C steht für Inter-Integrated Circuit und es handelt sich hierbei um einen seriellen Datenbus, der für die geräteinterne Kommunikation genutzt wird. I²C an sich ist als Master-Slave-Bus konzipiert, dies bedeutet, dass der Datentransfer immer durch einen Master initiiert wird. Der über die SLAVE_ADRESS angesprochene Slave reagiert auf diesen Datentransfer. Ein weiterer Pluspunkt ist, dass mehrere Master im Multimaster-Mode ermöglicht werden. Zwei Master können direkt miteinander kommunizieren, bei diesem Vorgang arbeitet ein Gerät als Slave.[1]

Aufwandsabschätzung und Arbeitszeitaufzeichnung

Janeczek:

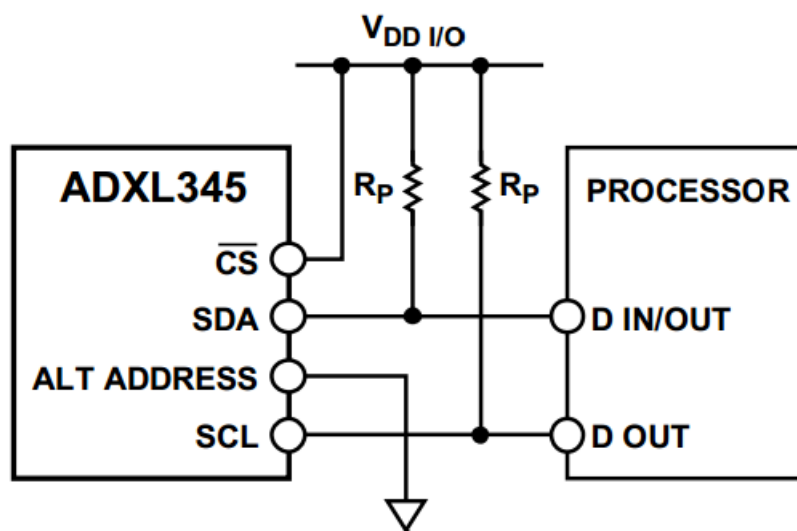
PHASE	TASK	ESTIMATION	ACTUAL	COMMENT
Implementierung	Die Konfiguration von UART	0:30:00	1:00:00	Eine Zeile der Konfiguration ist wesentlich.
Implementierung	Analyse des Example-Codes I2C	1:00:00	0:45:00	Mit Hilfe der Kommentare, die im Example-Code geschrieben wurden und der Dokumentation, die in der Driver-Library zu finden ist, war die Intention des Codes klar übersichtlich und verständlich.
Implementierung	Schreiben der eigentlichen Funktionalität	3:00:00	4:00:00	Aufgrund mangelnden Verständis musste im Datasheet und in der Driver-Library nachgeschlagen werden, um die Funktionen erfolgreich zu implementieren.
Implementierung	Debuggen des Codes und Testing mit dem Beschleunigungssensor	4:00:00	5:00:00	Es muss ein Schreibvorgang auf das POWER_MODE Register erfolgen, sodass der Sensor Werte liefert. Desweiteren hat der Vorgang des Bitshiftings mehr Zeit in Anspruch genommen, als gedacht.
Dokumentation	Schreiben des Protokolls	1:00:00	1:30:00	Technologiebeschreibung, Verbindung zwischen MC und ACM, Code-Snippets, etc.
SUM		9:30:00	12:15:00	

Arbeitsdurchführung

Zuallererst bin ich die Anforderungsanalyse Schritt für Schritt durchgegangen und habe meine Arbeitsschritte in folgende Punkte gegliedert:

- **Lesen der Datenblätter(PLURAL) und der Driver-Library um Verständnis für I²C zu erlangen**
Die Driver-Library[2] ist wundervolles Dokument, indem die Hintergründe der einzelnen Funktionen erläutert werden
- **Das Verbinden des ADXL345-Beschleunigungssensors mit dem Mikrocontroller, wie im Datenblatt beschrieben**

Wie im Datenblatt des Sensors beschrieben müssen folgende Kontakte miteinander verbunden werden. Ich referenziere hierfür auf eine Graphik im ADXL345-Datenblatt[3]:



GND(Ground) mit – auf dem Steckbrett

V(Voltage) mit + auf dem Steckbrett

SCL(Clock) mit dem Port **PB2** des MC

SDA(Data) mit dem Port **PB3** des MC

Auf das Verwenden von Widerstände darf nicht vergessen werden. Die Größe dieser wird im Datasheet erläutert.

- **Die Benutzung eines Templates mit funktionstüchtigem Makefile**
Hierfür habe ich das Template von mborko verwendet[4]
- **Die Konfiguration von UART, sodass die Daten im späteren Verlauf ausgegeben werden können**

```
void InitConsole(void)
{
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
    ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
    ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
    UARTStdioConfig(0, 115200, 16000000);
}
```

➤ **Das Setzen jeglicher verwendeter Register, wie im Datenblatt beschrieben**

```
// ADXL DOCUMENTATION PAGE 23
// REGISTER MAP

// Page 18
#define SLAVE_ADRESS 0x53
#define ID 0x00
// Page 23
#define P_MODE 0x3f
// Config on page 25
#define P_SETTINGS 0x2d
// Adresses for raw data are located at page 23
#define X_AXIS 0
#define DATA_X0 0x32
#define DATA_X1 0x33

#define Y_AXIS 1
#define DATA_Y0 0x34
#define DATA_Y1 0x35

#define Z_AXIS 2
#define DATA_Z0 0x36
#define DATA_Z1 0x37
```

➤ **Die Freischaltung der Ports und Pins, die von I²C verwendet werden, sowie die Initialisierung des I²C-Master-Blocks**

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
GPIOPinConfigure(GPIO_PB2_I2C0SCL);
GPIOPinConfigure(GPIO_PB3_I2C0SDA);
GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_3);
GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
//Initializes the I2C Master block.
I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), false);
```

➤ **Das Schreiben auf das POWER_MODE Register, sodass der Beschleunigungssensor Daten liefert**

```
// Without writing to the P_MODE, the sensor won't start to measure
// We have to write the power-settings to the power-mode register
writeProcess(SLAVE_ADRESS, P_SETTINGS, P_MODE);
```

➤ **Der Lesevorgang mittels I²C**

```
//Sets the address that the I2C Master places on the bus.
//This function configures the address that the I2C Master places on
//the bus when initiating a transaction. When the bReceive parameter
//is set to true, the address indicates that the I2C
//Master is initiating a read from the slave; otherwise the address
//indicates that the I2C Master is initiating a write to the slave.
I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, false);
//Transmits a byte from the I2C Master.
```

```
I2CMasterDataPut(I2C0_BASE, registerAddress);
//Controls the state of the I2C Master module.
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
//Indicates whether or not the I2C Master is busy.
while(I2CMasterBusy(I2C0_BASE)) {
    //Continue as soon as the I2C Master isn't busy anymore.
}
int8_t data = 0;
    //Sets the address that the I2C Master places on the bus.
I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, true);
//Controls the state of the I2C Master module.
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
//Indicates whether or not the I2C Master is busy.
while(I2CMasterBusy(I2C0_BASE)) {
    //Continue as soon as the I2C Master isn't busy anymore.
}
//Receives a byte that has been sent to the I2C Master.
data = (int8_t)I2CMasterDataGet(I2C0_BASE);
return data;
```

➤ **Der Schreibvorgang mittels I²C**

```
I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, false);
//Transmits a byte from the I2C Master.
I2CMasterDataPut(I2C0_BASE, registerAddress);
//Controls the state of the I2C Master module.
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
//Indicates whether or not the I2C Master is busy.
while(I2CMasterBusy(I2C0_BASE)) {
    //Continue as soon as the I2C Master isn't busy anymore.
}
    //Sets the address that the I2C Master places on the bus.
I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, false);
    //Transmits a byte from the I2C Master.
I2CMasterDataPut(I2C0_BASE, writeProcess);
//Controls the state of the I2C Master module.
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
while(I2CMasterBusy(I2C0_BASE)) {
    //Continue as soon as the I2C Master isn't busy anymore.
}
```

➤ **Der Bitshifting-Vorgang**

```
//Bitshifting to the left
int16_t shifted = (int16_t) ((right << 8) | (left & 0xff));
//      Floating point info through multiplication
return (int32_t) shifted * 4;
```

➤ **Alle Funktionalitäten zusammen**

```
while (true) {  
    UARTprintf("DATA_X: %i -", shiftProcess(readProcess(SLAVE_ADRESS, DATA_X0),  
        readProcess(SLAVE_ADRESS, DATA_X1)));  
  
    UARTprintf("DATA_Y: %i -", shiftProcess(readProcess(SLAVE_ADRESS, DATA_Y0),  
        readProcess(SLAVE_ADRESS, DATA_Y1)));  
  
    UARTprintf("DATA_Z: %i\n", shiftProcess(readProcess(SLAVE_ADRESS, DATA_Z0),  
        readProcess(SLAVE_ADRESS, DATA_Z1)));  
  
    //Provides a small delay.  
    // 1/10 of the processor clock rate.  
  
    SysCtlDelay(SysCtlClockGet()/10);  
}
```

Testbericht

- Eine ordnungsgemäße Verbindung zum Sensor:

Expected: Verfälschte Werte

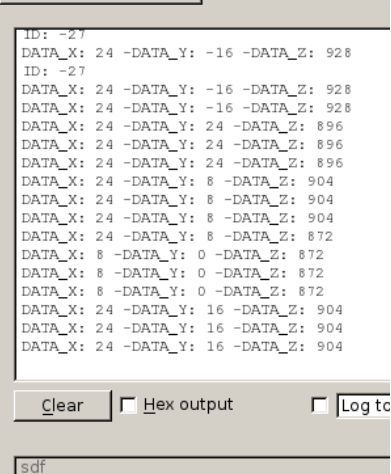
Actual: Verfälschte Werte

```
ROM_SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN);
InitI2C();
InitConsole();

UARTprintf("ID: %i\n", readProcess(SLAVE_ADRESS, ID));

// Without writing to the P_MODE, the sensor won't start
writeProcess(SLAVE_ADRESS, P_SETTINGS, P_MODE);

while (true) {
    UARTprintf("DATA_X: %i -", shiftProcess(readProcess(SLAVE_ADRESS, DATA_X)));
    UARTprintf("DATA_Y: %i -", shiftProcess(readProcess(SLAVE_ADRESS, DATA_Y)));
    UARTprintf("DATA_Z: %i\n", shiftProcess(readProcess(SLAVE_ADRESS, DATA_Z)));
    SysCtlDelay(SysCtlClockGet()/10);
}
return(0);
```



- Das POWER-MODE Register wurde nicht gesetzt:

Expected: Verfälschte Werte

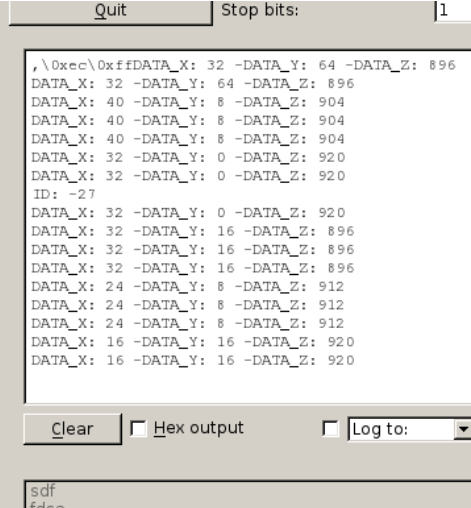
Actual: Verfälschte Werte

```
ROM_SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN);
InitI2C();
InitConsole();

UARTprintf("ID: %i\n", readProcess(SLAVE_ADRESS, ID));

// Without writing to the P_MODE, the sensor won't start
//writeProcess(SLAVE_ADRESS, P_SETTINGS, P_MODE);

while (true) {
    UARTprintf("DATA_X: %i -", shiftProcess(readProcess(SLAVE_ADRESS, DATA_X)));
    UARTprintf("DATA_Y: %i -", shiftProcess(readProcess(SLAVE_ADRESS, DATA_Y)));
    UARTprintf("DATA_Z: %i\n", shiftProcess(readProcess(SLAVE_ADRESS, DATA_Z)));
    SysCtlDelay(SysCtlClockGet()/10);
}
return(0);
```



Conclusio

- Wenn uns der Professor mitteilt, dass das Lesen der Dokumentation einen Haufen an Arbeit abnimmt, dann sollten wir diesen Rat dankend annehmen und dies auch tun.
- Wenn man mit UART Probleme hat, sollte man diese nicht hinterfragen(UART kann mit einer Freundin, die ihre Periode hat, verglichen werden). Die Fehlersuche bei UART-Problemen ist eine Aufgabe für sehr geduldige Menschen.
- Die essentielle Nachricht dieser Aufgabe ist: POWER_MODE setzen!

Quellenangabe

- [1] **Inter-Integrated Circuit**, Wikipedia, <http://de.wikipedia.org/wiki/I%C2%B2C>
- [2] **Driver-Library**, Tiva, <https://github.com/mborko/tiva-template/blob/master/docs/SW-TM4C-DRL-UG-2.0.1.11577.pdf>
- [3] **ADXL345 Datasheet**, ADXL, http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf
- [4] **Tiva Template**, Michael Borko, <https://github.com/mborko/tiva-template>