STM32 - I2C

u23 2013

andy, florob, gordin, ike, meise, tobix, zakx

Chaos Computer Club Cologne e.V.
http://koeln.ccc.de

Cologne
2013-11-11

Teile!

Ey! Psst! Brauchste Teile?

Liegen hier vorne auf dem Tisch

andy <andy@koeln.ccc.de>

I2C

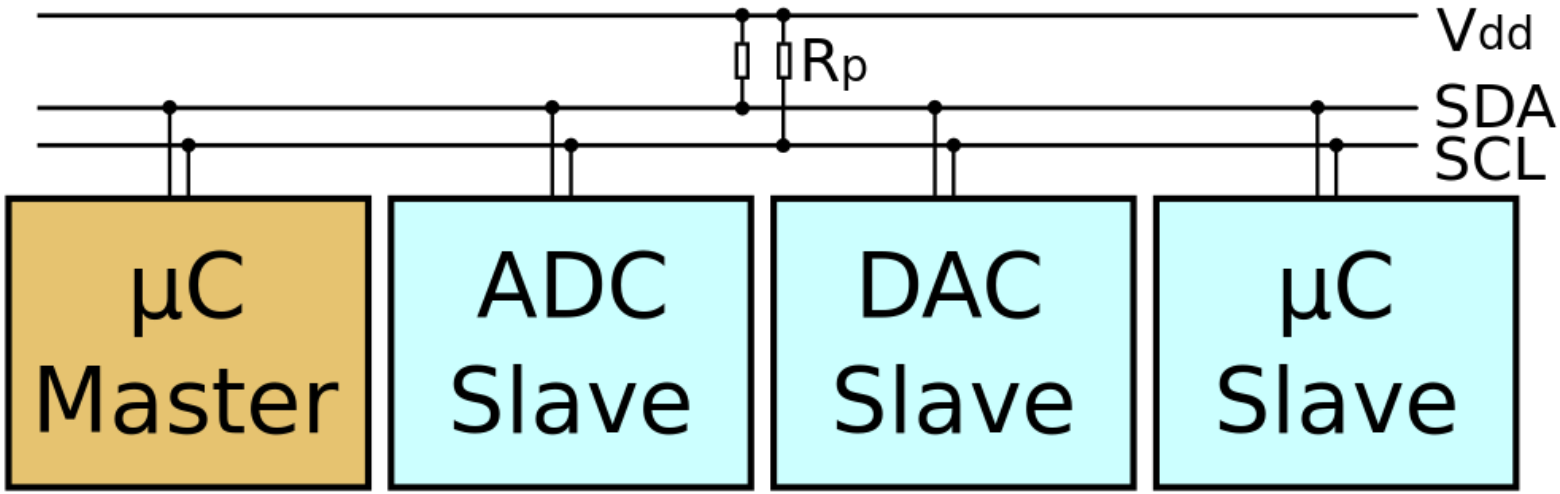
- I2C = Inter-integrated Circuit (I-Quadrat-C oder I-squared-C gesprochen)
- Markenname von Philips, sonst TWI = Two Wire Interface
- 2-Draht Bussystem für Bausteine die in einem geschlossenen System zusammen arbeiten (selbe Platine, selbes Gehäuse, ...)
- Alle Kommunikation geht von einem oder mehreren Mastern aus, Slavebausteine machen nie etwas von sich aus
- Verschiedene Geschwindigkeiten: 100kHz, 400kHz (fast mode), 1MHz (fast mode plus) und 3,4MHz (high speed mode)
- 7- oder 10-Bit lange Adressen für Slavebausteine

I2C

- Zwei Leitungen:
  1. SCL = Clock
  2. SDA = Data
- Beides Open-Drain-Leitungen (wir erinnern uns an die GPIOs)
- Pull-Ups müssen eigentlich auf Leitungslänge, Anzahl Teilnehmer, Geschwindigkeit etc. abgestimmt werden
- In der Praxis: Nehmt was zwischen 4,7K und 10K
- Standard scheint so 4,7K zu sein
- Mein großer Quadrocopter hat 2,2K, tut auch...
- Die HMC5883L-Kompassboards haben 2,2K

I2C

So sieht son Bus aus:



I2C

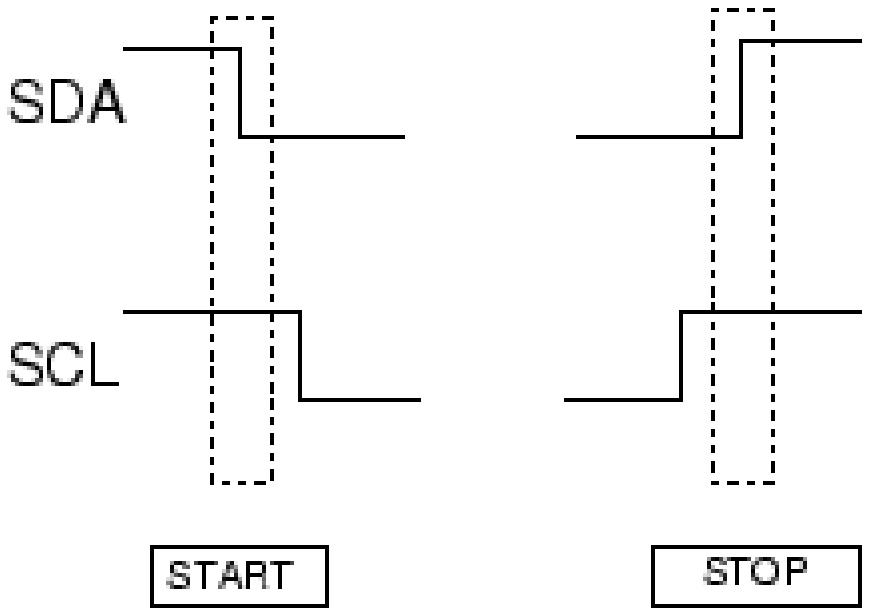
- SPI war einfach gestrickt: Daten rein und raus bei Clockgewackel, nur eine Richtung pro Datenleitung
- I2C ist etwas komplizierter, da hier ein Protokoll gesprochen werden muss
- Nur eine Datenleitung zum lesen und schreiben und Adressen für Slaves

I2C Protokoll

- 4 spezielle Bus-Conditions:
  1. START
  2. STOP
  3. ACK
  4. NACK

START- und STOP-Condition

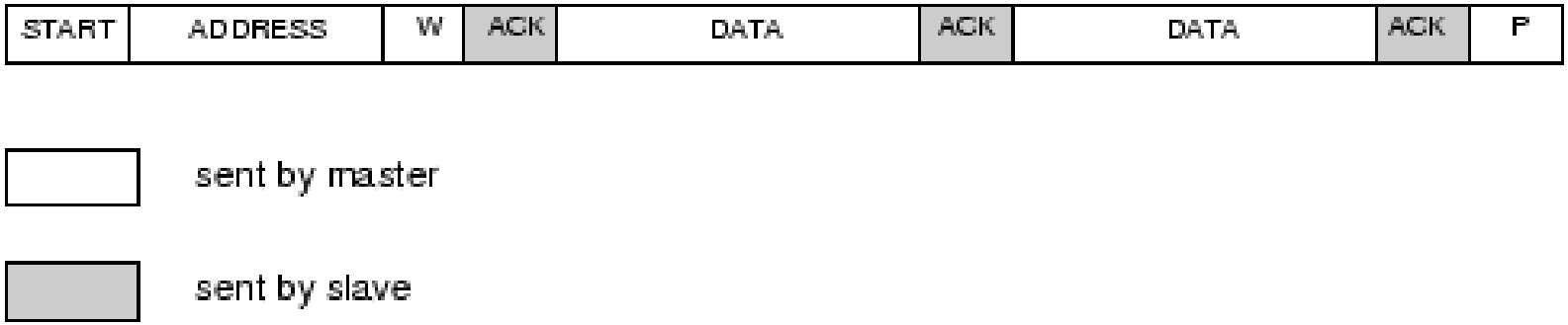
- START wird genutzt um den Bus zu aktivieren und eine Transaktion zu starten
- STOP beendet eine Transaktion egal in welchem Zustand und gibt den Bus wieder frei



ACK und NACK

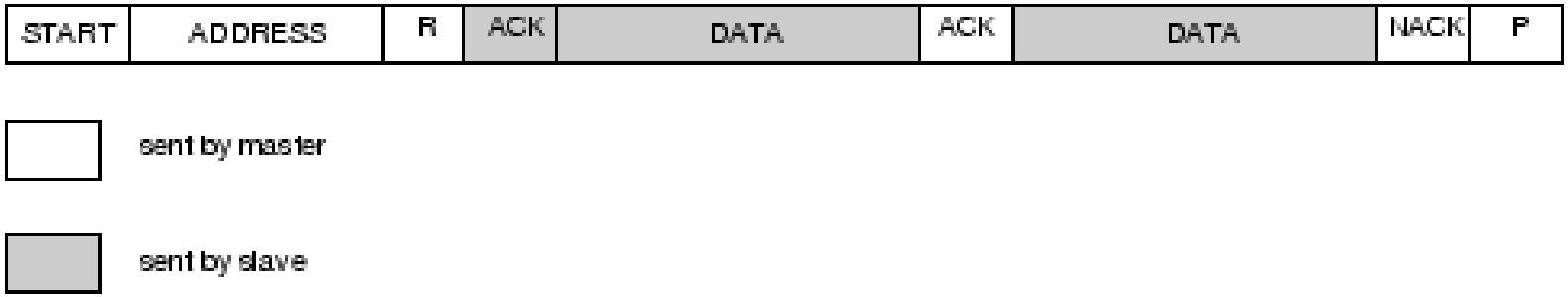
- ACK und NACK sind Bits die zu bestimmten Zeitpunkten im Komunikationsablauf gesendet werden
- ACK bestätigt empfang eines Bytes
- NACK bestätigt zwar den Empfang, signalisiert aber gleichzeitig einen Fehler oder ähnliches
- Nack einem NACK vom Slave kann der Master nur ein STOP oder Repeated START senden

Transfer Master -> Slave



1. START senden
2. 7-Bit Adresse senden
3. Read/Write-Bit senden (1=Read, 0=Write)
4. Auf ein ACK vom Slave warten
5. 8 Bit senden
6. Auf ACK-Bit warten
7. ...
8. 8 Bit senden
9. Auf ACK-Bit warten
10. STOP senden

Transfer Slave -> Master



1. START senden
2. 7-Bit Adresse senden
3. Read/Write-Bit senden (1=Read, 0=Write)
4. Auf ein ACK vom Slave warten
5. 8 Bit empfangen
6. ACK-Bit Senden
7. ...
8. 8 Bit empfangen
9. NACK-Bit senden
10. STOP senden

Repeated Start

- Wir können also jetzt lesen und schreiben
- Aber was ist, wenn wir erst schreiben wollen und dann lesen?

Register List
The table below lists the registers and their access. All address locations are 8 bits.

Address Location	Name	Access
00	Configuration Register A	Read/Write
01	Configuration Register B	Read/Write
02	Mode Register	Read/Write
03	Data Output X MSB Register	Read
04	Data Output X LSB Register	Read
05	Data Output Z MSB Register	Read
06	Data Output Z LSB Register	Read
07	Data Output Y MSB Register	Read
08	Data Output Y LSB Register	Read
09	Status Register	Read
10	Identification Register A	Read
11	Identification Register B	Read
12	Identification Register C	Read

Table2: Register List

Repeated Start

- Man schreibt die Registeradresse in einer ersten Transaktion
- Und liest dann den 8-Bit Wert
- Der Slave sollte dann den Registerinhalt zurückliefern
- VORSICHT! Das muss nicht so sein, in der Regel liest und schreibt man aber so Register

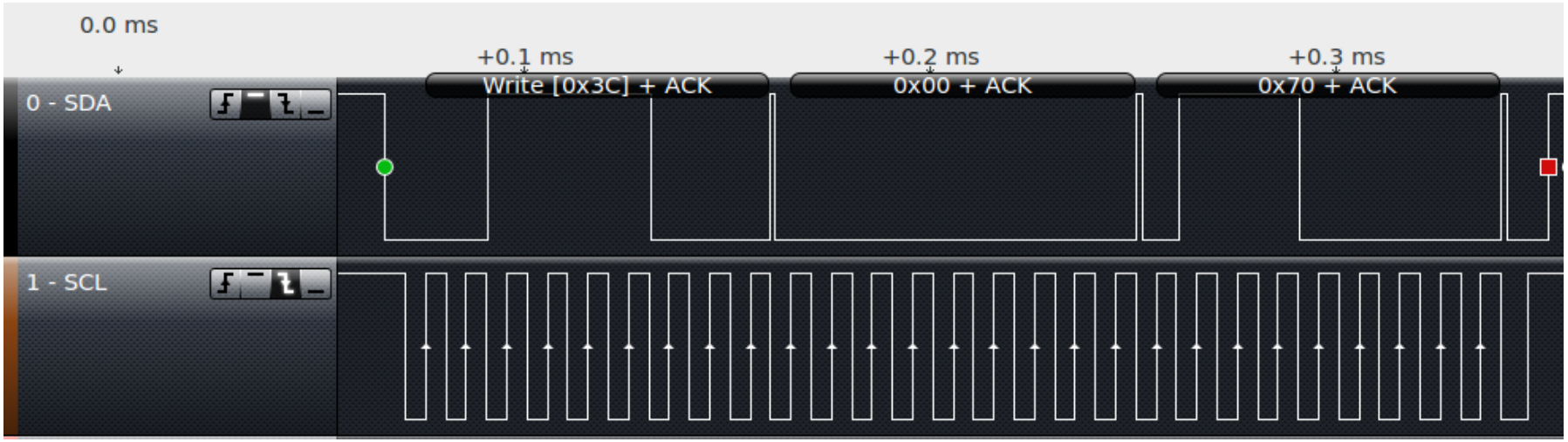
Repeated Start

- Dumm nur: Bus wird zwischen dem Write der Registeradresse wieder freigegeben
- In der Zwischenzeit kann such ein anderer Master den Bus unter den Nagel reißen
- Der schreibt dann eine Registeradresse auf dem selben Slave -> Wir lesen dann einen falschen Wert

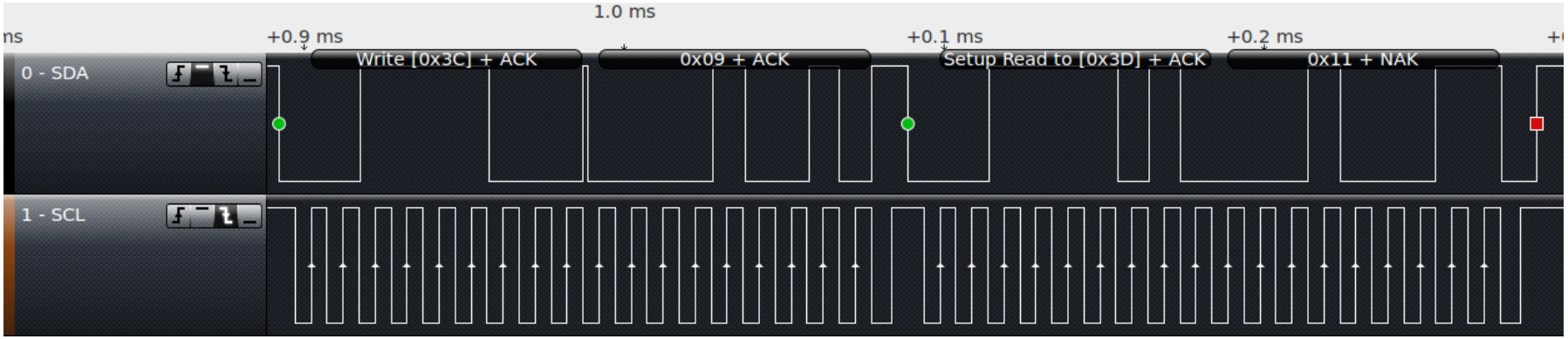
Repeated Start

- Die Lösung ist Repeated Start
- Statt einem STOP sendet man einfach nochmal START
- Dann wieder die Slaveadresse und ein R/W-Bit logischerweise jetzt invertiert, da man ja den Modus ändern will

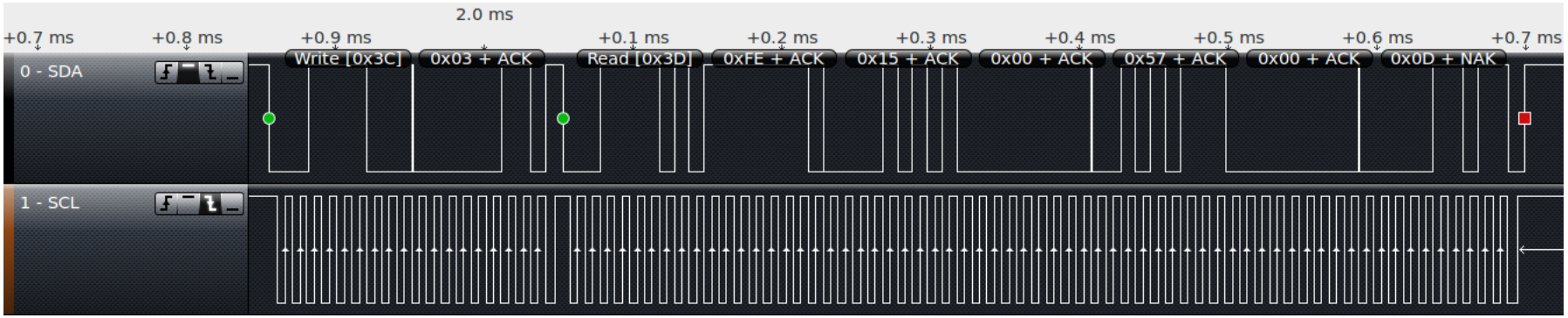
Write


0.0 ms
+0.1 ms
+0.2 ms
+0.3 ms
Write [0x3C] + ACK
0x00 + ACK
0x70 + ACK
0 - SDA
1 - SCL

Write, Repeated Start, Read



Read mehrerer Werte



Konfiguration

```
// Enable clock for GPIOA and GPIOC peripheral
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

// Set Pin modes for PA8 (SCL)
GPIO_Init(GPIOA, &(GPIO_InitTypeDef){
    .GPIO_Speed = GPIO_Speed_2MHz,
    .GPIO_Mode = GPIO_Mode_AF, //Not OUT, but AF = Alternate Function
    .GPIO_OType = GPIO_OType_OD,
    .GPIO_PuPd = GPIO_PuPd_NOPULL, //External pullups, so none here
    .GPIO_Pin = GPIO_Pin_8
});

// Set Pin modes for PC9 (SDA)
GPIO_Init(GPIOC, &(GPIO_InitTypeDef){
    .GPIO_Speed = GPIO_Speed_2MHz,
    .GPIO_Mode = GPIO_Mode_AF, //Not OUT, but AF = Alternate Function
    .GPIO_OType = GPIO_OType_OD,
    .GPIO_PuPd = GPIO_PuPd_NOPULL, //External pullups, so none here
    .GPIO_Pin = GPIO_Pin_9
});

//Attach alternate pin functions
GPIO_PinAFConfig(GPIOA, GPIO_PinSource8, GPIO_AF_I2C3); //SCL
GPIO_PinAFConfig(GPIOC, GPIO_PinSource9, GPIO_AF_I2C3); //SDA
```

Konfiguration

```
I2C_Init(I2C3, &(I2C_InitTypeDef){
    .I2C_ClockSpeed = 100000, //see note above
    .I2C_Mode = I2C_Mode_I2C, //we want raw I2C, no SMBUS or other stu
    .I2C_DutyCycle = I2C_DutyCycle_2, //only relevant for fast mode
    .I2C_OwnAddress1 = 0xEE, //only relevant for slave mode
    .I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit, //7-bit ad
    .I2C_Ack = I2C_Ack_Disable //wether or not to acknowledge automatical
});

//Turn that thing on...
I2C_Cmd(I2C3, ENABLE);
```

Register schreiben

```
I2C_start(I2C3, SLAVE_ADDRESS<<1, I2C_Direction_Transmitter);
I2C_write(I2C3, 0x00); // set pointer to CRA
I2C_write(I2C3, 0x70); // write 0x70 to CRA
I2C_stop(I2C3);
```

Register lesen

```
//Read Status register, just for the lulz
uint8_t status = 0;
I2C_start(I2C3, SLAVE_ADDRESS<<1, I2C_Direction_Transmitter);
I2C_write(I2C3, 0x09); // set pointer to Status register
I2C_restart(I2C3, SLAVE_ADDRESS<<1, I2C_Direction_Receiver); //repeate
status = I2C_read_nack(I2C3); //read byte, nack and stop
```

I2C Beispiel

1. Eigentlich ist alles was ihr benötigt im *09_i2c* example
2. Das Ding redet mit einem HMC5883L I2C-Kompass
3. Da haben wir leider nur 2 von :(
4. Ansonsten gibts noch 3 Realtimeclocks, wo es noch keinen Code für gibt -> Viel Spaß!

Aufgaben

1. Redet mal mit dem Magnetometer und der Realtimeclock
2. Ihr könnt auch mal den I2C-Slavemode implementieren (hab ich noch nie gemacht)
3. besorgt euch ein paar andere Teile und macht $stuff
4. Beispiel: Steuert (Multiplexing) 7-Segment Anzeigen, Displays, Joysticks, die Funkmodule, Servos usw. an