Proseminar Mikrocontroller:
TWI im ATmega 16

Christopher Israel

Juli 2006

Inhaltsverzeichnis

1 Einleitung

1.1 Allgemeines

Der ATmega 16 besitzt ein „Two-Wire Serial Interface“. Diese Schnittstelle ist äquivalent zum I^{2}C-Bus, der, wie „Two Wire“ schon sagt, mit nur zwei Leitungen auskommt.
Dieser Bus ist ein serieller Datenbus, der um 1980 von Philips entwickelt wurde, um in Geräten wie Fernsehern mehrere ICs an Mikroprozessoren anschließen zu können ohne breite Adress- und Datenleitungen zu benötigen. Daher kommt auch die Bezeichnung I^{2}C, was eine Abkürzung für „Inter IC“ ist. Gemeint ist damit, dass der Bus für Verbindungen zwischen ICs gedacht ist.
Der Bus hat in der aktuellen I^{2}C-Spezifikation 2.1 von 2001 eine maximale Datenübertragungsrate von 3,4 Mbit/s (High-Speed* Modus), jedoch ist der ATmega 16 ein Fast-Mode Gerät, daher können nur Übertragungsraten von maximal 400 kbit/s verwendet werden.

1.2 Eigenschaften

Der große Vorteil des Busses ist, dass man mit nur zwei Leitungen viele ICs miteinander verbinden kann.
Dazu kommt, dass während einer Datenübertragung der Takt automatisch von den beteiligten Geräten angepasst werden kann, falls eines der Geräte mehr Zeit zur Verarbeitung der Daten benötigt.
Es ist außerdem ein Arbitrationsverfahren vorgesehen. Dieses Verfahren sorgt dafür, dass nur ein Gerät erfolgreich Daten auf den Bus schreiben kann, wenn mehrere Geräte gleichzeitig versuchen eine Verbindung auf dem Bus zu initiieren.
Zur Adressierung der am Bus angeschlossenen Geräte stehen wahlweise 7-bit oder 10-bit Adressierung und damit 112† bzw. 1024 Adressen zur Verfügung. Dadurch, dass es so wenig Adressen gibt, jedoch die Auswahl an ICs mit I^{2}C-Ansteuerung groß ist, kommt dazu, dass verschiedenen Geräten desselben Typs dieselbe Adresse zugewiesen wird.
Problematisch kann dies werden, wenn man beispielsweise mehrere serielle EEPROMs am Bus betreiben möchte, zumal manche I^{2}C-fähigen EEPROMs gleich 8 Adressen belegen, wodurch bei 7-bit Adressierung nur ein einziges EEPROM am Bus betrieben werden kann.
Generell können durch die in der Spezifikation festgelegte maximale Leitungskapazität von 400pF ca. 20 Geräte an einem Bus betrieben werden, alternativ können die Leitungen ca. 10 m lang sein. Durch I^{2}C Repeater o. Ä. kann diese Beschränkung jedoch umgangen werden.

*Die I^{2}C-Spezifikation unterscheidet zwischen drei verschiedenen Modi:
Standard-Mode mit 100 kbit/s, Fast-Mode mit 400 kbit/s und High-Speed Mode mit 3,4Mbit/s. Fast-Mode und High-Speed Mode sind dabei zu den jeweils langsameren Modi abwärtskompatibel

†Bei der 7-bit Adressierung sind 16 Adressen reserviert

1.3 Einsatzgebiete

Der I^2C-Bus wird eingesetzt in Geräten wie zum Beispiel Fernsehern, Telefonen und Waschmaschinen. Es gibt Chipkarten, die über I^2C angesprochen werden und auch im Computerbereich wird der Bus eingesetzt, beispielsweise auf Mainboards in Form des SMBus, der auf dem I^2C-Bus basiert. In Monitorkabeln wird der DDC* verwendet, der ebenfalls auf dem I^2C-Bus basiert und der Grafikkarte erlaubt, Informationen mit einem Monitor auszutauschen.
ICs, die man in I^2C-Bussen verwenden kann sind u. a.

- EEPROMs
- Portexpander
- LCD-Displays
- Treiber
- AD-, bzw. DA-Wandler
- Sensoren

2 Ablauf einer Verbindung

2.1 SCL und SDA

Der I^2C-Bus verwendet zwei Leitungen zur Datenübertragung, dabei ist eine der beiden Leitungen eine Datenleitung, während die andere Leitung eine Taktleitung ist. Entsprechend werden die Leitungen **SDA** für **S**erial **DA**ta bzw **SCL** für **S**erial **CL**ock genannt. Beide Leitungen sind laut Spezifikation mit Pullups auf High-Level zu halten; die SDA- und SCL-Pins der angeschlossenen ICs sind Open-Drain bzw. Open-Collector. Daher muss ein am Bus angeschlossener IC um eine 1 zu senden die Leitung loslassen und um eine 0 zu senden die Leitung auf Masse ziehen. Es ergibt sich dadurch eine Wired-And Funktion der Busleitungen.

2.2 START- und STOP-Condition

Der Bus ist so ausgelegt, dass ein Busteilnehmer eine Verbindung zu einem anderen Busteilnehmer initiiert und nachher die Verbindung beendet oder eine weitere Verbindung aufbaut. Während einer Verbindung dürfen andere Busteilnehmer nicht auf den Bus zugreifen. Damit die am Bus angeschlossenen Geräte erkennen können, ob der Bus frei oder belegt ist, werden START- und STOP-Conditions verwendet, die Anfang und Ende einer Verbindung kennzeichnen.
Eine START-Condition wird ausgelöst durch einen *High-Low* Übergang auf SDA, während SCL High ist und eine STOP-Condition wird umgekehrt ausgelöst durch einen *Low-High* Übergang auf SDA, während SCL High ist.
Außer beim Auslösen von START- bzw. STOP-Conditions darf sich SDA während der High-Phase von SCL jedoch nicht verändern.

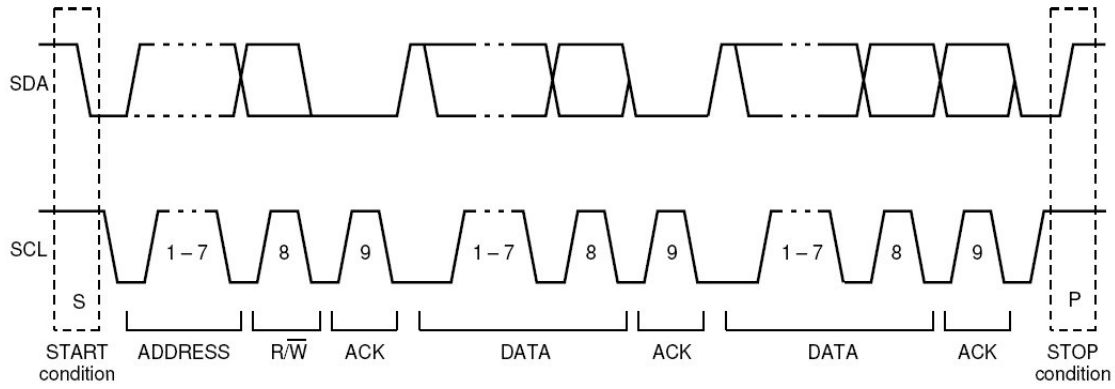
*Display Data Channel

2.3 Master und Slave, Transmitter und Receiver

Die Busteilnehmer können jeweils „Master“ oder „Slave“, „Transmitter“ oder „Receiver“ sein. Bei einer Verbindung zwischen zwei Busteilnehmern ist jeweils eines der Geräte Master bzw. Slave; gleichermaßen ist jeweils eines der Geräte Transmitter oder Receiver.
Dabei ist der Master dafür zuständig, die Verbindung zu einem Slave-Gerät zu initiieren und zu beenden, sowie den Takt auf der SCL-Leitung bereitzustellen. Außerdem bestimmt der Master die Datenrichtung der Verbindung, d. h. welches der beiden Geräte Receiver und welches Transmitter ist.
Ein Gerät, das die Rolle eines Slave einnimmt, spricht auf eine Verbindungsanfrage eines Masters an, wenn es die eigene Geräteadresse empfängt. Es reagiert dann abhängig von der Datenrichtung mit dem Senden, bzw. Empfangen von Daten. Zusätzlich hat der Slave die Möglichkeit, die Taktrate auf der SCL-Leitung herabzusetzen, wenn diese zu schnell ist um die empfangenen bzw zu sendenden Daten abzuarbeiten.
Der Transmitter ist das Gerät, das nach erfolgreichem Verbindungsaufbau Daten auf den Bus legt und der Receiver ist das Gerät, das die vom Transmitter gesendeten Daten empfängt.

Beispiel: Wenn ein Mikrocontroller einen Wert von einem Temperatursensor liest, so ist der Mikrocontroller ein Master Receiver (denn er initiiert die Verbindung zum Sensor und empfängt dessen Daten), während der Sensor ein Slave Transmitter ist (denn er wird adressiert und sendet Daten).

2.4 Verbindungsablauf



Jede Verbindung auf dem I^2C-Bus beginnt damit, dass ein Master eine START-Condition sendet. Sobald die START-Condition übertragen wurde, sendet der Master die 7-bittige Adresse des Slave-Geräts mit dem Datenrichtungsbit ($R/\bar{W}$). Wenn der Slave mit der passenden Adresse am Bus angeschlossen ist, so wird er zum nächsten SCL-High die SDA-Leitung auf Low ziehen und somit das ACK-Bit setzen.
Wenn sich jedoch kein Slave meldet, bleibt SDA auf High und es wird somit ein NACK-Bit gesetzt. In dem Fall eines NACK hat der Master die Wahl, entweder

eine STOP-Condition zu senden um den Bus wieder freizugeben, oder eine wiederholte START-Condition zu senden um einen anderen Slave zu adressieren.
Sofern ein ACK empfangen wurde, sendet der Transmitter eines oder mehrere Datenbytes, die vom Receiver jeweils mit ACK oder mit NACK bestätigt werden. Wenn der Receiver nach dem Empfang eines Datenbytes mit NACK antwortet, so ist dies für den Transmitter das Zeichen, dass der Receiver keine weiteren Bytes mehr benötigt.
Nach jedem ACK oder NACK hat der Master die Möglichkeit, die Übertragung mit einer STOP-Condition zu beenden oder mit einer wiederholten START-Condition eine neue Übertragung mit neuer Slave-Adresse und Datenrichtung zu beginnen.

Beispiel: Ein Mikrocontroller liest zwei Datenbytes ab einer bestimmten Speicheradresse von einem EEPROMs

- Der Mikrocontroller (der hier als Master arbeitet) sendet eine START-Condition und anschließend die Adresse des EEPROMs und das Schreibbit (1010xxx0).
- Das EEPROM bestätigt mit ACK, dass es seine Adresse empfangen hat und bereit ist, Daten zu empfangen.
- Der Mikrocontroller sendet die Speicheradresse* und das EEPROM bestätigt den Empfang mit ACK.
- Der Mikrocontroller sendet eine wiederholte START-Condition und die Adresse des EEPROMs, allerdings diesmal mit einem Lesebit (1010xxx1) um die Datenrichtung umzukehren, also um nun vom EEPROM zu lesen.
- Das EEPROM sendet ein Datenbyte und Mikrocontroller antwortet mit ACK.
- Das EEPROM sendet ein weiteres Datenbyte und Mikrocontroller antwortet diesmal mit NACK, um dem EEPROM mitzuteilen, dass es keine weiteren Datenbytes senden soll.
- Der Mikrocontroller sendet eine STOP-Condition.

2.5 Adressierung

Beim I^2C-Bus gibt es 2 verschiedene Arten der Adressierung: Die 7-Bit Adressierung und die 10-Bit Adressierung.

Bei der **7-Bit Adressierung** wird nach einer START-Condition eine Adresse mit 7 Bits, sowie ein Datenrichtungsbit übertragen. Die drei niedrigstwertigen Bits sind dabei meist am jeweiligen IC einstellbar (z. B. durch Adresspins am IC). Zwei Adressbereiche von jeweils 8 Adressen (1111XXX und 0000XXX) sind

*Bei den meisten I^2C-EEPROMS besteht die Möglichkeit, Daten ab einer bestimmten Wortadresse zu lesen, wenn zuerst die Wortadresse an das EEPROM übertragen wird und dann vom EEPROM gelesen wird. Nach dem Lesen eines Datenbytes inkrementiert das EEPROM meist die Wortadresse automatisch, sodass mehrere Datenbytes in Folge gelesen werden können.

jedoch reserviert. Die folgende Tabelle aus der I²C-Spezifikation beschreibt, wofür die reservierten Adressen verwendet werden.

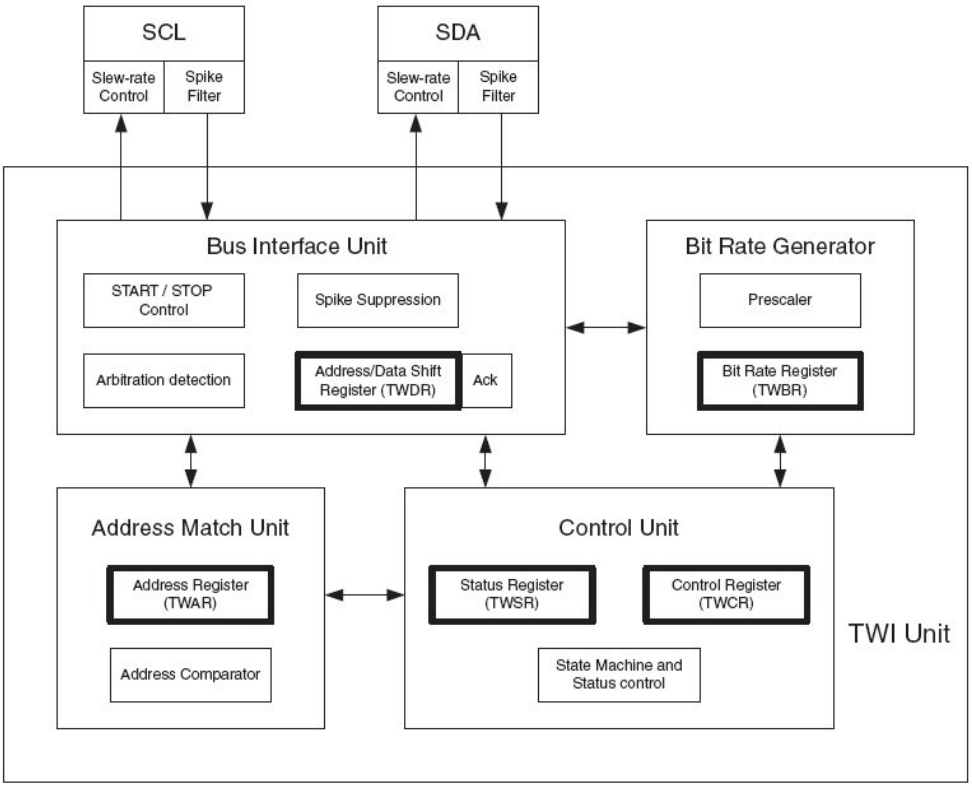
Table 2 Definition of bits in the first byte

SLAVE ADDRESS	$\overline{\text{R/W}}$ BIT	DESCRIPTION
0000 000	0	General call address
0000 000	1	START byte(1)
0000 001	X	CBUS address(2)
0000 010	X	Reserved for different bus format(3)
0000 011	X	Reserved for future purposes
0000 1XX	X	Hs-mode master code
1111 1XX	X	Reserved for future purposes
1111 0XX	X	10-bit slave addressing

Da mit der 7-Bit Adressierung nur 112 verschiedene Adressen möglich sind, wurde in der I²C-Spezifikation 1.0 von 1992 die **10-Bit Adressierung** eingeführt. Dadurch sind 1024 weitere Adressen verfügbar. Die 10-Bit Adressierung funktioniert dadurch, dass nach der START-Condition erst eine spezielle 7-Bit Adresse (11110XX) mit einem Schreibbit (0) übertragen wird, wobei die beiden niedrigstwertigen Bits dieser 7-Bit Adresse die beiden höchstwertigen Bits der 10-Bit Adresse sind. Weil die 7-Bit Adresse zu den reservierten Adressen gehört, wird kein Gerät darauf antworten, das nur 7-Bit Adressierung versteht, sondern ausschließlich Geräte, die eine 10-Bit Adresse haben, deren beiden höchstwertigen Bits den beiden niederwertigsten Bits der 7-Bit Adresse entsprechen. Nachdem ein Slave auf die Adresse geantwortet hat, kann nun der Master ein weiteres Datenbyte übertragen, das nun die nächsten 8 Bit der Adresse enthält. Wenn auf diese Adresse ein Slave mit einem ACK antwortet, so kann der Master anfangen Daten zu senden.
Sollte der Master allerdings Daten vom Slave lesen, so muss er eine wiederholte START-Condition auslösen und wieder die 7-Bit Adresse senden, allerdings diesmal mit einem Lesebit (11110XX1). Dadurch erkennt der zuvor adressierte Slave, dass er als Slave-Transmitter funktionieren soll und fängt an, Daten zu senden.

3 I²C im ATmega 16

3.1 Aufbau der TWI-Einheit



Wie man auf dem obigen Bild erkennen kann, besteht die TWI-Einheit des ATmega 16 aus 4 Teilen: Der Bus Interface Unit, der Control Unit, der Adress Match Unit und dem Bit Rate Generator.

- Die Bus Interface Unit ist dafür zuständig, die am Bus angelegten Daten zu empfangen und im TWI Data Register zu speichern oder Daten auf den Bus zu senden, die im Data Register bereitstehen. Außerdem erkennt bzw. erzeugt sie die START- und STOP-Conditions.
- Die Control Unit steuert das TWI-Modul. Je nachdem, welcher Wert in das TWI Control Register geschrieben wird, führt die Control Unit eine Aktion durch und aktualisiert anschließend das TWI Status Register.
- Die Adress Match Unit überprüft beim Empfang einer Slave Adresse, ob die Adresse mit der übereinstimmt, die im TWI Adress Register gespeichert ist. Wenn dies der Fall ist, so wurde der ATmega 16 als Slave adressiert.
- Der Bit Rate Generator generiert, wenn die TWI-Einheit als Master fungiert, den Takt auf der SCL-Leitung. Die Taktrate hängt, wie man in der Formel auf der nächsten Seite sehen kann, vom CPU-Takt des ATmega 16, dem Wert des TWI Bit Rate Registes und einem Prescaler ab.

Die SCL- und SDA-Pins gleichen geringe Spannungsschwankungen durch einen Spike Filter aus und haben einen Slew Rate Limiter, der dafür sorgt, dass die Anstiegs- und Abfallszeiten der Signale der I^2C-Spezifikation entsprechen (ca. 20ns - 300ns).

3.2 Register und Benutzung der TWI-Einheit

Die TWI-Einheit des ATmega enthält die folgenden 5 Register:

Register	Bezeichnung
TWBR	TWI Bit Rate Register
TWCR	TWI Control Register
TWSR	TWI Status Register
TWDR	TWI Data Register
TWAR	TWI Adress Register

TWI Bit Rate Register

Bit	7	6	5	4	3	2	1	0	
	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Das TWI Bit Rate Register bestimmt die Taktfrequenz von SCL nach folgender Formel:

$$\text{TaktfrequenzSCL} = \frac{\text{TaktfrequenzCPU}}{16 + 2 \cdot (\text{TWBR}) + 4^{\text{TWPS}}}$$

TWI Control Register

Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Das TWI Control Register steuert die TWI-Einheit des ATmega 16. Welche Aktion am Bus ausgeführt wird, hängt vom Wert ab, der in dieses Register geschrieben wird.

Bedeutung der einzelnen Bits:

Bit	Bezeichnung	Beschreibung
TWINT	TWI Interrupt Flag	Dieses Bit wird immer dann gesetzt, wenn die TWI-Einheit mit einer Aktion fertig ist. Solange das Bit gesetzt ist, wird die TWI-Einheit angehalten. Um das Bit wieder zu löschen muss es mit einer logischen 1 beschrieben werden.
TWEA	TWI Enable Acknowledge Bit	Dieses Bit muss gesetzt werden, wenn nach Empfang eines Datenbytes oder nach dem Empfang der eigenen Slave-Adresse ein ACK-Bit gesendet werden soll. Wenn das Bit nicht gesetzt ist, kann die TWI-Einheit nicht als Slave funktionieren.
TWSTA	TWI START Condition Bit	Dieses Bit löst eine START-Condition aus, sobald der Bus frei ist.
TWSTO	TWI STOP Condition Bit	Dieses Bit löst eine STOP-Condition aus.
TWWC	TWI Write Collision Flag	Dieses Bit wird gesetzt, wenn auf das Datenregister TWDR geschrieben wird, während die TWI-Einheit noch beschäftigt ist.
TWIE	TWI Interrupt Enable	Wenn dieses Bit gesetzt ist, wird ein Interrupt ausgelöst, sobald TWINT gesetzt wird.

TWI Status Register

Bit	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

Das TWI Status Register enthält einerseits in den niedrigstwertigen zwei Bits den Wert eines Prescalers, der beim Betrieb als Master in die Taktrate von SCL eingeht, und andererseits einen Statuscode in den fünf höchstwertigen Bits. Dieser Statuscode gibt an, in welchem Zustand sich der Bus momentan befindet, d. h. ob die TWI Einheit Master oder Slave, Transmitter oder Receiver ist, welche Aktion zuletzt durchgeführt wurde und ob ein ACK oder NACK empfangen wurde.

Die Bedeutungen der Statuscodes und die Handlungsmöglichkeiten, die nach dem Auftreten eines bestimmten Codes gegeben sind, lassen sich im Datenblatt des ATmega 16 nachschlagen. Dort sind fünf Tabellen zu finden, je eine für jede Kombination von Master bzw. Slave, Transmitter bzw. Receiver sowie eine Tabelle mit sonstigen Statuscodes*.

Um die TWI-Einheit zu benutzen überprüft man also das Statusregister, wobei man darauf achten muss, die Prescaler-Bits auszumaskieren, und löst durch Schreiben ins TWI Control Register und eventuell durch Schreiben bzw. Lesen aus dem TWI Data Register eine passende Reaktion aus.

*für Busfehler oder wenn keine Statusinformation vorhanden ist

Hier ist als Beispiel die Tabelle der Statuscodes für den Betrieb im Master-Transmitter Modus:

Table 74. Status Codes for Master Transmitter Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
$08	A START condition has been transmitted	Load SLA+W	0	0	1	X	SLA+W will be transmitted; ACK or NOT ACK will be received
$10	A repeated START condition has been transmitted	Load SLA+W or	0	0	1	X	SLA+W will be transmitted; ACK or NOT ACK will be received
		Load SLA+R	0	0	1	X	SLA+R will be transmitted; Logic will switch to Master Receiver mode
$18	SLA+W has been transmitted; ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be Reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be Reset
$20	SLA+W has been transmitted; NOT ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
$28	Data byte has been transmitted; ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
$30	Data byte has been transmitted; NOT ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
$38	Arbitration lost in SLA+W or data bytes	No TWDR action or	0	0	1	X	Two-wire Serial Bus will be released and not addressed Slave mode entered
		No TWDR action	1	0	1	X	A START condition will be transmitted when the bus becomes free

TWI Data Register

Bit	7	6	5	4	3	2	1	0	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

Das TWI Data Register enthält das zu sendende bzw. das zuletzt empfangene Datenbyte. Es wird jedoch auch zum Senden der Slave-Adresse verwendet.

TWI Adress Register

Bit	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

Das TWI Adress Register enthält die Slave-Adresse des ATmega. Das niedrigstwertigste Bit (TWGCE = TWI General Call Recognition Enable Bit) des Registers bestimmt, ob der ATmega neben der Adresse in den sieben höchstwertigen Bits auch auf den „General Call“* reagiert.

*Adresse 0000000, durch die alle am Bus befindlichen Geräte adressiert werden

4 Beispiel

Als Beispiel habe ich ein Led-Lauflicht mit einem I²C-Portexpander (PCF8574) und einem ATmega 16 realisiert. Die Programmierung fand dabei in C statt und es wurden keine Interrupts verwendet. Der Mikrocontroller arbeitet im Master-Transmitter Modus, weil er eine Verbindung zum Portexpander aufbaut und Daten sendet.

In der Main-Funktion des Programms wird in einer Endlosschleife eine 8-Bit Variable ständig linksherum geschoben und der Wert nach der Verschiebung an den Portexpander übertragen. Dabei muss zwischen dem Verschieben und dem Senden eine „Zwangspause“ eingelegt werden, weil die LEDs sonst für das menschliche Auge zu schnell schalten und scheinbar dauerhaft leuchten.

```
void main(void){
  unsigned char i;
  i=0x01;
  while(1) {    // Hauptschleife
    if(i==0x80) i=0x01; // wenn am linken Ende, von vorne anfangen
    else i<<=1;         // ansonsten nach links schieben

    _delay_ms(150); // 150 ms warten

    sendeByte(i,ADRESSE); // Wert an LEDs übertragen
  }
}
```

In der Funktion, die das eigentliche Senden über den I²C-Bus übernimmt, werden zuerst die internen Pullups für SCL und SDA aktiviert. Wenn bereits externe Pullups vorhanden sind ist dieser Schritt zwar eigentlich nicht nötig, hat aber keine negativen Auswirkungen.
Anschließend wird eine START-Condition und die Slave-Adresse des Portexpanders mit dem Schreibbit gesendet. Wenn der Portexpander an dieser Stelle nicht reagiert, wird eine STOP-Condition ausgelöst und die Übertragung abgebrochen. Andernfalls hat der Portexpander ein ACK gesendet und der zu sendende Wert wird in das TWI Data Register (TWDR) geladen. Dieser Wert wird anschließend gesendet. Nachdem der Wert gesendet wurde, wird eine STOP-Condition ausgelöst und somit die Verbindung beendet.

```
int sendeByte(unsigned char datenbyte, unsigned char adresse){
  DDRC &= !((1<<DD0)|(1<<DD1));     // SDA und SCL lesen...
  PORTC= (1<<DD0)|(1<<DD1);         // ... und Pullups aktivieren

  // TWBR  =0x00;   // Schnellster Bitratenwert
  // TWSR  =0x00;   // Schalte Prescaler aus

  TWCR= ((1<<TWINT)|(1<<TWSTA)|(1<<TWEN));  // TWI aktivieren und START-Condition auslösen
  while(!(TWCR & (1<<TWINT)));              // warten auf START-Condition
  if((TWSR & 0xF8) != TW_START) return 0;   // Wenn keine START-Condition gesendet wurde, abbrechen

  TWDR=adresse & (0xFE) ;                   // Adresse mit Schreibbit (XXXXXXX0) in Datenregister laden ...
  TWCR= ((1<<TWINT)|(1<<TWEN));             // ... und senden
  while(!(TWCR & (1<<TWINT)));              // warten auf ACK oder NACK
  if((TWSR & 0xF8) != TW_MT_SLA_ACK) return twiStop(); // Wenn kein Slave reagiert, abbrechen

  TWDR=datenbyte;                           // Byte in Datenregister laden ...
  TWCR= ((1<<TWINT)|(1<<TWEN));             // ... und senden
  while(!(TWCR & (1<<TWINT)));              // warten auf ACK oder NACK
  if((TWSR & 0xF8) != TW_MT_DATA_ACK) return twiStop(); // Wenn nicht von Slave akzeptiert, abbrechen

  TWCR= ((1<<TWINT)|(1<<TWSTO)|(1<<TWEN));  // STOP-Condition auslösen
  return 1;
}
```

5 Quellen

- http://www.semiconductors.philips.com/acrobat_download/literature/9398/39340011.pdf
  I^2C-Bus Spezifikation 2.1
- http://www.atmel.com/dyn/resources/prod_documents/doc2466.pdf
  Datenblatt ATmega 16
- http://www.semiconductors.philips.com/products/interface_control/i2c/
  Philips-Seiten zum Thema I^2C
- http://www.standardics.philips.com/support/i2c/usage/
  I^2C-Einführung von Philips
- http://www.i2c-bus.org/
- http://de.wikipedia.org/w/index.php?title=I^2C&oldid=17718117
  Wikipedia-Artikel zum Thema I^2C