11 I2C Schnittstelle

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

11 I2C Schnittstelle

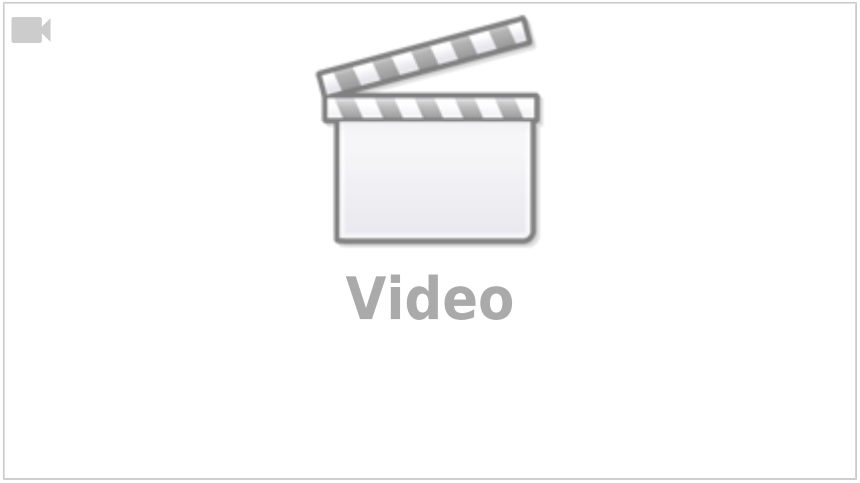
Ziele

Nach dieser Lektion sollten Sie:

1. wissen wie die Kommunikation zwischen I2C Master und Slave funktioniert

Im Video wird eine Library für die Kommunikation verwendet. Wir werden in untenstehenden Beispiel die Register selbst schreiben.

Video

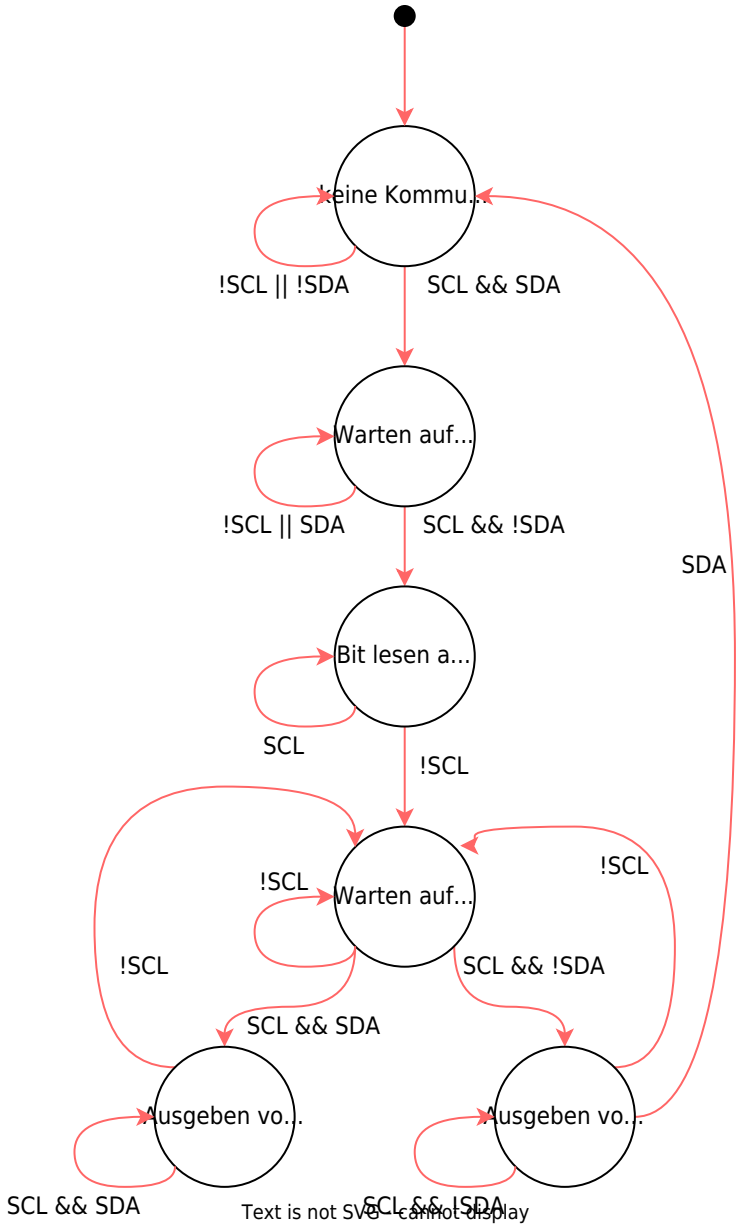


Statemachine für Datenpaket

Statemachine der I2C Kommunikation

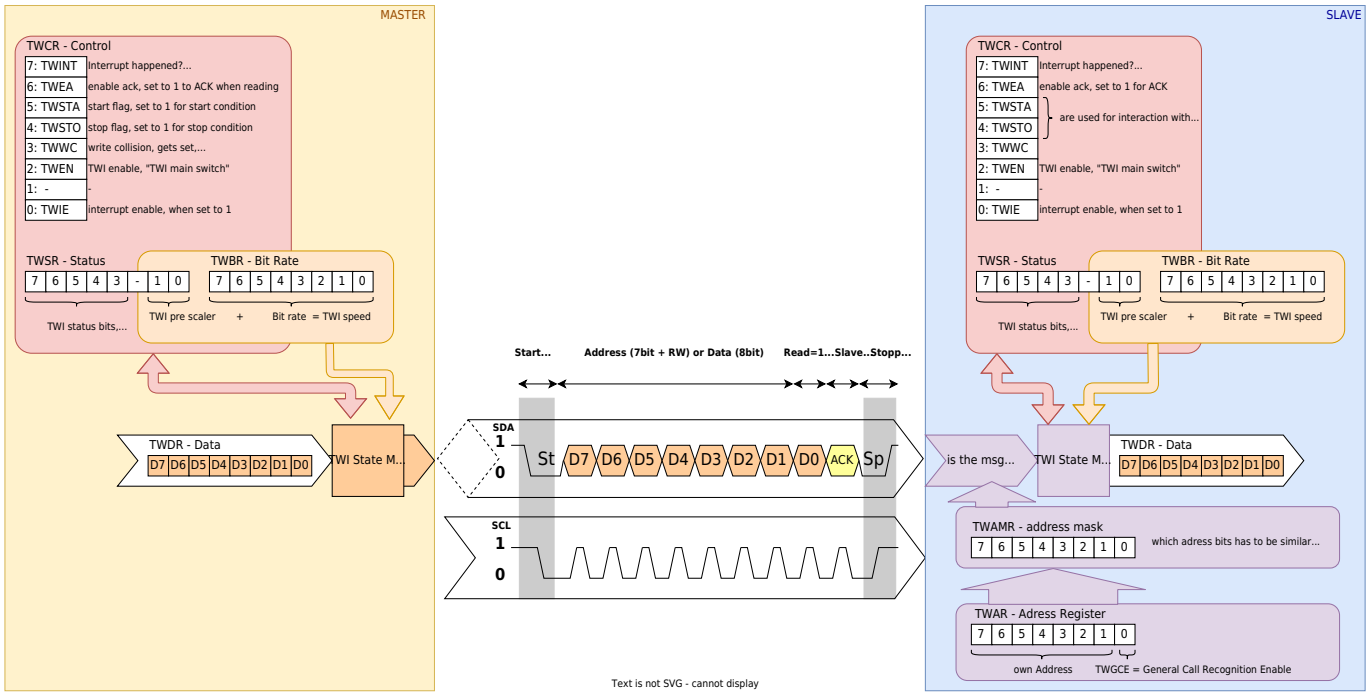
Statemachine der I2C Kommunikation

Fig. 1: Statemachine der I2C Kommunikation



I2C in Kürze und Zeitverlaufsdiagramm

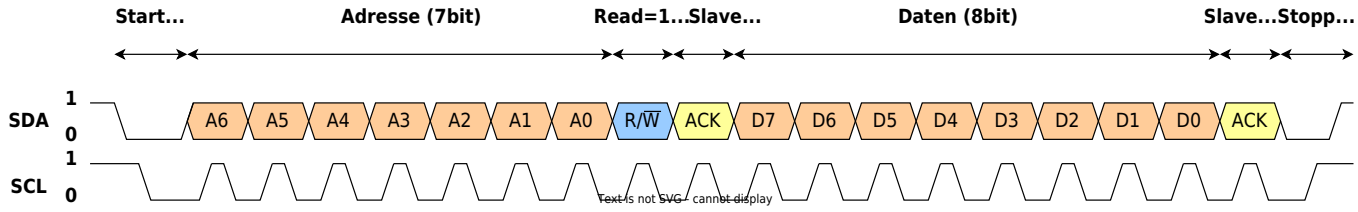
Fig. 2: Zusammenspiel der TWI-Register



Übertragung

Für die I2C Übertragung “trommelt” der Master-IC auf der Taktleitung (SCL). Bei jedem “Trommelschlag” (SCL=High), darf der Slave die Datenleitung (SDA) lesen.
D.h. während der Datenübertragung bleibt die Datenleitung bei SCL=High konstant.
Eine Flanke (=Signalwechsel) auf der Datenleitung während SCL=High definiert Beginn und Ende der Kommunikation.
Eine fallende Flanke auf SDA bei SCL=High stellt das Startbit dar, eine steigende Flanke das Stoppbit.
Läuft keine Kommunikation sind Daten- und Tankleitung auf High.

Fig. 3: Zeitverlaufsdiagramm der I2C Kommunikation

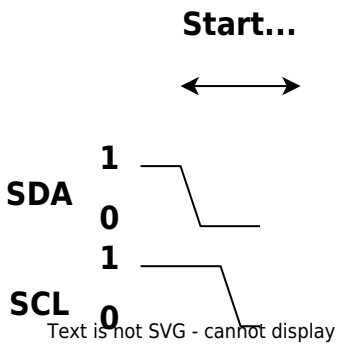


Startbedingung

Um die Übertragung zu beginnen muss die Startbedingung eingeleitet werden. Während SCL HIGH ist (a), geht SDA von HIGH auf LOW. Anschließend startet SCL mit LOW (b).

Für eine Startbedingung werden die Bits innerhalb des TWCR wie folgt gesetzt:

Fig. 4: Startbedingung

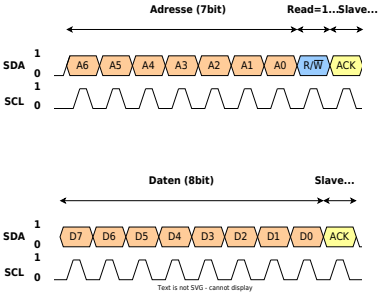



```
TWCR = (1<<TWINT)|(1<<TWEN);       // Setting
TWINT clears interupt flag
                                   // to set the
following state:
     | (1<<TWIE )                  // Enable TWI
Interrupt. --> nur wichtig, falls der Interrupt
geprüft wird!
     | (1<<TWSTA)|(0<<TWSTO);      // Initiate a
START condition.
```

Übertragung

Die entscheidende Voraussetzung für eine erfolgreiche Bitübertragung ist, dass sich der Zustand von SDA nur ändern darf solange SCL auf LOW ist. Allerdings ist der Zustand von SDA erst gültig, wenn SCL auf HIGH ist.

Fig. 5: Übertragung



Für die Übertragung eines Bytes muss TWDR und TWCR wie folgt gesetzt werden.
Zunächst wird die Übertragung der Adresse (SLA_W) betrachtet:

```
TWDR = SLA_W;                      // Load SLA_W
into TWDR
TWCR = (1<<TWINT)|(1<<TWEN);       // Setting
TWINT clears interupt flag
                                   // to start
transmission of address
```

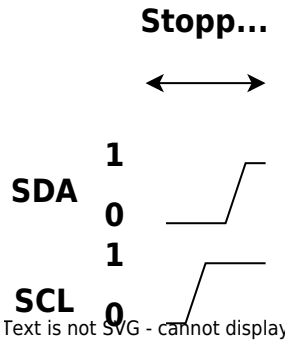
Die Daten (DATA) werden in gleicher Weise übertragen:

```
TWDR = DATA;                       // Load DATA
into TWDR
TWCR = (1<<TWINT)|(1<<TWEN);       // Setting
TWINT clears interupt flag
                                   // to start
transmission of address
```

Stoppbedingung

Die Stoppbedingung beendet die Übertragung. SCL geht auf HIGH (c), anschließend wechselt die SDA-Leitung von LOW nach HIGH (d).

Fig. 6: Stoppbedingung



Für eine Stoppbedingung werden die Bits innerhalb des TWCR wie folgt gesetzt:

```
TWCR = (1<<TWINT)|(1<<TWEN);       // Setting
TWINT clears interupt flag
                                   // to set the
following state:
     | (1<<TWIE )                  // Enable TWI
Interrupt. --> nur wichtig, falls der Interrupt
geprüft wird!
     | (0<<TWSTA)|(1<<TWSTO);      // Initiate a
STOP condition.
```

Software

wichtiger Hinweis

Die im Mikrocontroller fest verdrahtete State Machine von Microchip / ATMEL kann sich (selbst in der Simulation) durch ungünstiges Timing bzw. ungünstige Zustände aufhängen.
Diese Zustände lassen sich leicht dadurch beheben, dass eine erneute Initialisierung der I2C Register nach jedem Versenden / Empfangen durchgeführt wird.

einfache Anwendung - nur Polling

Im ersten Schritt ist im folgenden eine einfache Anwendung dargestellt.
Bei dieser werden Schalterstellungen vom Master an den Slave übertragen.

I. Vorarbeiten

Laden Sie die Datei simple_i2c.zip herunter und entpacken Sie diese.
In ihr finden Sie die Simulation als `simple_I2C.sim1` und das Microchip Solution File `simple_I2C.atsln`.

II. Analyse des fertigen Programms

1. Initialisieren des Programms
    1. Öffnen Sie SimulIDE und öffnen Sie dort mittels  die Datei `simple_I2C.sim1`
    2. In der Simulation finden Sie unten links 8 Dip-Schalter, welche durch Klick auf eine der acht kleinen Quadrate aktiviert (Wechsel auf grün)bzw deaktiviert werden kann.

3. Die hex Files sollten bereits in der simulation verlinkt sein, sodass diese nach dem Start auch lauffähig sei soll.
   Falls nicht finden Sie die hex Files unter `..\simple_I2C_Master\Debug` bzw `..\simple_I2C_Slave\Debug`.
4. Wenn Sie die Simulation starten, sehen Sie oben rechts im Logic Analyzer den Verlauf der Signale auf SDA (Daten) und SCL (Clock, also Takt).
5. Sobald die Stopp-Bedingung erfüllt ist (positive Flanke auf SDA, wenn SCL bereits High ist), wird die Simulation automatisch gestoppt.
   Dies geschieht, da im Logic Analyzer als Trigger `Ch1R & Ch2H`eingetragen wurde.

2. Sie können hier einige Dinge analysieren:
   1. Was passiert in den I2C Registern (TWAR, TWBR, TWCR, TWDR, TWSR) auf beiden Seiten?
   2. Was wird übertragen?
3. Das Programm zu diesem Hexfile soll nun erstellt werden

III. Code in Microchip Studio

I2C Master

```
/* ------------------------
---------------------------
----------------------
 Experiment 10:   I2C
Kommunikation
 =============
============================
===
 Dateiname      :
I2C_SimpleMaster.c
 Autoren        : Tim
Fischer       (Hochschule
Heilbronn, Fakultaet TE)
 Datum          : 18.11.2023
 Version        : 1.1
 Hardware       : Simulide
1.0.0 >R810
 Software       :
Entwicklungsumgebung:
Microchip Studio 7.0
                  C-
Compiler: AVR/GNU C Compiler
5.4.0
 Funktion       : einfacher
I2C Master, der
Schalterwerte überträgt
 Displayanzeige : keine
 Tastenfunktion : Die
Tastenstellung der Dip-
Schalter an Port B werden
```

```
Deklarationen
==================================
```

1. Hier wird die Frequenz des Quarz direkt eingestellt.
2. Weiterhin wird eine Konstante für die I2C Frequenz definiert

```
als Wert über I2C
weitergegeben.
                  Dabei
zählt ein gedrückter
Schalter (= hellgrün) als
logisches High Signal
 Jumperstellung : keine
 Fuses im uC    : keine
// ------------------------
---------------------------
----------------------*/
// Deklarationen
============================
============================
======
// Festlegung der
Quarzfrequenz
#define F_CPU 8000000UL
// CPU Frequenz von 8MHz
#define F_SCL   100000L
// Baudrate von 100 kHz

// Include von Header-
Dateien
#include <avr/interrupt.h>
#include <util/delay.h>

// Konstanten
#define SET_BIT(BYTE, BIT)
((BYTE) |=  (1 << (BIT))) //
Bit Zustand in Byte setzen
#define CLR_BIT(BYTE, BIT)
((BYTE) &= ~(1 << (BIT))) //
Bit Zustand in Byte loeschen
#define TGL_BIT(BYTE, BIT)
((BYTE) ^=  (1 << (BIT))) //
Bit Zustand in Byte wechseln
(toggle)

#define SET_ALL_PULLUPS
(0xFF)
// Konstante für die
Aktivierung der Pullup
#define INVERING_MASK
(0xFF)
// Konstante für die
invertierung der
Schalterstellungen
#define TWI_ADRESS
(0b0001010)
// Konstante für die I2C
```

3. Die Header-Dateien und die Bit-ändernden Makros entsprechen denen der letzten Programme.

4. Die weiteren Konstanten sind:
    1. Konstante für die Pullups
    2. Konstante für die Invertierung der Schalterstellungen
    3. Konstante für die I2C Adresse

5. Zwei globale Variablen beinhalten die I2C Adresse und die Daten

Hauptprogramm ==========

1. es werden zunächst die Pull-up Widerstände aller Pins an Port B aktiviert, um die Schalter einlesen zu können
2. in der Haupschleife läuft:
    1. Zu Beginn eine Iitialisierung der I2C Schnittstelle. Die wiederholte Initialisierung vermeidet Probleme der I2C Zustandsmaschine.
    2. Als nächstes wird die startende Flanke gesendet.
    3. Das erste Byte auf dem I2C Bus ist die Adressebyte. Dieses setzt sich aus der Adresse und einem Bit zusammen, welches angibt, ob der Slave nur zuhören (0) oder antworten (1) soll.
    4. Nun wird der Zustand das Port B als I2C Daten eingelesen.
    5. Diese werden per I2C übertragen.
    6. Und zum Schluss wird das Stopp Bit gesendet

I2C Initialisierung ==========

1. Durch das Zurücksetzen der Bits TWPS0 und TWPS1 wird kein Prescaler (bzw. einer von 1) gewählt
2. Auch das Kontrollregister wird zurückgesetzt

```
Adresse

//Funktionsprototypen
void I2C_Init();
void I2C_transmitStart();
void
I2C_transmitDataOrAddress(ch
ar Data);
void I2C_transmitStop();

uint8_t TWI_Address =
TWI_ADRESS;
// Variable der I2C Adresse
uint8_t TWI_Data    =
0b00000000;
// Variable für die I2C
Daten


int main(void)
{
    PORTB = SET_ALL_PULLUPS;
// Pull-up Widerstände an
Port B aktivieren
    while (1)
    {
        I2C_Init();
// Initialisierung von TWI
anstoßen
        I2C_transmitStart();
// Startbit schreiben
I2C_transmitDataOrAddress((T
WI_Address<<1) + 0);//
Adresse senden: LSB = 0,
zeigt Slave an, dass dieser
nur empfangen soll
        TWI_Data =
PINB^INVERING_MASK;
// Lese Tasterstellungen
ein. Invertiere jedes Bit
I2C_transmitDataOrAddress(TW
I_Data);           // Daten
senden
        I2C_transmitStop();
// Stoppbit schreiben
        _delay_us(1);
// erst durch den Delay ist
ein Triggern im Simulide
möglich
    }
}
```

3. im Bitraten Register wird die I2C Frequenz eingestellt

I2C Startbit senden ==========

1. Es soll die I2C Schnittstelle aktiviert (TWEN setzen) und das Startbit gesendet (TWSTA setzen) werden. Das Schreiben einer 1 in TWINT löscht dieses Interruptflag
2. Nach dem Ändern des Konrollregisters muss die Abarbeitung abgewartet werden. Dies ist daran zu erkennen, das TWINT gleich 1 wird

I2C Adressbyte/Daten senden ==========

1. Bevor Daten übertragen werden sollen, müssen diese erst in das I2C Datenregister geschrieben werden.
2. Wieder die I2C Schnittstelle aktivieren (TWEN setzen) und das Inrerruptflag löschen (TWINT setzen)
3. auch hier muss wieder die Abarbeitung abgewartet werden

I2C Stoppbit senden ==========

1. Das Stoppbit wird wieder im Kontrollregister aktiviert
2. Hier ist kein Warten notwendig

```
//////////////////////////////
//////////////
// I2C Initialisierung
//////////////////////////////
//////////////
void I2C_Init()
{
    CLR_BIT(TWSR, TWPS0);
// Es wird kein Prescaler
verwendet:
    CLR_BIT(TWSR, TWPS1);
//        Deshalb TWPS0 = 0
und TWPS1 = 0
    TWCR = 0;
// Control Register
zurücksetzen
    TWBR =
((F_CPU/F_SCL)-16)/2;
// die Bitrate wird mittels
CPU Frequenz und Serial
Clock Frequenz ermittelt
}

//////////////////////////////
//////////////
// I2C Startbit senden
//////////////////////////////
//////////////
void I2C_transmitStart()
{
    TWCR =
(1<<TWINT)|(1<<TWEN)|(1<<TWS
TA);               // TWSTA
= Startbit aktivieren, TWEN
= TWI starten (ENable),
TWINT = Interrupt bit
löschen (durch beschreiben)
    while (!(TWCR &
(1<<TWINT)));
// warten bis Übertragung
erfolgreich, Trigger ist
hier das Setzen von TWINT
}



//////////////////////////////
//////////////
// I2C Adressbyte/Daten
senden
//////////////////////////////
//////////////
```

```
void
I2C_transmitDataOrAddress(ch
ar Data)
{
    TWDR = Data;
// Data Variabel in Daten
Register schreiben
    TWCR =
(1<<TWINT)|(1<<TWEN);
// TWEN = TWI starten
(ENable), TWINT = Interrupt
bit löschen (durch setzen)
    while (!(TWCR &
(1<<TWINT)));
// warten bis Übertragung
erfolgreich, Trigger ist
hier das Setzen von TWINT
}

////////////////////////////
/////////////
// I2C Stoppbit senden
////////////////////////////
/////////////
void I2C_transmitStop()
{
TWCR=(1<<TWINT)|(1<<TWEN)|(1
<<TWSTO);             //
TWSTO = Stopptbit
aktivieren, TWEN = TWI
starten (ENable), TWINT =
Interrupt bit löschen (durch
setzen)
}
```

I2C Slave

```
/* ------------------------
---------------------------
----------------------
 Experiment 10:   I2C
Kommunikation
 =============
============================
===
 Dateiname      :
I2C_SimpleSlave.c
```

```
 Autoren        : Tim
Fischer       (Hochschule
Heilbronn, Fakultaet TE)
 Datum          : 18.11.2023
 Version        : 1.0
 Hardware       :  Simulide
0.5.16-RC5
 Software       :
Entwicklungsumgebung:
Microchip Studio 7.0
                  C-
Compiler: AVR/GNU C Compiler
5.4.0
 Funktion       : TBD

 Displayanzeige : keine

 Tastenfunktion : Die
Tastenstellung der Dip-
Schalter an Port B werden
als Wert über I2C
weitergegeben.
                 Dabei
zählt ein gedrückter
Schalter (= hellgrün) als
logisches High Signal

 Jumperstellung :  keine

 Fuses im uC    : keine
// ------------------------
----------------------------
----------------------*/
// Deklarationen
============================
============================
======
// Festlegung der
Quarzfrequenz
#define F_CPU 8000000UL
// CPU Frequenz von 8MHz
#define F_SCL   100000L
// Baudrate von 100 kHz

// Include von Header-
Dateien
#include <avr/interrupt.h>

// Konstanten
#define SET_BIT(BYTE, BIT)
((BYTE) |=  (1 << (BIT))) //
```

Deklarationen ==========

1. Hier wird die Frequenz des Quarz direkt eingestellt.
2. Weiterhin wird eine Konstante für die I2C Frequenz definiert

3. Die Header-Dateien und die Bit-ändernden Makros entsprechen denen der letzten Programme.

4. Die weiteren Konstanten sind:
    1. Konstante für die das Setzen der Pins als Ausgang
    2. Konstante für die I2C Adresse
    3. Konstante für die Maskierung der Adresse

5. Zwei globale Variablen beinhalten die I2C Adresse und die Maske für die eingelesene Adresse

Hauptprogramm ==========

1. Es werden alle Pins an Port B als Ausgang geschalten
2. Die I2C Adresse, auf welche der Slave hört, wird gesetzt

3. In der Haupschleife läuft nur das Setzen des Port B anhand der eingelesenen I2C Daten

Setzen der I2C Adresse auf die der Slave hört ==========

1. Die Adresse wird in das Adressbyte geschrieben.
2. Die Adressmaske wird in das Maskenbyte geschrieben. Durch die Maske lässt sich ein Slave auch mit verschiedenen Adressen

```
Bit Zustand in Byte setzen
#define CLR_BIT(BYTE, BIT)
((BYTE) &= ~(1 << (BIT))) //
Bit Zustand in Byte loeschen
#define TGL_BIT(BYTE, BIT)
((BYTE) ^=  (1 << (BIT))) //
Bit Zustand in Byte wechseln
(toggle)

#define SET_ALL_TO_OUT
(0xFF)
// Konstante für die
Aktivierung der Port
Ausgänge
#define TWI_ADRESS
(0b0001010)
// Konstante für die I2C
Adresse
#define TWI_ADRESS_MASK
(0b0001010)
// Konstante für die I2C
Adress-Maske

//Funktionsprototypen
void I2C_Init();
void I2C_setAddress(char
Address);
char I2C_readData();

uint8_t TWI_Address
= TWI_ADRESS;
// Variable der I2C Adresse
uint8_t TWI_AddressMask
= TWI_ADRESS_MASK;
// Variable Zum maskieren
der eingehenden Adresse

int main(void)
{
    DDRB= SET_ALL_TO_OUT;
// Auf DDRC die Daten
ausgeben
I2C_setAddress(TWI_Address);
// eigene Adresse setzen
    while (1)
    {
        PORTB =
I2C_readData();
// Daten an PortC ausgeben
    }
}
```

ansprechen, da die Maske angibt, welche Bits nicht berücksichtigt werden sollen (mit 0)
3. Im Kontrollregister wird die “Rückbestätigung” and den Master (Acknowledge) aktiviert (TWEA setzen) und das I2C Modul aktiviert (TWEN setzen). Hier darf TWINT nicht geändert werden!

Auslesen der übermittelten Daten
==========

1. Hier wird solange gewartet, bis I2C Daten vorliegen
2. Nach dem Ändern des Konrollregisters muss die Abarbeitung abgewartet werden. Dies ist daran zu erkennen, das TWINT gleich 1 wird

```
////////////////////////////
//////////////////////
// Setzen der I2C Adresse
auf die der Slave hört
////////////////////////////
//////////////////////
void I2C_setAddress(char
Address)
{
    TWAR = (Address<<1);
// Adresse in das
Pseudoregister schreiben
    TWAMR= TWI_AddressMask;
// Adressmaske in das
Pseudoregister schreiben
    TWCR =
(1<<TWEA)|(1<<TWEN);
// Enable Ack, Enable
Interupt und Enable TIW
}

////////////////////////////
//////////////////////
// Auslesen der
übermittelten Daten
////////////////////////////
//////////////////////
char I2C_readData()
{
    while (!(TWCR &
(1<<TWINT)));
// warte solange bis TWINT
gesetzt ist
    return TWDR;
// übermittle Daten
}
```

komplexere Anwendung - mit Interrupt

Als Beispiel wurde hier die Temperaturmessung aus Lektion 8 gewählt.
Das Projekt und die Simulation ist hier zu finden ad_wandler_i2c.zip

Bitte nutzen Sie diese als Vorlage, wen Sie eine I2C Schnittstelle implementieren wollen. Da in diesem Programmstand alles über Interrupts läuft, können auch weitere Funktionen abgearbeitet werden.

weiterführende Unterlagen

Die detaillierte Beschreibung zu I2C findet sich in des I2C Bus Specification and User Manual des Herstellers (ehemls Phillips nun NXP).

Bibliotheken

- TWI Module as I2C Master (AVR315):
  - Application Note: doc2564.pdf
  - Library und Beispielcode: avr315.zip
- TWI Module as I2C Slave (AVR311):
  - Application Note: doc2565.pdf
  - Library und Beispielcode: avr311.zip
- alternative und schlanke Implementierung des Slaves von The Gouger (GitHub)
  (Kopie vom 16.01.22: avr-i2c-slave-master.zip)

Beispiele

- In Simulide ist eine Umsetzung von Software I2C zu finden:
  `…\share\simulide\examples\Arduino\sofware_i2c_lcd\i2c_lcd-arduino`
- Eine vollständige Implementierung des Codes für den I2C Master ist in der Library von Peter Fleury zu finden: library, Dokumentation
- Eine Implementierung eines Software I2C Slave, also eines I2C an einem beliebigen Pin durch Bitmaipulation, habe ich bereitgestellt. Es soll jedoch stets bevorzugt werden die vorhandenen Hardware I2C zu nutzen.

Beschreibung

Die “Microchip University” hat auch eine schöne Einführung in I2C. Hier wird aber nicht die Implementierung in Microchip Studio auf einem AVR-Chip gezeigt, sondern in MPLAB X auf einem PIC. D.h. der Code ist nicht direkt übertragbar.