

11 I2C Schnittstelle

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

10 I2C Schnittstelle

Ziele

Video

Statemachine für Datenpaket

Statemachine der I2C Kommunikation

I2C in Kürze und Zeitverlaufdiagramm

Startbedingung

Übertragung

Stoppbedingung

Software

einfache Anwendung

TWI Master

TWI Slave

komplexere Anwendung

Bibliotheken

Beispiele

weiterführende Unterlagen

2

2

2

2

3

4

5

6

6

6

7

15

17

17

17

17

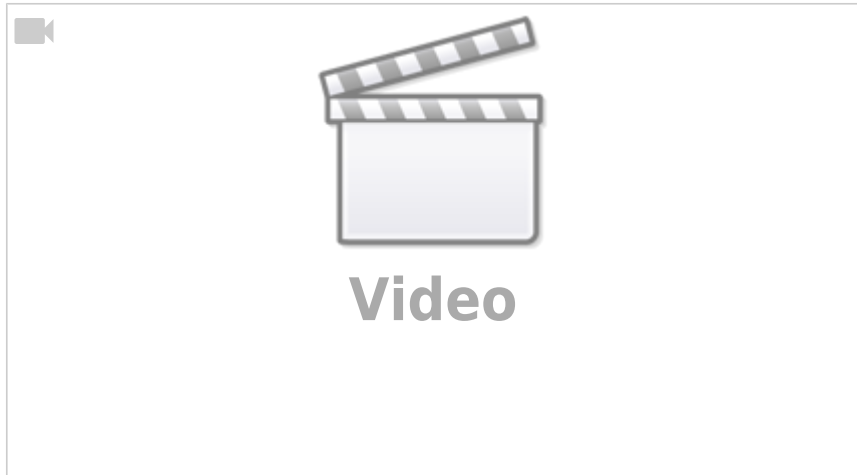
10 I2C Schnittstelle

Ziele

Nach dieser Lektion sollten Sie:

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

Video

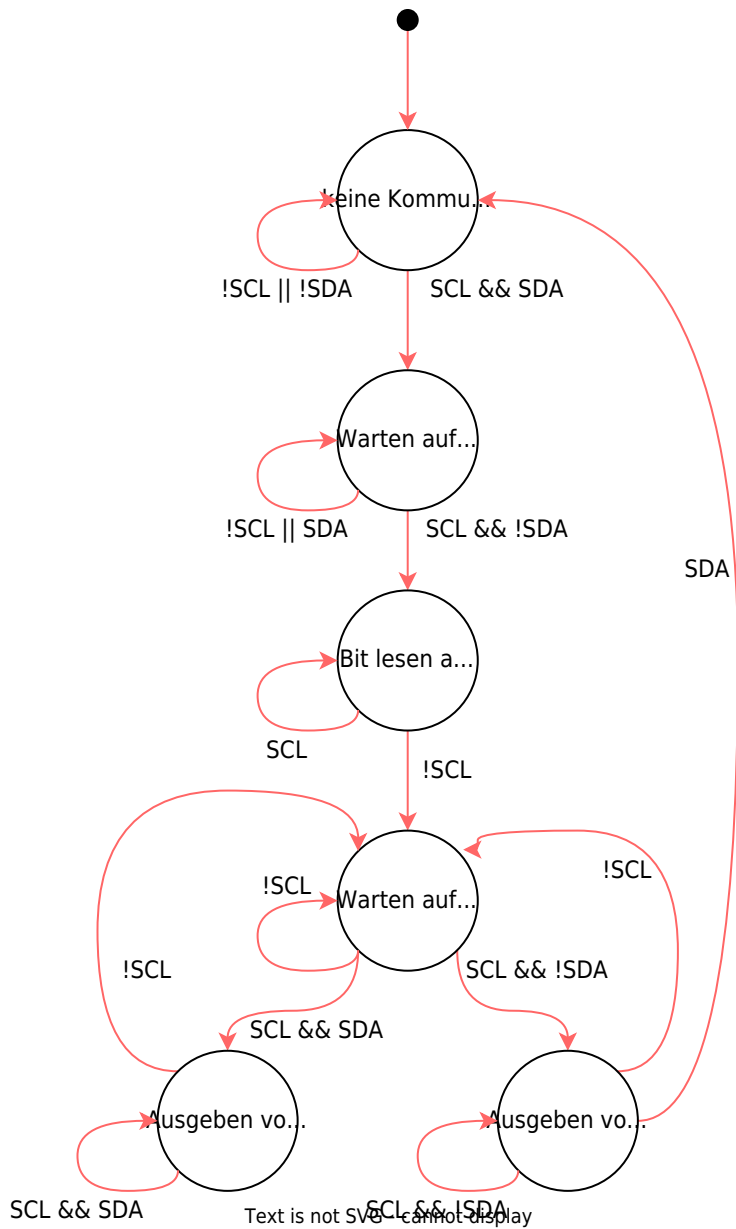


Statemachine für Datenpaket

[Statemachine der I2C Kommunikation](#)

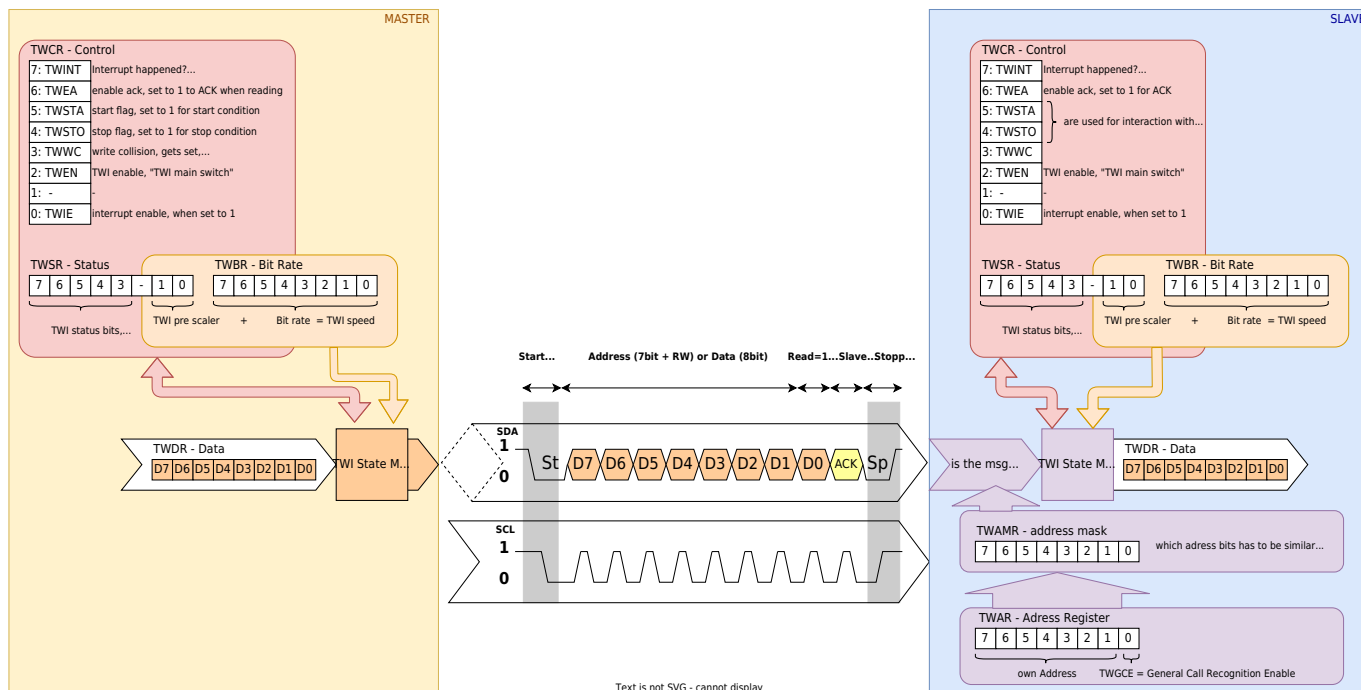
Statemachine der I2C Kommunikation

Fig. 1: Statemachine der I2C Kommunikation



I2C in Kürze und Zeitverlaufdiagramm

Fig. 2: Zusammenspiel der TWI-Register



Übertragung

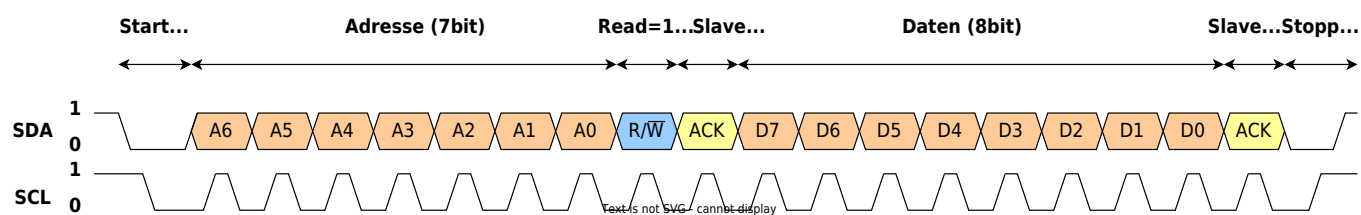
Für die I2C Übertragung "trommelt" der Master-IC auf der Taktleitung (SCL). Bei jedem "Trommelschlag" (SCL=High), darf der Slave die Datenleitung (SDA) lesen.

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 Stopbit. Läuft keine Kommunikation sind Daten- und Taktleitung auf High.

Fig. 3: Zeitverlaufsdiagramm der I2C Kommunikation

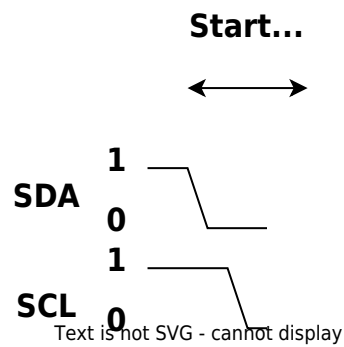


Startbedingung

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 interrupt flag              // to set the
                                         // following state:
      | (1<<TWIE )                       // Enable TWI
Interrupt. --> nur wichtig, falls der Interrupt
geprüft wird!
      | (1<<TWSTA)|(0<<TWST0);           // Initiate a
START condition.

```

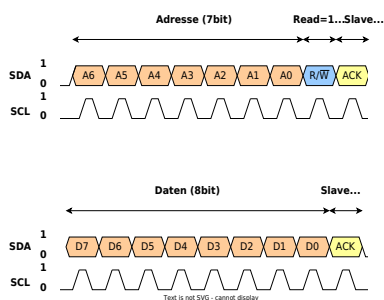
Übertragung

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.

Für die Übertragung eines Bytes muss TWDR und TWCR wie folgt gesetzt werden.

Zunächst wird die Übertragung der Adresse (SLA_W) betrachtet:

Fig. 5: Übertragung



```

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

```

Die Daten (DATA) werden in gleicher Weise übertragen:

```

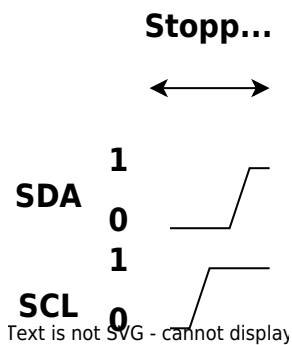
TWDR = DATA;                           // Load DATA
into TWDR
TWCR = (1<<TWINT)|(1<<TWEN);           // Setting
TWINT clears interrupt flag              // to start
                                         // transmission of address

```

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 interrupt flag                // to set the
                                           // following state:
      | (1<<TWIE )                        // Enable TWI
Interrupt. --> nur wichtig, falls der Interrupt
geprüft wird!
      | (0<<TWSTA)|(1<<TWST0);           // Initiate a
STOP condition.
```

Software

einfache Anwendung

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 & Ch2Heingetragen 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

TWI Master

```

/* -----
-----
-----
Experiment 10:  I2C
Kommunikation
=====
=====
===
Dateiname      :
I2C_SimpleMaster.c
Autoren        : Tim
Fischer        (Hochschule
Heilbronn, Fakultät TE)
Datum          : 18.11.2023
Version        : 1.1
Hardware       : Simulide
1.0.0 >R810
Software       :
Entwicklungsumgebung:
AtmelStudio 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
als Wert über I2C
weitergegeben.
Dabei
zählt ein gedrückter
Schalter (= hellgrün) als
logisches High Signal

```

Deklarationen

=====

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

```

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
Adresse

//Funktionsprototypen
void I2C_Init();
void I2C_transmitStart();
void

```

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 wird zunächst die Pull-up Widerstände aktiviert, um die Schalter einlesen zu können
2. in der Hauptschleife läuft:
 1. Zu Beginn eine Initialisierung 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 des Port B als I2C Daten eingelesen.
 5. Diese werden per I2C übertragen.
 6. Und zum Schluss wird das Stopp Bit gesendet

I2C Initialisierung

=====

1. im Statusregister werden die Bits TWSR mit dem Befehl ISR() wird eine Interrupt Service Routine für den OVerFlow Interrupt für TIMER2 angelegt.
2. Der Überlauf-Interrupt durch den Timer2 wird erst bei Überlauf des 8-Bit Wert ausgeführt. Auch hier ergibt sich durch den Prescaler und Modus (TCCR2A und TCCR2B) eine Periode von

```

I2C_transmitDataOrAddress(char Data);
void I2C_transmitStop();

uint8_t TWI_Address =
TWI_ADDRESS;
// 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((TWI_Address<<1) + 0); //
        Adresse senden: LSB = 0,
        zeigt Slave an, dass dieser
        nur empfangen soll
        TWI_Data =
        PINB^INVERTING_MASK;
        // Lese Tasterstellungen
        ein. Invertiere jedes Bit
        I2C_transmitDataOrAddress(TWI_Data); // Daten
        senden
        I2C_transmitStop();
        // Stoppbit schreiben
        // ggf. kann am Ende der
        Übertragung ein _delay_us(1)
        zur Synchronisierung helfen
    }
}

////////////////////////////////////
////////////////////////////////////
// I2C Initialisierung
////////////////////////////////////
////////////////////////////////////
void I2C_Init()

```

$T_{\text{ISR}} = 0,16 \text{ ms}$.

3. Auch das Kontrollregister wird zurückgesetzt
4. 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,

```
{
    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
```

müssen diese erst in das I2C Datenregister geschrieben werden.

2. Wieder die I2C Schnittstelle aktivieren (TWEN setzen) und das Inerruptflag 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

/* Teilprogramm 1: Blinkende LED =====

Hier ist das Programm der [Blinking LED](#) etwas angepasst eingefügt.

1. Zunächst wird ein Unterprogramm zur Anzeige des Displays aufgerufen
2. SET_BIT(DDRB, DDB0) wandelt den Anschluss B0 in einen Ausgang um
3. Die Schleife wird solange ausgeführt, bis die Flanke des Schalters 1 über sw1_slope erkannt wurde
4. Beim Aktivieren der LED wird auch auf dem

```

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
    <<TWST0);          //
    TWST0 = Stopptbit
    aktivieren, TWEN = TWI
    starten (ENable), TWINT =
    Interrupt bit löschen (durch
    setzen)
}

```

Display eine 1 geschrieben.

5. Nach einer Sekunde wird die LED ausgeschaltet und auf dem Display eine geschrieben.
6. Nach Beendigung der Schleife werden alle Flanken gelöscht. Damit wird verhindert, dass beim Aufruf des Hauptmenus sofort ein Sprung in ein Unterprogramm ausgeführt wird.

/* Teilprogramm 2: Soundgenerierung
====

Hier ist das Programm [Sound und Timer](#) etwas angepasst eingefügt.


```
/* Teilprogramm 3: Logische Funktionen  
====
```

Hier ist das Programm [Logische Funktionen](#) etwas angepasst eingefügt.

1. Durch den Anschluss des Tasters zwischen Port und Masse erzeugt ein geschlossener ein LOW Signal (logisch 0). Hier sollen aber nun der Tastendruck dem Wert HIGH (logisch 1) entsprechen. Aus diesem Grund sind die Tasterwerte in den Bedingungen negiert, z.B. `(!sw3_alt)&&(!sw4_alt)`

/* Teilprogramm 4: Up-Down-Counter ====

Hier ist das Programm [Up/Down Counter](#) etwas angepasst eingefügt.

1. Im wesentlichen gleicht das Programm dem bereits bekanntem. Es kann aber auf die bereits berechnete Flanken `sw2_slope` bis `sw4_slope` zurückgegriffen.

Auswahl im Hauptmenu ermitteln

=====

1. Je nach gedrückter Taste wird hier die Variable `modus` gesetzt

TWI Slave

TWI Slave

```

/* -----
-----

Experiment 10:   I2C Kommunikation
=====

Dateiname   : I2C_SimpleMaster.c

Autoren     : Tim Fischer           (Hochschule Heilbronn, Fakultät
T1)

Datum       : 23.06.2021

Version     : 1.0
Hardware:    Simulide 0.5.16-RC5

Software:    Entwicklungsumgebung: AtmelStudio 7.0
              C-Compiler: AVR/GNU C Compiler 5.4.0

Funktion:    tbd

// -----
-----*/

// Deklarationen
=====

// Festlegung der Quarzfrequenz
#define F_CPU 16000000UL
#define F_SCL 10000L //100 kHz

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

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

uint8_t TWI_Address      = 0b0001010;
uint8_t TWI_AddressMask  = 0b11111110;

int main(void)
{
    DDRD= 0xFF; // Auf DDRC die
Daten ausgeben
    I2C_setAddress(TWI_Address); // eigene Adresse
setzen
    while (1)
    {
        PORTD = I2C_readData(); // Daten an PortC
ausgeben
    }
}

////////////////////////////////////////
// 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

Als Beispiel wurde die Temperaturmessung gewählt

- [twi_slave_master.zip](#)

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

- Simulide: ...\\share\\simulide\\examples\\Arduino\\software_i2c_lcd\\i2c_lcd-arduino (hierbei wird Software I2C eingesetzt)
- Software I2C:
 - Library von Peter Fleury: [library](#), [Dokumentation](#)
 - [Software I2C Slave](#)

weiterführende Unterlagen

- 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.

From:
<https://wiki.mexle.org/> - **MEXLE Wiki**

Permanent link:
https://wiki.mexle.org/microcontrollertechnik/11_i2c_schnittstelle?rev=1700357262

Last update: **2023/11/19 02:27**

