/\*#################################################################################################

 Title : TWI SLave

 Author : Martin Junghans <jtronics@gmx.de>

 Hompage : www.jtronics.de

 Software: AVR-GCC / Programmers Notpad 2

 License : GNU General Public License

 Aufgabe :

 Betrieb eines AVRs mit Hardware-TWI-Schnittstelle als Slave.

 Zu Beginn muss init\_twi\_slave mit der gewünschten Slave-Adresse als Parameter aufgerufen werden.

 Der Datenaustausch mit dem Master erfolgt über die Buffer rxbuffer und txbuffer, auf die von Master und Slave zugegriffen werden kann.

 rxbuffer und txbuffer sind globale Variablen (Array aus uint8\_t).

 Ablauf:

 Die Ansteuerung des rxbuffers, in den der Master schreiben kann, erfolgt ähnlich wie bei einem normalen I2C-EEPROM.

 Man sendet zunächst die Bufferposition, an die man schreiben will, und dann die Daten. Die Bufferposition wird

 automatisch hochgezählt, sodass man mehrere Datenbytes hintereinander schreiben kann, ohne jedesmal

 die Bufferadresse zu schreiben.

 Um den txbuffer vom Master aus zu lesen, überträgt man zunächst in einem Schreibzugriff die gewünschte Bufferposition und

 liest dann nach einem repeated start die Daten aus. Die Bufferposition wird automatisch hochgezählt, sodass man mehrere

 Datenbytes hintereinander lesen kann, ohne jedesmal die Bufferposition zu schreiben.

 Abgefangene Fehlbedienung durch den Master:

 - Lesen über die Grenze des txbuffers hinaus

 - Schreiben über die Grenzen des rxbuffers hinaus

 - Angabe einer ungültigen Schreib/Lese-Adresse

 - Lesezuggriff, ohne vorher Leseadresse geschrieben zu haben

 LICENSE:

 This program is free software; you can redistribute it and/or modify

 it under the terms of the GNU General Public License as published by

 the Free Software Foundation; either version 2 of the License, or

 any later version.

 This program is distributed in the hope that it will be useful,

 but WITHOUT ANY WARRANTY; without even the implied warranty of

 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

 GNU General Public License for more details.

//#################################################################################################\*/

#include <util/twi.h> // Bezeichnungen für Statuscodes in TWSR

#include <avr/interrupt.h> // behandlung der Interrupts

#include <stdint.h> // definiert Datentyp uint8\_t

#include "twislave.h"

//#################################### Macros

//ACK nach empfangenen Daten senden/ ACK nach gesendeten Daten erwarten

#define TWCR\_ACK TWCR = (1<<TWEN)|(1<<TWIE)|(1<<TWINT)|(1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|(0<<TWWC);

//NACK nach empfangenen Daten senden/ NACK nach gesendeten Daten erwarten

#define TWCR\_NACK TWCR = (1<<TWEN)|(1<<TWIE)|(1<<TWINT)|(0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|(0<<TWWC);

//switched to the non adressed slave mode...

#define TWCR\_RESET TWCR = (1<<TWEN)|(1<<TWIE)|(1<<TWINT)|(1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|(0<<TWWC);

//########################################################################################## init\_twi\_slave

void init\_twi\_slave(*uint8\_t* adr)

{

 TWAR= adr; //Adresse setzen

 TWCR &= ~(1<<TWSTA)|(1<<TWSTO);

 TWCR|= (1<<TWEA) | (1<<TWEN)|(1<<TWIE);

 buffer\_adr=0xFF;

 sei();

}

//########################################################################################## ISR (TWI\_vect)

//ISR, die bei einem Ereignis auf dem Bus ausgelöst wird. Im Register TWSR befindet sich dann

//ein Statuscode, anhand dessen die Situation festgestellt werden kann.

ISR (TWI\_vect)

{

 *uint8\_t* data=0;

 switch (*TW\_STATUS*) // TWI-Statusregister prüfen und nötige Aktion bestimmen

 {

 case *TW\_SR\_SLA\_ACK*: // 0x60 Slave Receiver, wurde adressiert

 TWCR\_ACK; // nächstes Datenbyte empfangen, ACK danach

 buffer\_adr=0xFF; // Bufferposition ist undefiniert

 break;

 case *TW\_SR\_DATA\_ACK*: // 0x80 Slave Receiver,Daten empfangen

 data=TWDR; // Empfangene Daten auslesen

 if (buffer\_adr == 0xFF) // erster Zugriff, Bufferposition setzen

 {

 if(data<=buffer\_size)// Kontrolle ob gewünschte Adresse im erlaubten bereich

 {

 buffer\_adr= data; // Bufferposition wie adressiert setzen

 }

 else

 {

 buffer\_adr=0; // Adresse auf Null setzen. Ist das sinnvoll?

 }

 TWCR\_ACK; // nächstes Datenbyte empfangen, ACK danach, um nächstes Byte anzufordern

 }

 else // weiterer Zugriff, Daten empfangen

 {

 rxbuffer[buffer\_adr]=data; // Daten in Buffer schreiben

 buffer\_adr++; // Buffer-Adresse weiterzählen für nächsten Schreibzugriff

 if(buffer\_adr<(buffer\_size-1)) // im Buffer ist noch Platz für mehr als ein Byte

 {

 TWCR\_ACK; // nächstes Datenbyte empfangen, ACK danach, um nächstes Byte anzufordern

 }

 else // es kann nur noch ein Byte kommen, dann ist der Buffer voll

 {

 TWCR\_NACK; // letztes Byte lesen, dann NACK, um vollen Buffer zu signaliseren

 }

 }

break;

case *TW\_ST\_SLA\_ACK*: //

case *TW\_ST\_DATA\_ACK*: // 0xB8 Slave Transmitter, weitere Daten wurden angefordert

if (buffer\_adr == 0xFF) // zuvor keine Leseadresse angegeben!

{

buffer\_adr=0;

}

TWDR = txbuffer[buffer\_adr]; // Datenbyte senden

buffer\_adr++; // bufferadresse für nächstes Byte weiterzählen

if(buffer\_adr<(buffer\_size-1)) // im Buffer ist mehr als ein Byte, das gesendet werden kann

{

TWCR\_ACK; // nächstes Byte senden, danach ACK erwarten

}

else

{

TWCR\_NACK; // letztes Byte senden, danach NACK erwarten

}

break;

case *TW\_ST\_DATA\_NACK*: // 0xC0 Keine Daten mehr gefordert

case *TW\_SR\_DATA\_NACK*: // 0x88

case *TW\_ST\_LAST\_DATA*: // 0xC8 Last data byte in TWDR has been transmitted (TWEA = “0”); ACK has been received

case *TW\_SR\_STOP*: // 0xA0 STOP empfangen

default:

TWCR\_RESET; // Übertragung beenden, warten bis zur nächsten Adressierung

break;

} //end.switch (TW\_STATUS)

} //end.ISR(TWI\_vect)