AN701.65-116

für

LI-Ion Ladeschaltung Nμ701.65

Application Note 116: Programmierbeispiel in C

Rev. 0.1

Inhaltsverzeichnis

1 Zweck

Der Nµ701.65 lässt sich über eine serielle Schnittstelle prüfen. (Siehe AN701.65-115) Dazu ist der Zugriff auf interne Parameter und Funktionen vorgesehen. Die implementierten Funktionen und Befehle können z. B. für den Baugruppentest über einen externen Controller genutzt werden. Ebenfalls kann die Schnittstelle in der Qualitätssicherung und im Service zur Fehleranalyse genutzt werden.
Es folgt ein Programmierbeispiel, in dem ein Microcontroller (z.B. Atmel AVR) mit dem Nµ701.65 kommuniziert.
Ein weiteres Programmierbeispiel für ein National-Instruments DAQ-PAD (LabView) ist zu finden im AN701.65-117.

1.1 Vorbemerkungen

Während der Kommunikation agiert der Nµ701.65 als Slave, der Microcontroller agiert als Master und treibt das Clock-Signal SCLK.
Das SD-Signal ist bidirektional. Betreibt man die Schnittstelle über einen Levelshifter, muss dessen Richtung umschaltbar sein. Die Richtung des Levelshifters liefert im Beispiel der Pin SDIR.

2 Programmierbeispiel in C

2.1 Allgemeine Definitionen:

```
#include <avr/io.h>
#include <stdio.h>
#include <lcd-routines.h>
#include <avr/pgmspace.h>
#include <util/delay.h>
#include "_pins.h"
#include "uart.h"

#ifndef F_CPU
#define F_CPU 1000000
#endif

//SPI-Pins
#define SCLK PD2
#define SD          PD3
#define SDIR PD4
#define DDR_SPI     DDRD

typedef signed char int8_t;             //int8_t steht für einen 8-Bit Integer mit
einem Wertebereich -128 bis +127.
typedef unsigned char uint8_t;   //uint8_t steht für einen 8-Bit Integer ohne
Vorzeichen (unsigned int) mit einem Wertebereich von 0 bis 255

static int BITUS = 50;                  //Bit- zu Bit-Zeit in Mikrosekunden
```

2.2 Standalone-Betrieb aktivieren

```
void standalone()                       //schaltet SD / SCLK auf High
{
       PORTD |= (1<<SDIR);        //Level Shifter auf schreiben stellen
//erst DDR_SDIR 1, dann DDR_SD!
       DDR_SPI |= 0b00011100;
       _delay_us(BITUS);
       PORTD |= (1<<SD) | (1<<SCLK);    // SD und SCLK auf high: Standalone-Betrieb ein
//     PORTA = 2;
       lcd_setcursor(14,2);
       lcd_string("SA");

}
```

2.3 Schreiben vom Microcontroller auf den Nμ701.65

2.3.1 Schreiben eines Bits (0 / 1 / Startbit)

```
void send0()
{
       PORTD |= (1<<SD) | (1<<SCLK);           //SD und SCLK high
       _delay_us(BITUS);                       //warten
       PORTD &= ~(1<<SCLK);                    //SCLK low
       _delay_us(BITUS);                       //warten
       }

void send1()
{
       char POPO = 0x00;
       POPO = PORTD & ~(1<<SD);
       PORTD = POPO | (1<<SCLK); //SD low, SCLK high
       _delay_us(BITUS);
       PORTD &= ~(1<<SCLK);             //SCLK low
       _delay_us(BITUS);
}

void startbit()
{
       char POPO = 0x00;
```

```
	POPO = PORTD & ~(1<<SCLK);
	PORTD = POPO | (1<<SD);                 //SD high, SCLK low
	_delay_ms(5);
	PORTD &= ~((1<<SCLK) | (1<<SD)); //setzt Pin SCLK und SD low
	_delay_ms(5);                       //warten
	PORTD |= (1<<SD);                   // SD auf high setzen
	_delay_ms(5);
	PORTD &= ~((1<<SCLK) | (1<<SD)); //setzt Pin SCLK und SD low
	_delay_ms(5);
	PORTD |= (1<<SD);                   // SD auf high setzen
	_delay_us(BITUS);
}
```

2.3.2 Schreiben eines Adress- und Datenworts mit Start- RW- und Stopbit

```
void senddata(unsigned char Adress, unsigned char Data)
{
	PORTD |= (1<<SDIR);             //Level Shifter auf schreiben stellen
	DDR_SPI |= 0b00011100;

	startbit();

	//RW-Bit
	send0();

	//Beginn der Übertragung Adresse 6 bit
	int i=6;
	while (i>0)
	{	i--;
		if (Adress & (1<<i))
			send1();
		else
			send0();
	}


	//Beginn der Übertragung Daten 8 bit
	int j=8;
	while (j>0)
	{
		j--;
		if (Data & (1<<j))
			send1();
		else
			send0();
	}
	//Stopbit
	send0();

	PORTD &= ~(1<<SD);        //SD auf low schalten
	_delay_us(BITUS);
}
```

2.4 Lesen vom Nμ701.65 in den Microcontroller

2.4.1 Ein Bit einlesen

```
uint8_t ReadBit ()
{
	_delay_us(BITUS);
	PORTD |= (1<<SCLK); //SCLK auf high
	uint8_t k;
	k = (PIND & (1<<SD));     //SD-Pin einlesen
	_delay_us(BITUS);
	PORTD &= ~(1<<SCLK);      //SCLK auf low
	if (k)              //Ergebnis invertieren
	return(0);
	else return(1);
}
```

2.4.2 Eine Leseadresse senden und ein Datum einlesen

```
uint8_t readdata(unsigned char Adress)
{
	PORTD |= (1<<SDIR);                     //Dir auf high: Level Shifter auf
schreiben stellen
	PORTD &= ~((1<<SD) | (1<<SCLK));        //SD und SCLK auf low
	DDR_SPI |= 0b00011100;                  //Alle 3 Pins auf Ausgang
	_delay_ms (1);

	startbit();

	//RW-Bit
	send1();

	//Beginn der Übertragung Adresse 6 bit
	int i;
	for (i=5; i>=0; i--)
	if (Adress & (1<<i))
		send1();
	else
		send0();

	//Umschaltung von Schreiben zum Lesen
	PORTD |= (1<<SDIR) | (1<<SD) | (1<<SCLK); //Alle 3 auf high setzen

	_delay_us(BITUS*8);                         // Warten auf Umschaltung

    DDR_SPI &= ~(1<<SD);                    //SD auf Lesen geschaltet
	PORTD &= ~(1<<SDIR);                        //SDIR auf 0 (lesen) setzen
	PORTD &= ~(1<<SCLK);                        //SCLK auf low


	//Beginn der Übertragung Daten 8 bit
	uint8_t Daten = 0;
	int h;
	for (h=7; h>=0; h--)
	Daten |= (ReadBit()<<(h));

	return(Daten);
}
```