```c
/*----------------------------------------------------------------------------
 Copyright:      Radig Ulrich  mailto: mail@ulrichradig.de
 Author:         Radig Ulrich
 Remarks:        
 known Problems: none
 Version:        24.10.2007
 Description:    RS232 Routinen

 Dieses Programm ist freie Software. Sie können es unter den Bedingungen der 
 GNU General Public License, wie von der Free Software Foundation veröffentlicht, 
 weitergeben und/oder modifizieren, entweder gemäß Version 2 der Lizenz oder 
 (nach Ihrer Option) jeder späteren Version. 

 Die Veröffentlichung dieses Programms erfolgt in der Hoffnung, 
 daß es Ihnen von Nutzen sein wird, aber OHNE IRGENDEINE GARANTIE, 
 sogar ohne die implizite Garantie der MARKTREIFE oder der VERWENDBARKEIT 
 FÜR EINEN BESTIMMTEN ZWECK. Details finden Sie in der GNU General Public License. 

 Sie sollten eine Kopie der GNU General Public License zusammen mit diesem 
 Programm erhalten haben. 
 Falls nicht, schreiben Sie an die Free Software Foundation, 
 Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA. 
------------------------------------------------------------------------------*/
#include "config.h"
#include "usart.h"

#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <stdlib.h>
#include <stdarg.h>
#include <ctype.h>
#include <string.h>
#include <avr/io.h>

	
volatile unsigned int buffercounter = 0;

char usart_rx_buffer[BUFFER_SIZE];
char *rx_buffer_pointer_in	= &usart_rx_buffer[0];
char *rx_buffer_pointer_out	= &usart_rx_buffer[0];
	
//----------------------------------------------------------------------------
//Init serielle Schnittstelle
void usart_init(unsigned long baudrate) 
	{ 
	#if !USE_CAM
	//Serielle Schnittstelle 1
	//Enable TXEN im Register UCR TX-Data Enable
	UCR =(1 << TXEN | 1 << RXEN | 1<< RXCIE);
	// 0 = Parity Mode Disabled
	// 1 = Parity Mode Enabled, Even Parity
	// 2 = Parity Mode Enabled, Odd Parity
	//UCSRC = 0x06 + ((parity+1)<<4);
	//UCSRC |= (1<<USBS);
	//Teiler wird gesetzt 
	UBRR=(F_CPU / (baudrate * 16L) - 1);
	usart_status.usart_disable = 0;
	#endif //USE_CAM
	}

//----------------------------------------------------------------------------
//Routine für die Serielle Ausgabe eines Zeichens (Schnittstelle0)
void usart_write_char(char c)
	{
	#if CMD_TELNET
	if(usart_status.usart_disable)
		{
		if(rx_buffer_pointer_in == (rx_buffer_pointer_out - 1))
			{
			telnetd_send_data ();							//Datenverlust !
			while(telnetd_status.ack_wait)
```

```
72  ————————————{
73  ————————————eth_get_data();
74  ————————————}
75  ————————}
76  ————————*rx_buffer_pointer_in++ = c;
77  ————————if (rx_buffer_pointer_in == &usart_rx_buffer[BUFFER_SIZE-1])
78  ————————————{
79  ————————————rx_buffer_pointer_in = &usart_rx_buffer[0];
80  ————————————}
81  ————————}
82  ————————return;
83  ————#else
84  ————#if !USE_CAM
85  ————if(!usart_status.usart_disable)
86  ————————{
87  ————————//Warten solange bis Zeichen gesendet wurde
88  ————————while(!(USR & (1<<UDRE)));
89  ————————//Ausgabe des Zeichens
90  ————————UDR = c;
91  ————————}
92  ————return;
93  ————#endif //USE_CAM
94  ————#endif
95  ————}
96
97  //---------------------------------------------------------------------------
98  void usart_write_P (const char *Buffer,...)
99  ————{
100 ————va_list ap;
101 ————va_start (ap, Buffer);————
102 ————
103 ————int format_flag;
104 ————char str_buffer[10];
105 ————char str_null_buffer[10];
106 ————char move = 0;
107 ————char Base = 0;
108 ————int tmp = 0;
109 ————char by;
110 ————char *ptr;
111 ————————
112 ————//Ausgabe der Zeichen
113 ————for(;;)
114 ————————{
115 ————————by = pgm_read_byte(Buffer++);
116 ————————if(by == 0) break; ————————————————————// end of format string
117 ————————if (by == '%')
118 ————————————{
119 ————————————by = pgm_read_byte(Buffer++);
120 ————————————if (isdigit(by)>0)
121 ————————————————{
122 ————————————————str_null_buffer[0] = by;
123 ————————————————str_null_buffer[1] = '\0';
124 ————————————————move = atoi(str_null_buffer);
125 ————————————————by = pgm_read_byte(Buffer++);
126 ————————————————}
127 ————————————switch (by)
128 ————————————————{
129 ————————————————case 's':
130 ————————————————————ptr = va_arg(ap,char *);
131 ————————————————————while(*ptr)
132 ————————————————————————{
133 ————————————————————————usart_write_char(*ptr++);
134 ————————————————————————}
135 ————————————————————break;
136 ————————————————case 'b':
137 ————————————————————Base = 2;
138 ————————————————————goto ConversionLoop;
139 ————————————————case 'c':
140 ————————————————————format_flag = va_arg(ap,int);————————————//Int to char
141 ————————————————————usart_write_char (format_flag++);
142 ————————————————————break;
```

```
143     ————————————case 'i':
144     ————————————————Base = 10;
145     ————————————————goto ConversionLoop;
146     ————————————case 'o':
147     ————————————————Base = 8;
148     ————————————————goto ConversionLoop;
149     ————————————case 'x':
150     ————————————————Base = 16;
151     ————————————————//***************************
152     ————————————————ConversionLoop:
153     ————————————————//***************************
154     ————————————————itoa(va_arg(ap,int),str_buffer,Base);
155     ————————————————int b=0;
156     ————————————————while (str_buffer[b++] != 0)
157     ————————————————————{};
158     ————————————————b--;
159     ————————————————if (b<move)
160     ————————————————————{
161     ————————————————————move -=b;
162     ————————————————————for (tmp = 0;tmp<move;tmp++)
163     ————————————————————————{
164     ————————————————————————str_null_buffer[tmp] = '0';
165     ————————————————————————}
166     ————————————————————//tmp ++;
167     ————————————————————str_null_buffer[tmp] = '\0';
168     ————————————————————strcat(str_null_buffer,str_buffer);
169     ————————————————————strcpy(str_buffer,str_null_buffer);
170     ————————————————————}
171     ————————————————usart_write_str (str_buffer);
172     ————————————————move =0;
173     ————————————————break;
174     ————————————}
175     ————————}
176     ————————else
177     ————————————{
178     ————————————usart_write_char ( by );
179     ————————————}
180     ————————}
181     ————va_end(ap);
182     ————}
183
184     //---------------------------------------------------------------------------
185     //Ausgabe eines Strings
186     void usart_write_str(char *str)
187     ————{
188     ————while (*str)
189     ————————{
190     ————————usart_write_char(*str++);
191     ————————}
192     ————}
193
194     //---------------------------------------------------------------------------
195     //Empfang eines Zeichens
196     #if !USE_CAM
197     #if !USE_ARTNET
198     ISR
        (USART_RX)————————————————————————————————————————————————————————————————
        // A.Fischer => ISR = Interrupt Service Routine
199     ————{
200     ————if(!usart_status.usart_disable)
201     ————————{
202     ————————unsigned char receive_char;
203     ————————receive_char =
                (UDR);————————————————————————————————————————————————————————————//
                A.Fischer => Zeichen aus dem UART-Empfangsregister auslesen
204     //————————#if
        USART_ECHO————————————————————————————————————————————————————————————// A.
        Walder => in Usart.h
205     //————————usart_write_char(receive_char);
206     //————————#endif
207     ————————if
```

```
        (usart_status.usart_ready)                      // A.Fischer => Das Flag wird in der Hauptschleife gelöscht
            {
            usart_status.usart_rx_ovl = 1;              // A.Fischer => Kommando noch nicht bearbeitet (Overflow)
            return;                                     // A.Fischer => Ausgelesenes Zeichen verwerfen, ISR verlassen
            }
        if (receive_char == 0x08)                       // A.Fischer => Backspace empfangen?
            {
            if (buffercounter) buffercounter--;         // A.Fischer => Letztes Zeichen löschen
                return;                                 // A.Fischer => ISR  verlassen
                                                        // A.Fischer => Die 2. Hälfte der Bedingung (hinter &&) ist fragwürdig und falsch?!
                                                        // A.Fischer => Was für eine Sonderfunktion soll die Zeichenfolge "\<CR>" haben?
                                                        // A.Fischer => Was passiert, wenn buffercounter-1 negativ ist?
            }
        if (receive_char == '\r' && (!(usart_rx_buffer[buffercounter-1] == '\\')))  // A.Fischer => <CR> empfangen?
            {
            usart_rx_buffer[buffercounter] = 0;         // A.Fischer => Gesammelte Zeichen mit 0x00 terminieren
            buffercounter = 0;                          // A.Fischer => Einfügestelle für nächstes Zeichen zurücksetzen
                                                        // A.Fischer => einer Meinung nach keine gute Idee!!!!!
            usart_status.usart_ready = 1;               // A.Fischer => Hauptschleife informieren und Augen zu.
                                                        // A.Fischer => Keine weiteren Zeichen mehr sammeln, bis die Hauptschleife dran war!
            return;                                     // A.Fischer => ISR verlassen
            }
        if (buffercounter < BUFFER_SIZE - 1)
            {
            usart_rx_buffer[buffercounter++] = receive_char;    // A.Fischer =>     Zeichen sammeln
            }
        }
    else
        {                                               // A.Fischer => Was ist mit diesem Code-Teil?
                                                        // A.Fischer => Das soll sowas wie einen Ringpuffer implementieren.
                                                        // A.Fischer => Die folgende Erkennung eines vollen Puffers ist allerdings falsch!
                                                        // A.Fischer => Der Sonderfall, das rx_buffer_pointer_out auf usart_rx_buffer[0] zeigt und
                                                        // A.Fischer => rx_buffer_pointer_in auf usart_rx_buffer[BUFFER_SIZE-2] zeigt, wird nicht erkannt!!!
        if(rx_buffer_pointer_in == (rx_buffer_pointer_out - 1))
```

```
			{
			return;	//Datenverlust
			}	// A.Fischer => Die folgende Bedingung für das Rücksetzen des Zeigers an den Anfang
	// A.Fischer => lässt das letzte Byte im Puffer immer ungenutzt.
		*rx_buffer_pointer_in++ = UDR;	// A.Fischer => Zeichen im Puffer sammeln, Zeiger weiterrücken
		if (rx_buffer_pointer_in == &usart_rx_buffer[BUFFER_SIZE-1])	// A.Fischer => Ende des Puffers ?
			{
			rx_buffer_pointer_in = &usart_rx_buffer[0];	// A.Fischer => Zeiger zurÜck an den Anfang
			}
		}
	return;
	}
#endif //USE_ARTNET
#endif //USE_CAM
```