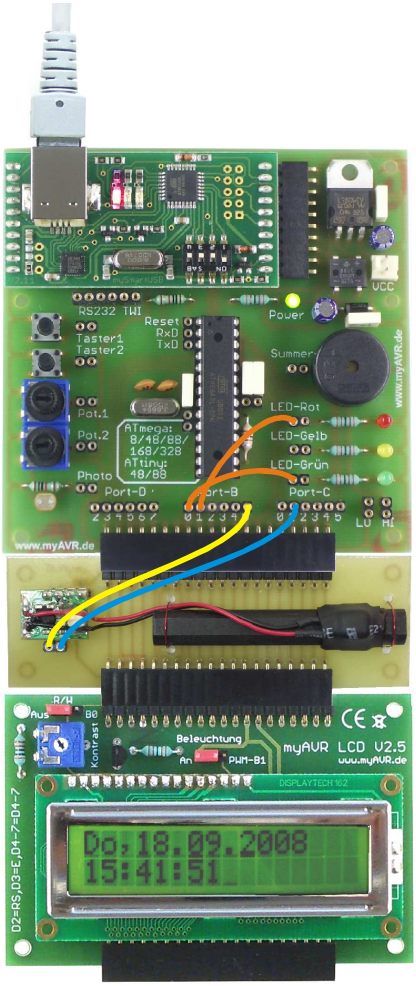


myFunkuhr

Inhalt

Promenadenring 8
02708 Löbau
Deutschland

www.myAVR.de
support@myavr.de

Tel: ++49 (0) 358 470 222
Fax: ++49 (0) 358 470 233

Einleitung

Eine Funkuhr ist eine Uhr, die von einem Langwellen-Zeitzeichensender per Funk ein ausgestrahltes Zeitsignal empfangen kann und dessen Uhrzeit selbständig übernimmt. Vorteile einer Funkuhr sind, dass die Uhrzeit dadurch immer sehr genau ausgegeben wird, es ist kein Nachstellen von Hand nötig und die Umstellung zwischen Sommer- und Winterzeit erfolgt automatisch.

Das DCF77-Signal ist die Definition der „richtigen Zeit Deutschlands“ und wird auf 77,5 kHz (Langwelle) in kodierter Form ausgestrahlt. Das D bedeutet das Ausstrahlungsland Deutschland, C kennzeichnet den Langwellensender, F ist die Nähe für Frankfurt und 77 ist die Sendefrequenz von 77,5 kHz. Das Signal ist auch außerhalb Deutschlands zu empfangen und wird dreimal stündlich als Morsezeichen gesendet. Weiterhin wird das Signal auch durch das Satellitensystem GPS und einen Telefondienst verbreitet.

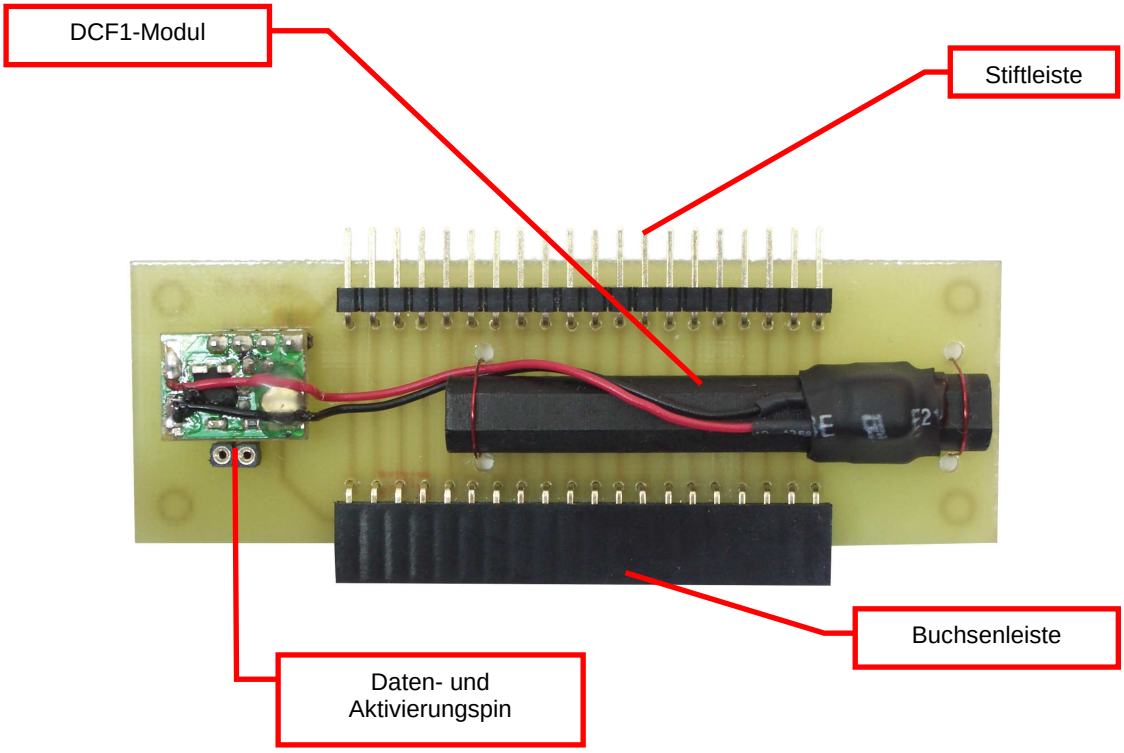
Neben Funkweckern und Funkuhren dient das Signal auch der Industrie und Wirtschaft als zentrale Zeitinformation, um Prozesse in unterschiedlichen Systemen zeitsynchron ablaufen zu lassen.

Eigenschaften Hardware

- myAVR Board MK2 USB, bestückt mit ATmega8 der Firma ATMEL, Taktfrequenz 3,6 MHz
- USB-Programmer mySmartUSB MK2 ist bereits auf dem Board integriert
- Spannungsversorgung 5 V
- DCF Funkmodul
- LCD mit Hintergrundbeleuchtung und 2x16 Zeichen
- einfache Kommunikation mit dem PC oder Notebook über die USB-Schnittstelle (virtueller COM Port)

Entwicklungsumgebung

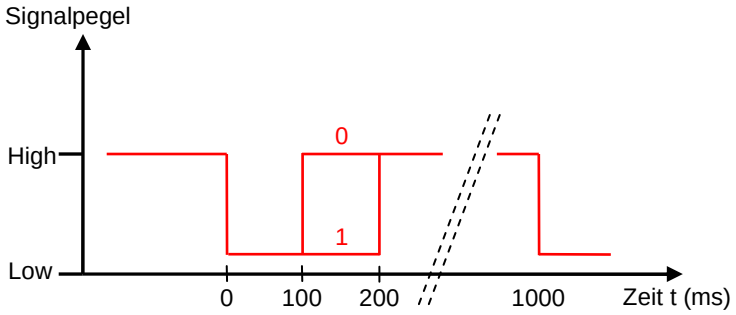
- Programmiersoftware: myAVR Workpad PLUS, SiSy AVR
- Programmertyp AVR910/AVR911, mySmartUSB MK2
- Anschluss: USB Port (virtueller COM Port)



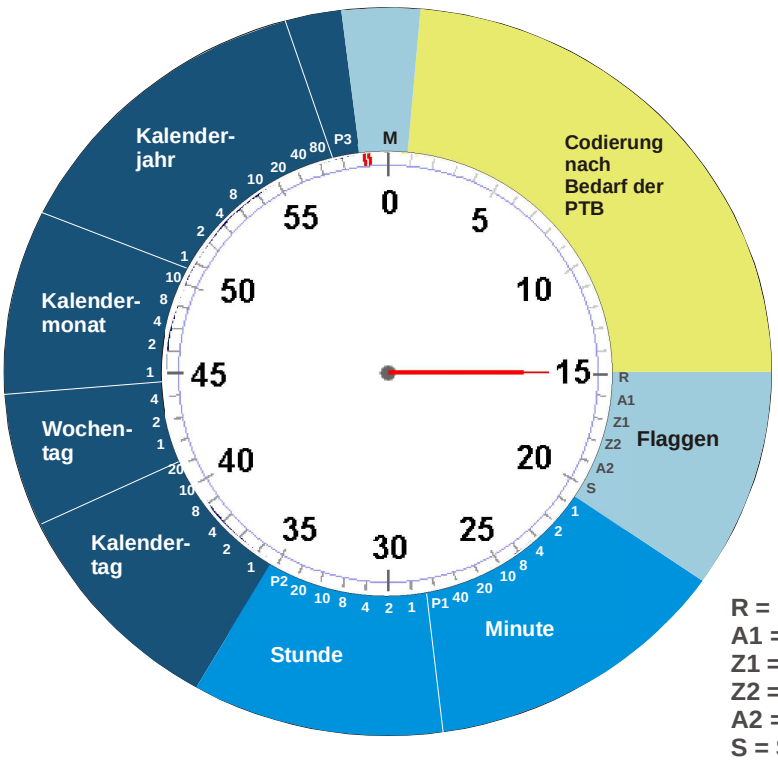
Grundlagen

Die Erzeugung des Signals erfolgt durch 3 voneinander unabhängige Atomuhren, die sich in der Sendeanlage in Mainflingen befinden. Diese unterliegen einer Überprüfung der drei Signale gegeneinander. Wird dabei eine Abweichung der Hauptsteuereinheit von einer der Reserveeinheiten festgestellt, so wird auf eine Reservesteuereinheit umgestellt. Wenn alle Signale voneinander abweichen, unterbricht die PTB die Ausstrahlung des Zeitsignals bis zur Wiederübereinstimmung der 3 Atomuhren.

Die bitweise Übertragung des DCF-Signals erfolgt mittels der amplitudenmodulierten Trägerfrequenz von 77,5 kHz. Die Bits werden dabei als Absenkung der Trägeramplitude auf 25 % realisiert. Das passiert 58 Sekunden lang. In der 59. Sekunde wird nicht abgesenkt, was die Minutenmarke kennzeichnet. Der Zustand des Bits wird dabei durch die unterschiedliche Länge der Absenkung übertragen. Eine binäre Null entspricht dabei einer Absenkung von 100 Millisekunden. Bleibt das Signal weitere 100 ms abgesenkt, so entspricht dies einer binären Eins.



Das Signal wird seit 1973 BCD-kodiert (auch 8-4-2-1-Kode) ausgestrahlt. Dabei handelt es sich um ein Kodierungsverfahren, das jede Ziffer einer Dezimalzahl einzeln, in einem Halbbyte, mit den Wertigkeiten 8-4-2-1 dualkodiert. Im folgenden DCF77-Zeittelegramm ist die Bedeutung der einzelnen Bits festgelegt und dargestellt.

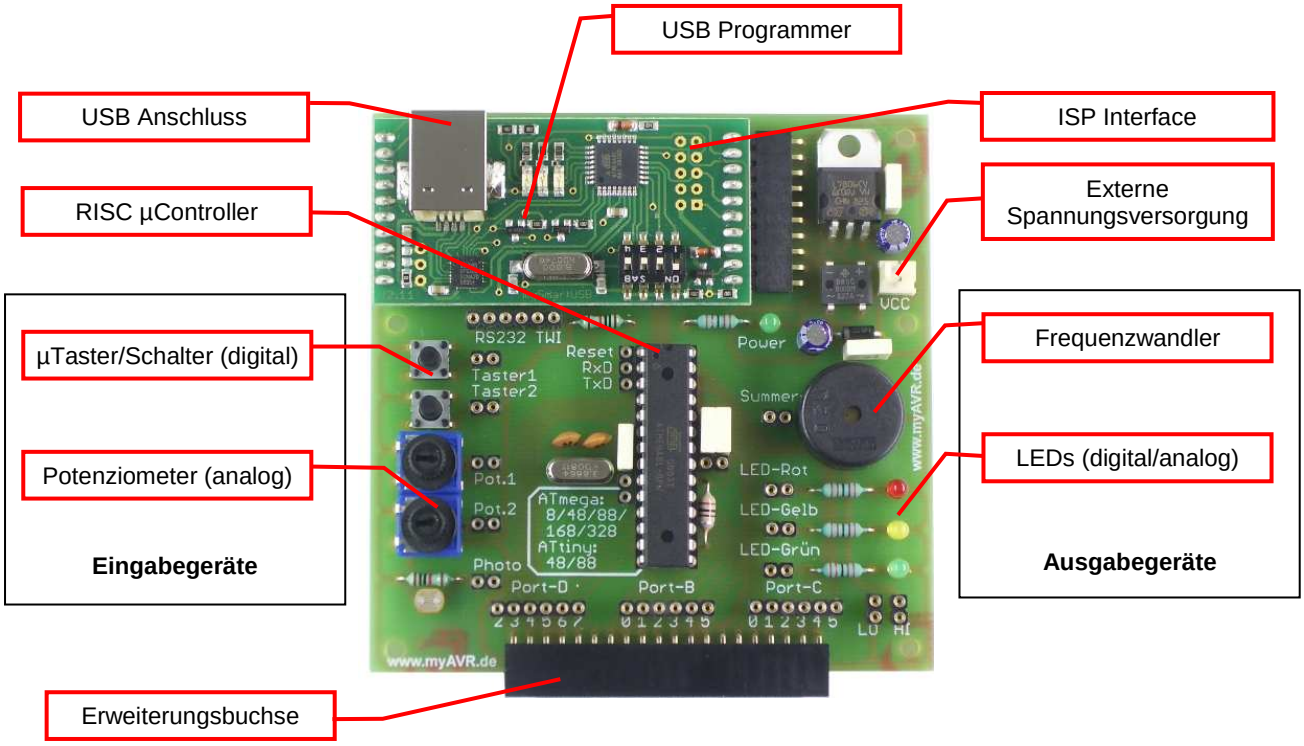


R = Reserveantenne
A1 = Schaltstunde
Z1 = Sonnenzeit, MEZ
Z2 = Winterzeit, MEZ
A2 = Schaltstunde
S = Standbit

Hardware myFunkuhr

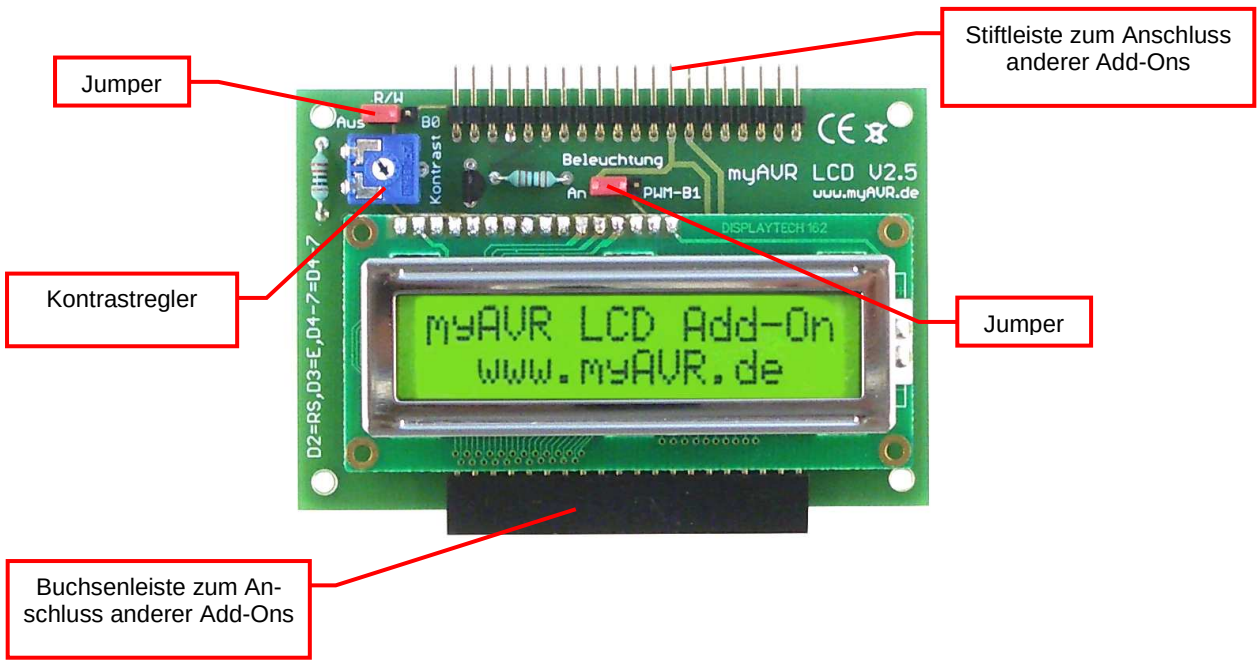
myAVR Board MK2 USB, bestückt

Das myAVR Board MK2 USB ist ein Lern- und Experimentierboard und besitzt folgende Struktur:
USB-Programmer, Peripherie zum Testen von Anwendungen, Erweiterungsbuchse.



myAVR LCD Add-On

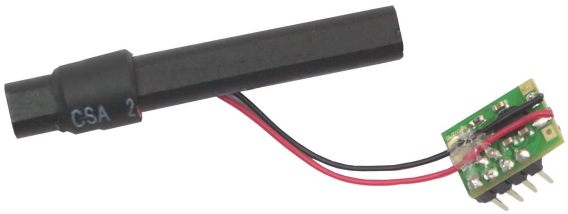
Das myAVR LCD Add-On ist ein anschlussfertiges LC-Modul für das myAVR Board. Es ist ausgestattet mit einem hochwertigen LC-Display (2 Zeilen, 16 Zeichen pro Zeile, Hintergrundbeleuchtung), Kontrastreglung, Anschlüsse für das myAVR Board sowie Jumper für die Einstellung des R/W Signals.



DCF Funkmodul

Die Spannungsversorgung von 5 V erfolgt direkt durch das myAVR Board. Die Signalpegel von 4 V (H-Pegel) und 1 V (L-Pegel) verarbeitet der verwendete Mikrocontroller ATmega8. Als Anschlüsse besitzt das Modul 4 Pins.

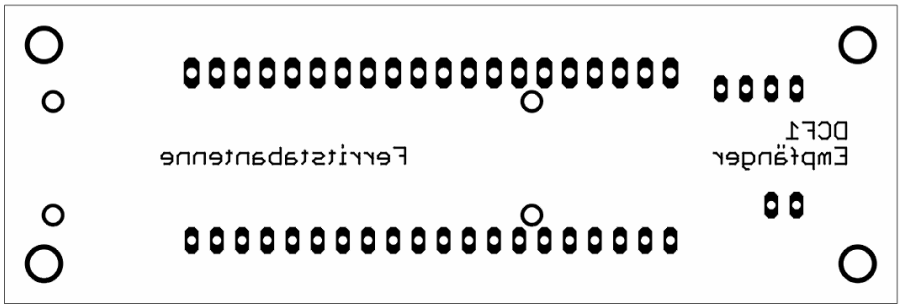
- Pin 1 mit der Bezeichnung VDD für die Spannungsversorgung (+5 V)
- Pin 2 (GND) für die Masse
- Pin 3 (DATA) als Signalausgangsleitung und
- Pin 4 (PON) als Aktivierungspin



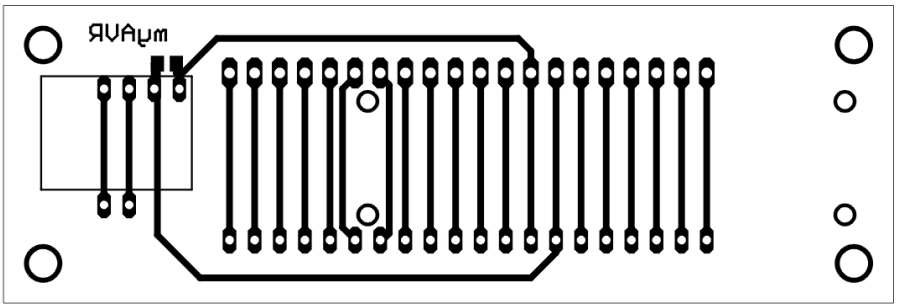
Lediglich die zwei Pins für die Spannungsversorgung wurden festgelegt. Dadurch können die Ponleitung und die Signalleitung frei verdrahtet werden. Ein Stabilisierungskondensator gleicht eventuelle Schwankungen des Moduls aus. Dafür wurde ein 100 nF Keramikkondensator (Kerko) verwendet, der zwischen VCC und GND geschalten wurde.

Layout

Platine Oberseite

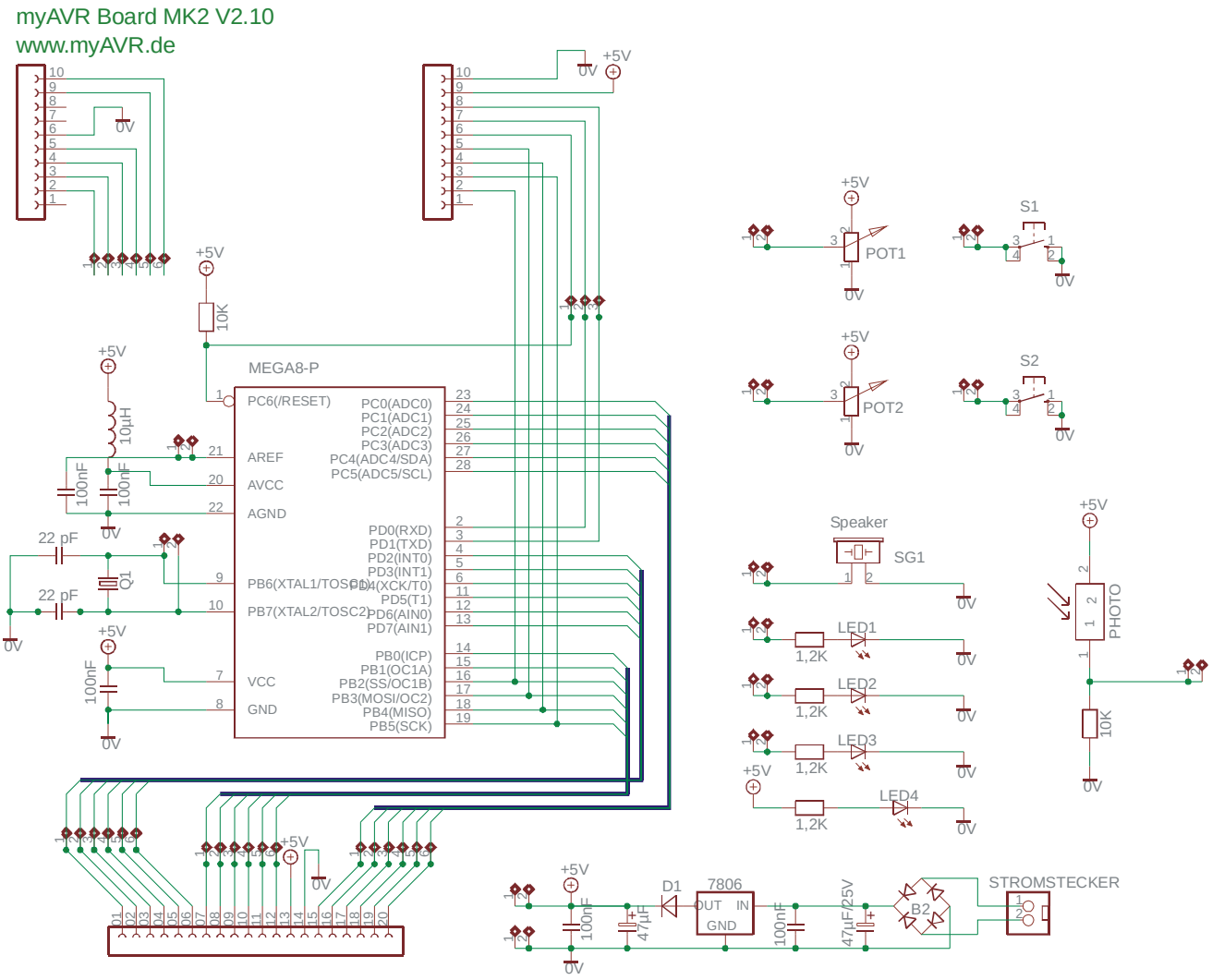


Platine Unterseite

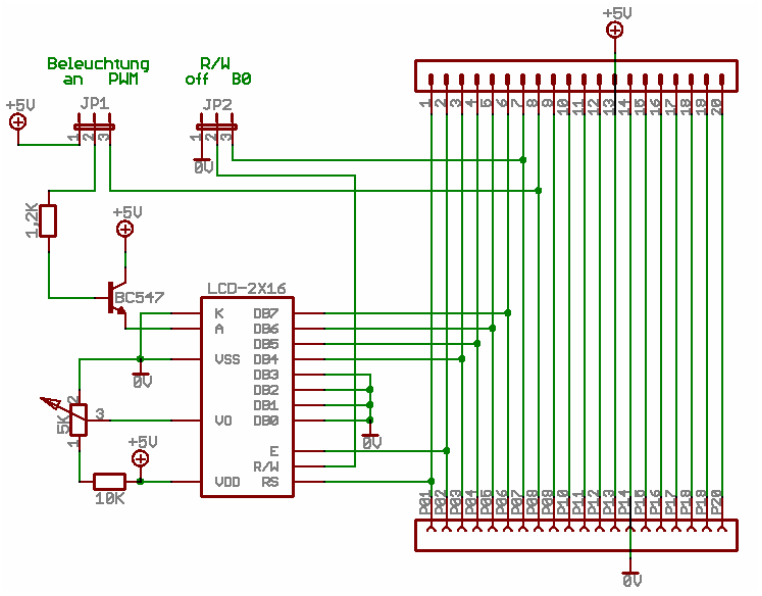


Schaltpläne

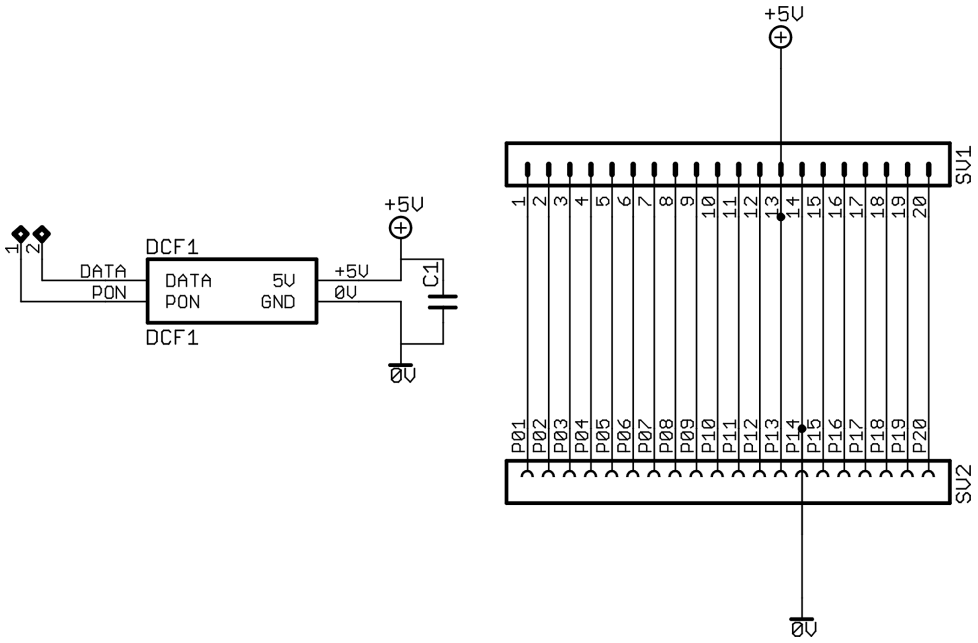
myAVR Board MK2 USB



myAVR LCD Add-On



Funkuhr Add-On



Stückliste

Bezeichnung im Schaltplan	Bauteil	Artikelnummer, Shop Suchbegriff	Anzahl
	myAVR Board MK2 USB	Board MK2 USB (http://shop.myavr.de)	1
	myAVR LCD Add-On	LCD Add-On http://shop.myavr.de	1
DCF1	DCF1-Modul	Fachhändler	1
	Schaltdraht ca. 20 Zentimeter (Farbsortiment)	Patchkabel (http://shop.myavr.de)	4
	USB-Kabel	USB-Kabel (http://shop.myavr.de)	1

Programmieren und Testen

Initialisierung

Als erstes müssen die Komponenten initialisiert werden. Da das Funkmodul zur Inbetriebnahme eine fallende Flanke von high nach low auf der Ponleitung benötigt, wird in der Initialisierungsfunktion init () der Port des Mikrocontrollers, an dem die Ponleitung des Funkmoduls angeschlossen ist (hier PORTB, 5), kurzzeitig auf high gesetzt. Erreicht wird dies mit der set Bit-Anweisung. sbi (PORTB, 5). Dieser High-Zustand wird mithilfe einer Zählschleife für 10.000 Takte gehalten, bevor die clear Bit-Anweisung cbi (PORTB,5) den Port wieder auf low setzt. Damit wird das DCF1-Modul aktiviert.

Die Initialisierungen und Grundfunktionen für die UART-Schnittstelle und das LCD Add-On, sowie benötigte Timer und Wartefunktionen werden durch das myAVR Workpad generiert.

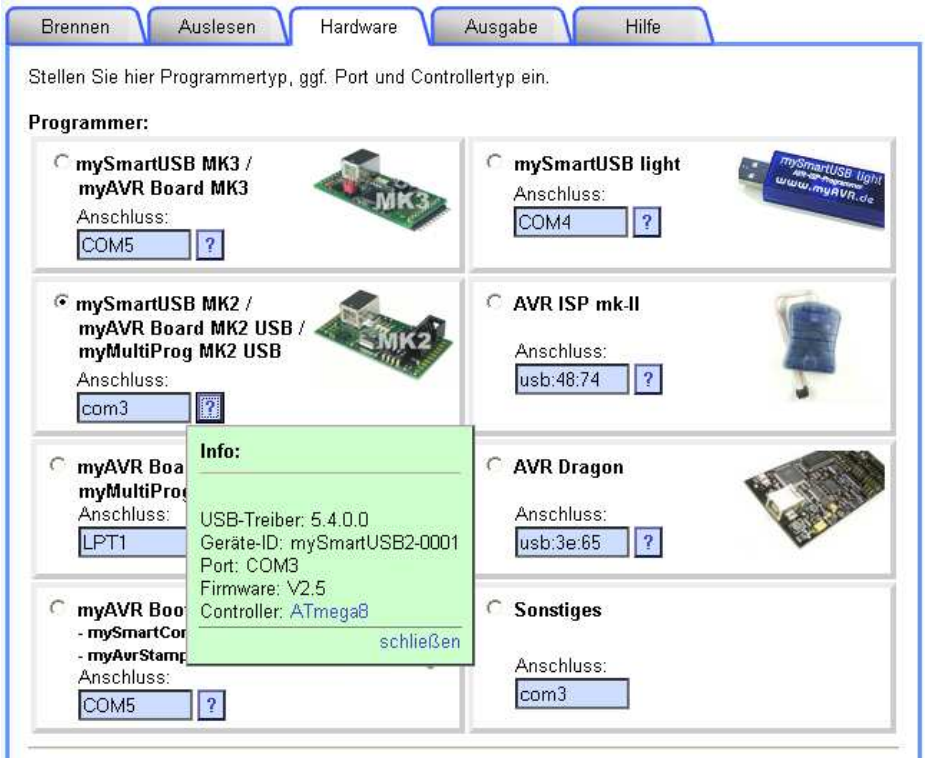
Synchronisation

Um die auszuwertenden Daten erfassen zu können, muss die Datenaufzeichnung mit der Empfangsfolge zeitlich aufeinander abgestimmt werden. Dies nennt man Synchronisation. Es ist ein software-seitiges Abwarten der Minutenmarke, das den Beginn einer neuen Minutenmarke kennzeichnet. Dies kann sehr gut identifiziert werden, da an dieser Stelle die Absenkung der Amplitude für eine volle Sekunde ausbleibt.
Mit einer if-Anweisung wird geprüft, ob nach einer Änderung von einem Low auf einen High-Pegel die Anzahl der durch den Mikrocontroller abgetasteten Low-Pegel größer als 40 war. Ist dies der Fall, war die Synchronisation erfolgreich und es beginnt die Datenaufzeichnung. In dem Acht Bit Integer `status` wird ein Synchronisationsbit `SYNC` durch eine bitweise ODER-Verknüpfung gesetzt. Das für die Sekundenzählung verantwortliche `offset` wird ebenso wie das `currentByte` Null gesetzt, um eine unbeabsichtigte Vorbelegung durch das Betriebssystem zu verhindern. Der Zählervariablen
`offsetBit` wird der Wert Eins zugewiesen.

Durch das Synchronisationsbit in der Status-Variable wird in der Mainloop durch die clear Bit-Anweisung cbi (PORTB, 1) die an diesem Port angeschlossene LED ausgeschalten. Dadurch wird die erfolgreiche Synchronisation angezeigt.

Ausgabe

Die Ausgabe der ermittelten Daten erfolgt mithilfe des myAVR LCD Add-Ons sowie dem myAVR Controlcenter und myAVR Workpad bzw. SiSy AVR. Sie übernehmen die Funktionen lcd_write und uartPutString. Diesen Funktionen werden mit ihrem Aufruf die Argumente `bufZ1` sowie `bufZ2` übergeben, um diese grafisch darzustellen.
Der Aufruf geschieht direkt in der Mainloop.



Quellcode für den Test auf Synchronisation

```
//------------------------------------------------------------------------
// Titel : Funkuhr V0.07
//------------------------------------------------------------------------
// Funktion : Augabe der aktuellen Zeit über UART und LCD
// Schaltung : myDCF77 V1.02
//------------------------------------------------------------------------
// Prozessor : ATmega8
// Takt : 3686400 Hz
// Sprache : C
// Datum : 11.9.2008
// Version : 0.07
// Programmer: mySmartUSB
//------------------------------------------------------------------------
// created by myAVR-CodeWizard
//------------------------------------------------------------------------

#define     F_CPU   3686400
#include <avr\io.h>
#include <avr\interrupt.h>
#include <util\delay.h>
#include <stdio.h>
#include <string.h>


void uartPutChar(char data);
void uartInit();

volatile uint8_t status = 0;
#define SYNC        0b10000000
#define TIMEREADY   0b01000000

int firstclear=0;

void wait_ms(int miliSec)
{
    _delay_loop_2( 1*(F_CPU/(1000/4)) * miliSec);    // 4 Zyklen warteschlei
fe
}
void wait_us(int mikroSec)
{
    _delay_loop_2( 1*(F_CPU/(1000000/4)) * mikroSec);    // 4 Zyklen wartesc
hleife
}
```

Quellcode von der Behandlung der Synchronisation in der Mainloop

```
//////////////////////////////////////////////////////
///
//  LCD-Funktionen für myAVR-Board + myAVR-LCD
//  4-BitModus an PortD Bit 4-7
//  PortD Bit 2 = RS, high=Daten, low=Kommando
//  PortD Bit 3 = E, high-Impuls für gültige Daten
//-----------------------------------------------------
--
//  Byte an LCD im 4-Bit-Modus senden
//  RS muss vorher richtig gesetzt sein
//  PE: data=zu sendendes Byte
void lcd_send(char data)
{
    // aktuelles RS ermitteln
    char rs=PORTD;
    rs&=4;
    // High-Teil senden
    char tmp=data;
    tmp&=0xf0;
    tmp|=rs;


    PORTD=tmp;
    // Schreibsignal
    sbi(PORTD,3);
    cbi(PORTD,3);
    // Low-Teil senden
    tmp=data;
    tmp&=0x0f;
    tmp*=16;
    tmp|=rs;
    PORTD=tmp;
    // Schreibsignal
    sbi(PORTD,3);
    cbi(PORTD,3);
    // verarbeiten lassen
    wait_ms(1);
}
//  Kommando an LCD senden
void lcd_cmd(char cmd)
{
    cbi(PORTD,2);       // RS löschen = Kommando
    lcd_send(cmd);      // senden
}
// Zeichen (Daten) an LCD senden
void lcd_write(char text)
{
    sbi(PORTD,2);       // RS setzen = Daten
    lcd_send(text);     // senden
}
//  Zeichenkette an LCD senden
void lcd_write(const char* pText)
{
    while(pText[0]!=0)
    {
        lcd_write(pText[0]);
        pText++;
    }
}
//  Zeichenkette an LCD senden
void lcd_write(const char* pText, int count)
```

```
{
    while(count!=0)
    {
        lcd_write(pText[0]);
        pText++;
        count--;
    }
}
//  Cursor auf Position 1,1
void lcd_home()
{
    lcd_cmd(0x02);
    wait_ms(2);         // warten
}
//  LCD-Anzeige löschen
void lcd_clear()
{
    lcd_cmd(0x01);
    wait_ms(2);         // warten
}

//  LCD anschalten
void lcd_on()
{

     lcd_cmd(0x0E);
 }

 //  LCD ausschalten
 void lcd_off()
 {
     lcd_cmd(0x08);
 }

 //  Cursorposition setzen
 void lcd_goto(int row, int col)
 {
     row--;              // Null-basierend
     row&=0x01;          // sicherheitshalber
     row*=0x40;          // Zeile nach Bit 6 bringen
     col--;              // Null-basierend
     col&=0x0f;          // sicherheitshalber
     char tmp=row|col;
     tmp|=0x80;          // Cursor setzen
     lcd_cmd(tmp);       // senden
 }

 //  lcd_init(..) - Schaltet die Ports und Initialisiert das LCD
 void lcd_init()
 {
     // Port D = Ausgang
     DDRD=0xff;
     PORTD=0;
     // warten bist LCD-Controller gebootet
     wait_ms(50);
     // SOFT-RESET
     PORTD = 0x30;    //0b00110000;
     sbi(PORTD,3);
     cbi(PORTD,3);
     wait_ms(5);
     PORTD = 0x30;    //0b00110000;
     sbi(PORTD,3);
     cbi(PORTD,3);
     wait_us(100);
     PORTD = 0x30;    //0b00110000;
     sbi(PORTD,3);
     cbi(PORTD,3);
     wait_ms(5);

     // 4-BitModus einschalten
     PORTD=0x20;
     // Schreibsignal
     sbi(PORTD,3);
     cbi(PORTD,3);
     wait_ms(5);

     // ab hier im 4-Bit-Modus
     lcd_cmd(0x28);       // Funktions-Set: 2 Zeilen, 5x7 Matrix, 4 Bit
     //lcd_off();
     lcd_cmd(0x06);       // Entry Mode
     lcd_on();
     lcd_clear();
 }
```

Quellcode für die Sekundenbestimmung

```
//------------------------------------------------------------
// TIMER1_COMPA_vect - Timer1 Interrupt bei Vergleichswert
// aktuelle Einstellung:  20 Hz  50 ms

//--------------------------------------------------------------------
volatile uint8_t anzLow=0;
volatile uint8_t anzHigh=0;
volatile uint8_t isLow=0;

struct DcfTime {
    uint8_t minute;
    uint8_t hour;
    uint8_t day;
    uint8_t weekday;
    uint8_t month;
    uint8_t year;
    };
DcfTime time,tempTime;
uint8_t z=0;
uint8_t second=0;
uint8_t second1=0;
uint8_t second2=0;
uint8_t offset=0;
uint8_t offsetBit;
uint8_t currentByte=0;
uint16_t i=0;

void debugPrintTime()
{
    uartPutChar('\n');
    uartPutChar(tempTime.minute);
    uartPutChar(tempTime.hour);
    uartPutChar(tempTime.day);
    uartPutChar(tempTime.weekday);
    uartPutChar(tempTime.month);
    uartPutChar(tempTime.year);
}
void clearLCD(char* bufZ1, char* bufZ2)
{
    if (firstclear)
        return;
    for(uint8_t i=0; i<20; ++i)
        bufZ1[i] = bufZ2[i] = ' ';
    firstclear=1;
}

char bufZ1[20] = {0};
char bufZ2[20] = {0};

char days[8][3] = {"So","Mo","Di","Mi","Do","Fr","Sa","So"};
void fillBufs(char* bufZ1, char* bufZ2, bool onlySeconds=false)
{
    // Buffer leeren = mit Leerzeichen ' ' füllen
    if(onlySeconds)
    {
        clearLCD(bufZ1,bufZ2);
        bufZ2[6] = '0' + ((offset/10));
        bufZ2[7] = '0' + (offset);
    }
    else
    {
        clearLCD(bufZ1,bufZ2);
        // Zeile 1
        sprintf( bufZ1, ",g,22f224.22ee7c.2000",days[time.weekday], time.da
y,time.month,time.year);
```

```
        // Zeile 2
        sprintf( bufZ2, "7c92003d:00:2289276",time.hour, time.minute, offse
t);
    }
}

ISR(TIMER1_COMPA_vect)
{
    cbi(PORTB,0);           // Traffic LED aus

//  uartPutChar((PINC & 0x02)*32);
    // high und war high
    if(PINC & 0x02 && !isLow)
        anzHigh++;
    // low und war low
    else if(!(PINC & 0x02) && isLow)
        anzLow++;
    // Änderung nach high -> test auf SYNC
    else if(isLow)
    {
        if(anzLow>40)
        {
            status|=SYNC;
            offset=0;
            offsetBit=1;
            currentByte=0;
        }
        isLow=false;
        anzLow=0;
    }
    // Änderung nach low -> Bitwert ermitteln
    else
    {
        sbi(PORTB,0);   // Traffic LED an
        uint8_t bit=0;
//uartPutChar(anzHigh);
        if(status & SYNC)
        {
            if(anzHigh>5)
            {
                bit=offsetBit;
                z++;            //1-en zählen
            }
        }
        offsetBit*=2;
        bool ok=true;

        ///////// Bitwert speichern ////////////////////
        if(offset<21)
        {
            if(offset==20)
            {
                currentByte=0;
                offsetBit=1;
            }
        }
        else if(offset<29)  // Minute
        {
            currentByte|=bit;
            if(offset==27)
            {
                tempTime.minute=currentByte & 0x7F;
```

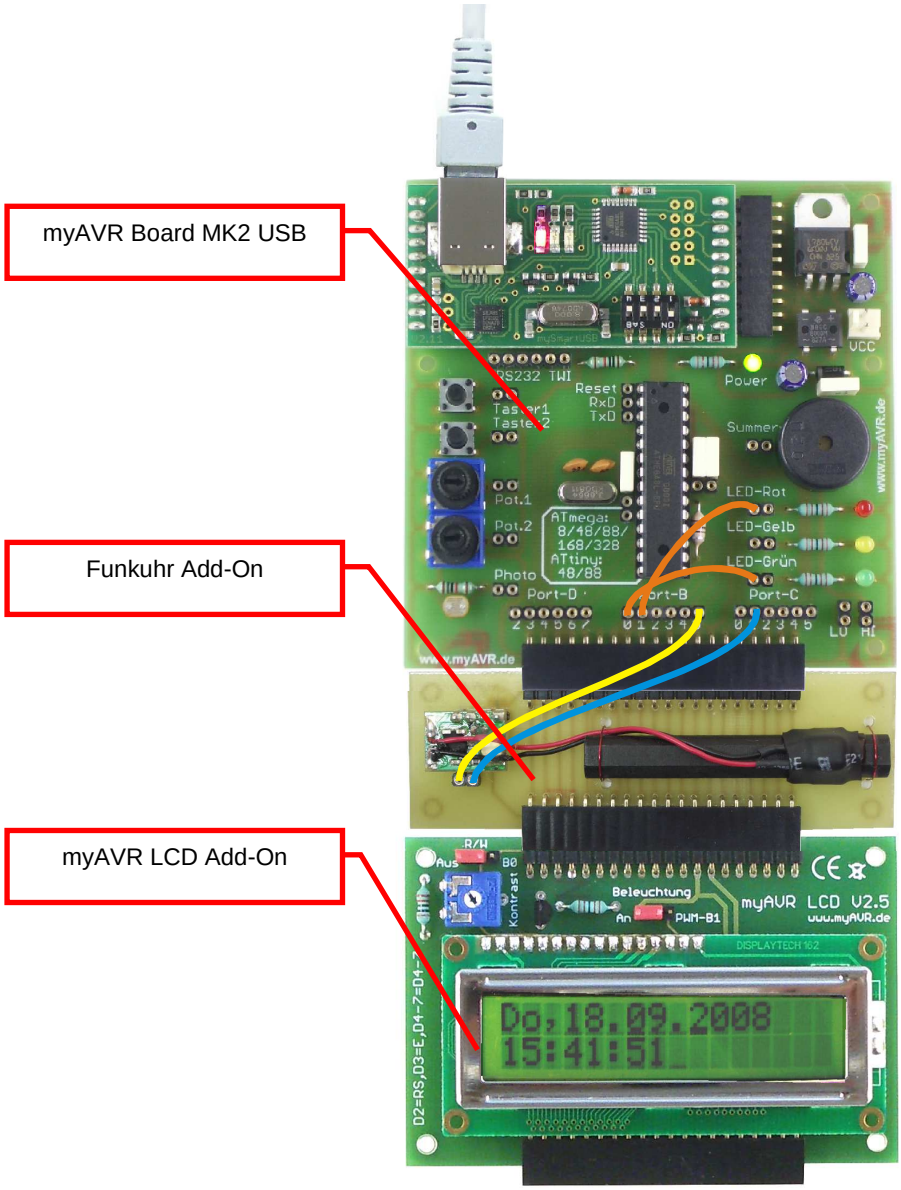
```
        }
        else if(offset==28) // Prüfbit
        {
            currentByte=0;
            offsetBit=1;
        }
    }
    else if(offset<36)  // Stunde
    {
        currentByte|=bit;
        if(offset==34)
        {
            tempTime.hour=currentByte & 0x3F;
        }
        else if(offset==35) // Prüfbit
        {
            currentByte=0;
            offsetBit=1;

        }
    }
    else if(offset<42)  // Tag
    {
        currentByte|=bit;
        if(offset==41)
        {
            tempTime.day=currentByte &0x3F;
            currentByte=0;
            offsetBit=1;
        }
    }
    else if(offset<45)  // Wochentag
    {
        currentByte|=bit;
        if(offset==44)
        {
            tempTime.weekday=currentByte & 0x07;
            currentByte=0;
            offsetBit=1;
        }
    }
    else if(offset<50)  // Monat
    {
        currentByte|=bit;
        if(offset==49)
        {
            tempTime.month=currentByte & 0x1F;
            currentByte=0;
            offsetBit=1;
        }
    }
    else if(offset<58)  // Year
    {
        currentByte|=bit;
        if(offset==57)
        {
            tempTime.year=currentByte;
            currentByte=0;
            offsetBit=1;
        }
    }
    else if(offset==58) // Prüfbit für Day...Year
    {
```

```
            /*TODO -> onError DeSync*/
            // wenn ok
            time=tempTime;
            status|=TIMEREADY;
        }

        offset++;
        // next
        isLow=true;
        anzHigh=0;
//debugPrintTime();
    }
}


void init()
{

    time.year=0;
    // Ports initialisieren
    cbi(DDRC,1);
    DDRB=0xFF;      // PORTB auf Ausgang


    //--- Timer 1 initialisieren ---
    TCCR1B=0x02;    // Teiler 1/8
    TCCR1B|=0x08;   // Modus: Zählen bis Vergleichswert
    OCR1A=23040/2;  // Vergleichswert speichern
    TIMSK|=0x10;    // Interrupt bei Vergleichswert

    uartInit();

    // DCF-Init
    sbi(PORTB,5);
    for(uint16_t i=0; i<10000; i++)
    {}
    cbi(PORTB,5);

    //--- Interrupts erlauben ---
    sei();
}
void uartPutString(const char* data, int size=-1)
{
    if(size == -1)
        size = strlen(data);
    for(int i=0; i<size; ++i)
        uartPutChar(data[i]);
}
```

Quellcode Ausgabefunktionen für LCD und UART

```
//////////////////////////////////////////////////////////////////////////
//
// Main-Funktion
//////////////////////////////////////////////////////////////////////////
//
main()
{
    init();     // Initialisierungen
    lcd_init();

    while (true)    // Mainloop-Begin
    {

        if(status & SYNC)

    {
        cbi(PORTB,1);
        fillBufs(bufZ1,bufZ2,true);
    }
    else
    {
        sbi(PORTB,1);
        sprintf(bufZ1,"Synchronisation     ");
        sprintf(bufZ2,"Bitte warten        ");
    }

    if(status & TIMEREADY)
    {
        status &= ~TIMEREADY;
        fillBufs(bufZ1,bufZ2);
        offset=0;
    }

    lcd_goto(1,1);
    lcd_write(bufZ1);
    lcd_goto(2,1);
    lcd_write(bufZ2);

    uartPutString(bufZ1);
    uartPutString(bufZ2);

}
// Mainloop-Ende
}


void uartInit()
{
    UBRRL  = 23;            //9600Baud siehe Baudratentabelle
    UCSRB = 0x08;           //Sender enable UCSRB / UCR bei z.B.: 2313
}

{
    //warte bis UDR leer ist UCSRA / USR bei z.B.: 2313
    while (!(UCSRA&32));
    //sende
    UDR=data;
}
```

Anwendungsbeispiel



Viel Erfolg und Spaß beim Programmieren und Testen!