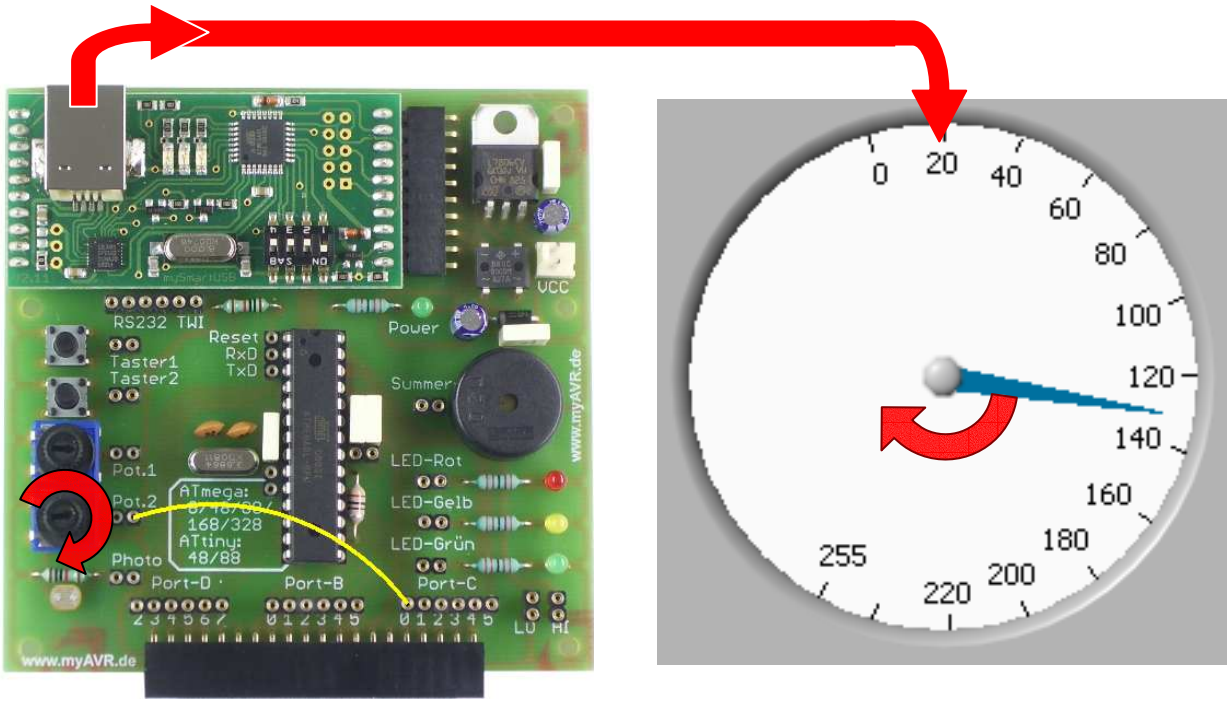


LabView mit der myAVR Produktfamilie

Erstellt von: Andreas Düring

Inhalt

Promenadenring 8
02708 Löbau
Deutschland

www.myAVR.de
support@myavr.de

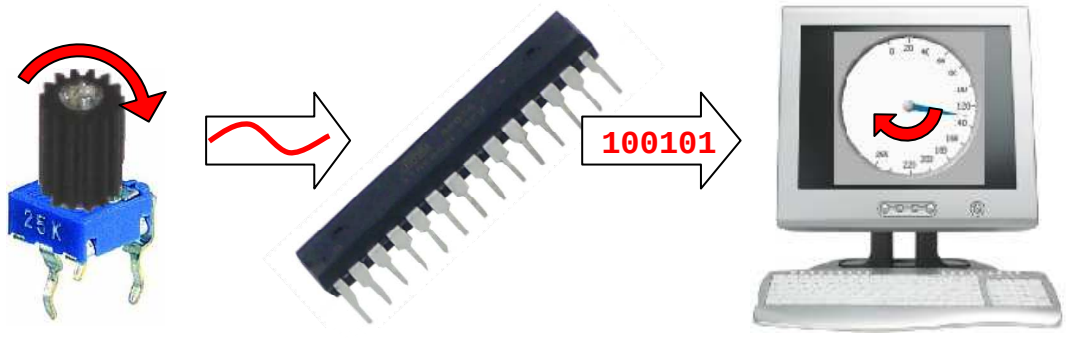
Tel: ++49 (0) 358 470 222
Fax: ++49 (0) 358 470 233

Einleitung

Das vorliegende Anwendungsbeispiel „*LabView* mit der myAVR Produktfamilie“ beschreibt eine Möglichkeit, die Lern- und Experimentiersysteme von myAVR zusammen mit dem Programm *LabView*[1] von National Instruments[2] zu nutzen.
LabView ist eine grafische Programmierumgebung zur Erstellung von Mess-, Prüf-, Steuer- und Regelsystemen. Die primären Funktionen sind Aufzeichnung, Darstellung, Analyse und Auswertung von Daten.
Ein in *LabView* erstelltes Programm wird als „Virtuelles Instrument“ bezeichnet und besteht aus 2 Hauptkomponenten, dem Frontpanel und dem Blockdiagramm.
Das Frontpanel enthält die Bedien- und Anzeigeelemente. Bei diesem Projekt sind dies der Signalgraph, das Rundinstrument, die Stop-Schaltfläche und das Dropdown-Menü zur Auswahl des richtigen COM-Portes.
Das Blockdiagramm enthält den grafischen Programmcode, auch G-Code genannt. Eine Erläuterung des G-Codes für dieses Projekt finden Sie am Ende des Dokumentes.

Zielstellung

Ziel dieses Projektes soll es sein, das Drehen am Potentiometer des myAVR Boards in dem Programm *LabView* grafisch darzustellen. Dies geschieht auf einem Rundinstrument und einem Signalverlaufsgraphen.

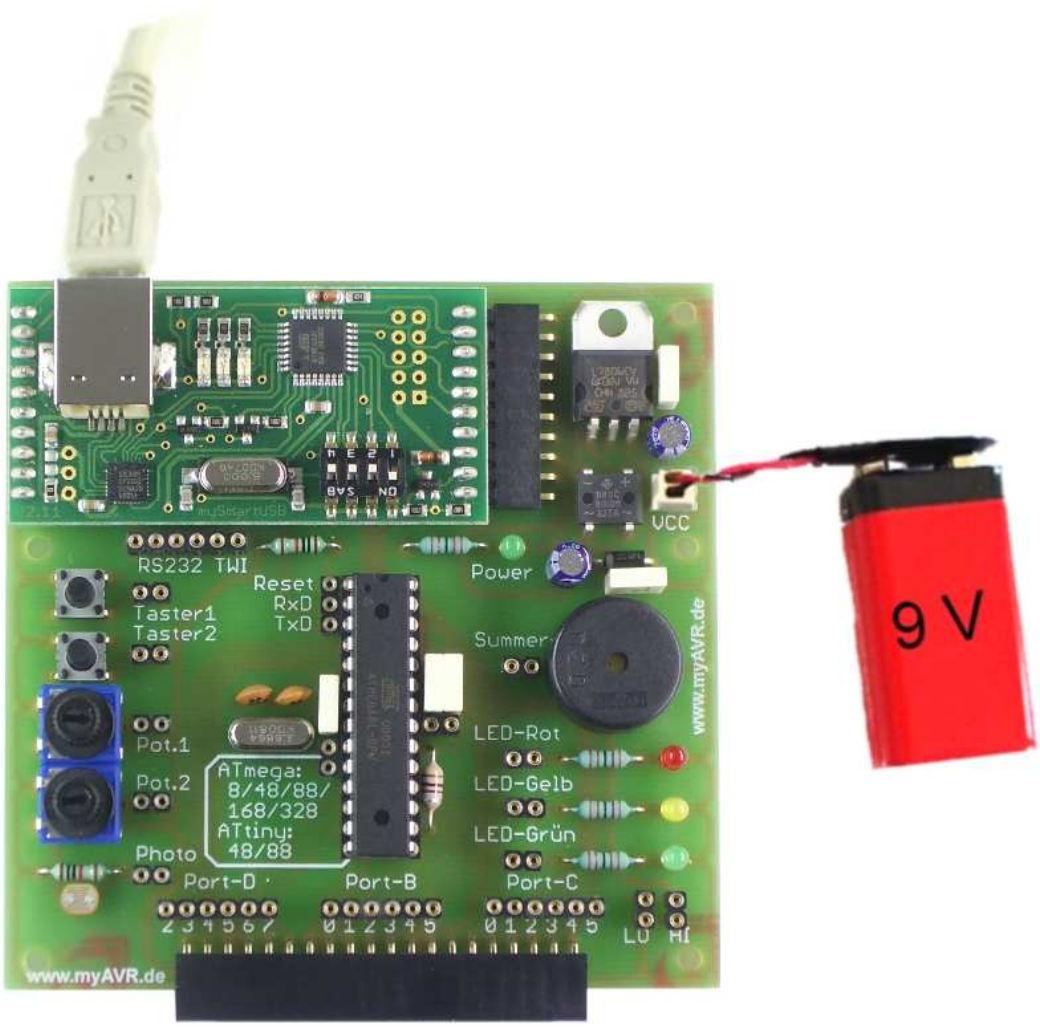


Das Drehen am Potentiometer erzeugt analoge Signale. Es können jedoch nur digitale Signale an den PC gesendet werden. Daher müssen die analogen Daten in digitale umgewandelt werden. Dies macht der A/D-Wandler des Mikrocontrollers. Er wandelt die Signale in den so genannten ADC-Code um.
Die Daten werden dann über die UART-Schnittstelle des Mikrocontrollers an den PC gesendet. Dazu wird in unserem Beispiel der mySmartUSB MK2 verwendet. Der mySmartUSB stellt einen virtuellen COM-Port im Computer zur Verfügung. Von diesem virtuellen COM-Port liest *LabView* die Daten ein, die das Mikrocontroller-Board sendet. Nähere Informationen zu den Themen Analog-Digital-Wandler und UART finden Sie im „myAVR Lehrbuch Mikrocontrollerprogrammierung“.

[1] *LabVIEW* ist eine Marke der National Instruments Corporation
[2] National Instruments, NI, und ni.com sind Marken der National Instruments Corporation

Voraussetzungen

- Anschluss: USB 2 Port am PC
- Programmertyp: mySmartUSB MK2 (virtueller COM Port)
- Board: myAVR Board MK2 USB mit ATmega8 und 3,6864 MHz
- Programmiersoftware: SiSy AVR oder myAVR Workpad oder myAVR ProgTool
- Weitere Software:
  - o *LabView* von National Instruments
  - o VISA-Treiber von National Instruments (http://joule.ni.com/nidu/cds/view/p/id/1370/lang/en)
  - o Die Datei *OsziPoti.vi*[3] zum Auswerten und Darstellen der Daten
  - o Hex-Datei *potioszi.hex*[4] zum Brennen auf Controller oder Assemblerdatei *potioszi.s*[5] zum selber übersetzen und brennen oder die C-Datei *potioszi.cc*[6] zum selber übersetzen und brennen



[3] ist im Projekt enthalten
[4] ist im Projekt enthalten
[5] ist im Projekt enthalten
[6] ist im Projekt enthalten

Durchführung

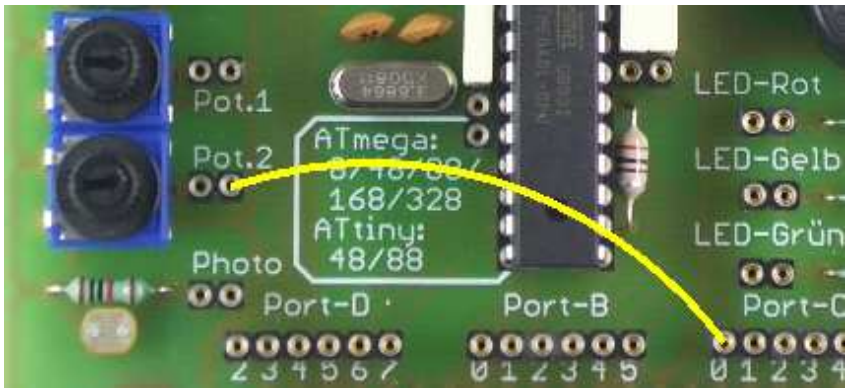
Mit dem myAVR Board MK2 USB

1. Schließen Sie das myAVR Board MK2 an den Computer an.
2. Öffnen Sie die Assemblerdatei *potioszi.s*[7] in myAVR Workpad oder SiSy AVR und kompilieren und brennen Sie diese
   oder
   Öffnen Sie die C-Datei *potioszi.cc*[8] in myAVR Workpad oder SiSy AVR und kompilieren und brennen Sie diese
   oder
   Brennen Sie die HEX Datei *potioszi.hex* auf den Controller

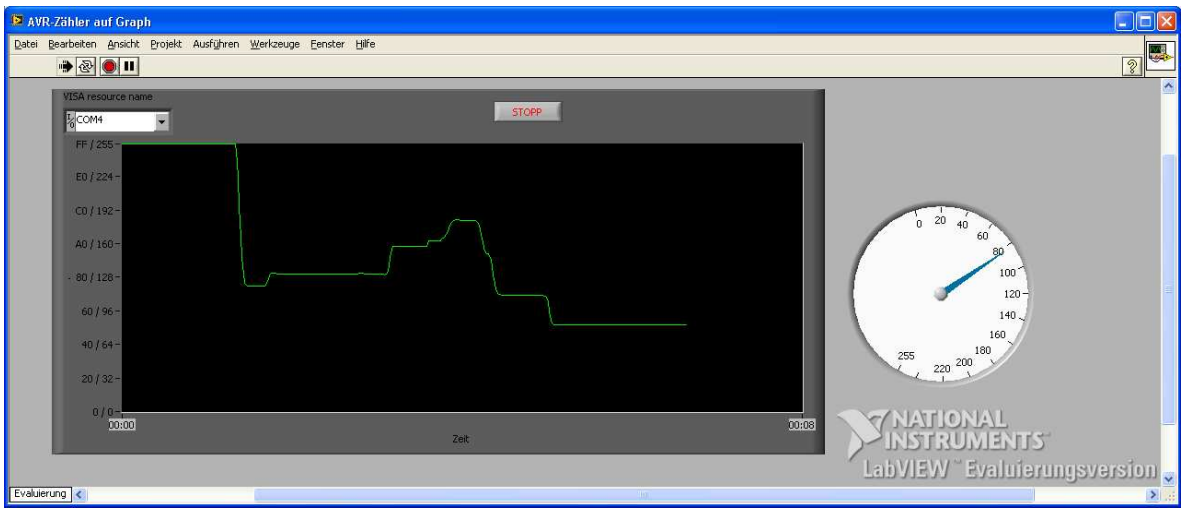
Hauptschleife in Assembler	Die Hauptschleife in C

`mainloop:`
`  ; Analog-Digital-Wandeln`
`  rcall  getADC`
`  ; Senden an LabView`
`  rcall  senden`
`  ; warten`
`  rcall  waitMs`
`  ; hauptschleife`
`  rjmp  mainloop`
	`while (true)`
	`{`
	`  // A/D Wert in Variable`
	`  wert = getValueADCandWait();`
	`  // Senden an LabView`
	`  uartPutChar(wert);`
	`  // Warten`
	`  myWait_10ms();`
	`// Mainloop-Ende`
	`}`

3. Verbinden Sie ein Potentiometer des Boards mit dem Port C.0

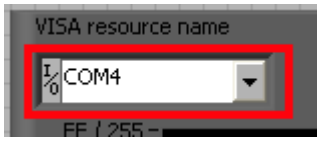


4. Starten Sie *LabView*
5. Öffnen Sie die Datei *OsziPoti.vi* in *LabView*.



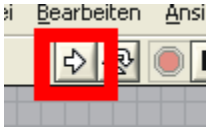
[7] siehe auch Anhang
[8] siehe auch Anhang

6. Stellen Sie in *LabView* den richtigen COM-Port ein. Den richtigen Port finden Sie folgendermaßen heraus:



   - im myAVR Workpad PLUS oder myAVR Workpad SE:
     Extras → Einstellungen → Hardware (bei mySmartUSB MK2 auf „Test“ klicken“)
   - in SiSy AVR:
     unter Projekt → Definieren → Extras (AVR) → Hardware einstellen (bei mySmartUSB MK2 auf „Test“ klicken“)

7. Schalten Sie das mySmartUSB MK2 in den Datenmodus, dazu betätigen Sie beim mySmartUSB MK2 den DIP-Schalter 2



8. Starten Sie das VI. Dazu klicken Sie auf die Schaltfläche mit dem Pfeil.
9. Drehen Sie am Potentiometer auf dem myAVR Board MK2 USB.
10. Sie können nun sehen, wie sich der Zeiger des Rundinstrumentes bewegt.

Programme

Quelltext für das Analog-Digital-Wandeln

Dies ist der Assembler-Quelltext für das Programm, das die Stellung des Potentiometers in einen ADC-Code wandelt und diesen per UART sendet.

Datei potioszi.s:

```
;+---------------------------------------------------------------------
;| Title            : Digitales Oszi
;+---------------------------------------------------------------------
;| Funktion         : Wandelt Analogsignal per A/D-Wandler um und
;|                  : sendet Daten per UART an den PC
;| Schaltung        : PORTC.0=Pot1
;+---------------------------------------------------------------------
;| Prozessor        : ATmega8
;| Takt             : 3,6864 MHz
;| Sprache          : Assembler
;| Datum            : 26.10.2009
;| Version          : 1.0
;| Autor            : Andreas Duering
;+---------------------------------------------------------------------
.include    "AVR.H"
;---------------------------------------------------------------------
;Reset and Interrupt vector              ;VNr.  Beschreibung
begin:      rjmp    main                ;1   POWER ON RESET
            reti                        ;2   Int0-Interrupt
            reti                        ;3   Int1-Interrupt
            reti                        ;4   TC2 Compare Match
            reti                        ;5   TC2 Overflow
            reti                        ;6   TC1 Capture
            reti                        ;7   TC1 Compare Match A
            reti                        ;8   TC1 Compare Match B
            reti                        ;9   TC1 Overflow
            reti                        ;10  TC0 Overflow
            reti                        ;11  SPI, STC Serial Transfer Complete
            reti                        ;12  UART Rx Complete
            reti                        ;13  UART Data Register Empty
            reti                        ;14  UART Tx Complete
            reti                        ;15  ADC Conversion Complete
            reti                        ;16  EEPROM Ready
            reti                        ;17  Analog Comparator
            reti                        ;18  TWI (IC) Serial Interface
            reti                        ;19  Store Program Memory Ready
;---------------------------------------------------------------------
;Start, Power ON, Reset
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
;-------------Einstellungen fuer den A/D-Wandler-------------------------
            ldi     r16,0
            out     ADMUX,r16           ; Kanal 0 waehlen
            ldi     r25,0b11000101      ; ADC - Einstellungen: VT 32
            out     ADCSRA,r25          ; ADC enable, Starte Umwandlung
;-------------Einstellung für UART - Uebertragung-----------------------
            sbi     UCSRB,3             ; Senden einschalten
            ldi     r16,23              ; Baudrate
            out     UBRRL,r16
```

```
;-----------------------------------------------------------------------
mainloop:   rcall   getADC              ; Analog-Digital-Wandeln
            rcall   senden              ; Senden an LabView
            rcall   waitMs              ; warten
            rjmp    mainloop            ; zurück zum Anfang
;-----------------------------------------------------------------------
getADC:     in      r26,ADCL            ; Low-Bit
            in      r27,ADCH            ; High-Bit
            asr     r27                 ; rotate right des high-Anteiles
                                        ; Überlauf in C-Flag
            ror     r26                 ; rotate right low (bit kommt aus C)
            asr     r27                 ; s.o
            ror     r26                 ; s.o
            mov     r25,r26             ; Verschieben
            sbi     ADCSRA,6            ; nächste Umwandlung
            ret
;-----------------------------------------------------------------------
senden:     rcall   waitReady           ; Warte bis bereit
            out     UDR,r25             ; senden
            ret
;-----------------------------------------------------------------------
waitReady:  sbis    UCSRA,5             ; Warte bis sendebereit
            rjmp    waitReady
            ret

;-----------------------------------------------------------------------
; UP WaitMs, Warteroutine im Millisekundenbereich
waitMs:     push    r16                 ;r16 sichern
            push    r17                 ;r17 sichern
            push    r18                 ;r18 sichern
            ldi     r16,10              ;Laufvariable ca 10 ms bei 3,6MHz
loop1:      ldi     r17,0xFF            ;Laufvariable loop1
loop2:      ldi     r18,13              ;Laufvariable loop3
loop3:      dec     r18                 ;Z3 -1 hier Kalibrierung auf MCU Takt
            brne    loop3               ;Solange nicht NULL
            dec     r17                 ;Zähler 2 -1
            brne    loop2               ;Solange nicht NULL
            dec     r16                 ;Zähler1 -1
            brne    loop1               ;Solange nicht NULL
            pop     r18                 ;r18 wiederherstellen
            pop     r17                 ;r17 wiederherstellen
            pop     r16                 ;r16 wiederherstellen
            ret                         ;Rücksprung
;-----------------------------------------------------------------------
```

Dies ist der C-Quelltext für das Programm, das die Stellung des Potentiometers in einen ADC-Code wandelt und diesen per UART sendet.

Datei potioszi.cc:

```
//-------------------------------------------------------------------------
// Titel : Digitales Oszi
//-------------------------------------------------------------------------
// Funktion : Wandelt Analogsignal per A/D-Wandler um und sendet Daten per
//              UART
// Schaltung : PORTC.0=Pot1
//-------------------------------------------------------------------------
// Prozessor : ATmega8
// Takt : 3686400 Hz
// Sprache : C
// Datum : 3.11.2009
// Version : 1.0
// Autor : Andreas Duering
//-------------------------------------------------------------------------
//
#define     F_CPU   3686400
#include <avr\io.h>
#define     BAUD    9600

//-------------------------------------------------------------------------
// Titel    : C-Funktion Zeichen zu UART senden.
//-------------------------------------------------------------------------
// Funktion     : Sendet Daten per UART
// IN       : char data
// OUT      : -
//-------------------------------------------------------------------------
void uartPutChar(char data)
{
    //warte bis UDR leer ist UCSRA / USR bei z.B.: 2313
    while (!(UCSRA&32));
    //sende
    UDR=data;
}

//-------------------------------------------------------------------------
// getValueADCandWait -
 Digitalisiert einen Analogwert und wartet auf das Ergebnis
//-------------------------------------------------------------------------
unsigned char getValueADCandWait()
{
    sbi(ADCSRA,6);  // Start ADC
    while(ADCSRA & 0x40)    // Warten bis fertig
    {}
    unsigned char wert=ADCH;    // Einlesen des Analogwertes
    return wert;
}
```

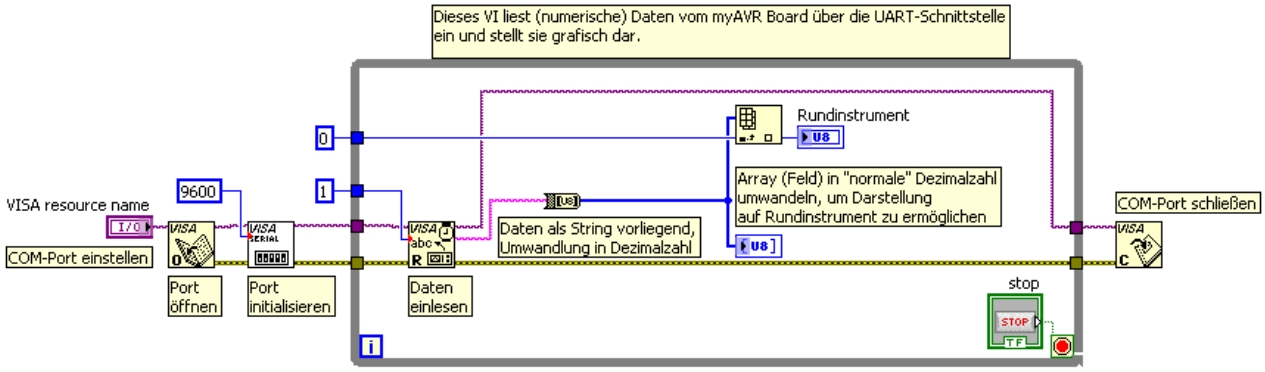
```
//-------------------------------------------------------------------
// Warte-Routine für 10 ms
// die Routine wartet inclusive Aufruf 10,06 ms
//-------------------------------------------------------------------
void myWait_10ms()
{
    __asm__ volatile (
        "push   r16\n"
        "ldi    r16,1\n"
        "myWait_10ms_3=:\n"
        "push   r16\n"
        "ldi    r16,48\n"
        "myWait_10ms_2=:\n"
        "push   r16\n"
        "ldi    r16,255\n"
        "myWait_10ms_1=:\n"
        "dec    r16     \n"
        "brne   myWait_10ms_1=\n"
        "pop    r16     ; Register wiederherstellen\n"
        "dec    r16     \n"
        "brne   myWait_10ms_2=\n"
        "pop    r16     ; Register wiederherstellen\n"
        "dec    r16     \n"
        "brne   myWait_10ms_3=\n"
        "pop    r16     ; Register wiederherstellen\n"
        :
        :
        );
}
//-------------------------------------------------------------------
// Initialisierungen
//-------------------------------------------------------------------
void init()
{
    // UART initialisieren
    sbi(UCSRB,3);   // TX aktiv
    sbi(UCSRB,4);   // RX aktivieren
    UBRRL=(uint8_t)(F_CPU/(BAUD*16L))-1;    // Baudrate festlegen
    UBRRH=(uint8_t)((F_CPU/(BAUD*16L))-1)>>8;   // Baudrate festlegen
    ADMUX=0x20;     // Port, Referenzspannung und Auflösung
    ADCSRA=0x85;    // Modus, Interrupt und Start
}

//////////////////////////////////////////////////////////////////////
// Main-Funktion
//////////////////////////////////////////////////////////////////////
main()
{
    init();     // Initialisierungen
    unsigned char wert;
    while (true)    // Mainloop-Begin
    {
        wert = getValueADCandWait();
        uartPutChar(wert);
        myWait_10ms();
    }
    // Mainloop-Ende
}
//-------------------------------------------------------------------
```

Virtuelles Instrument für *LabView*

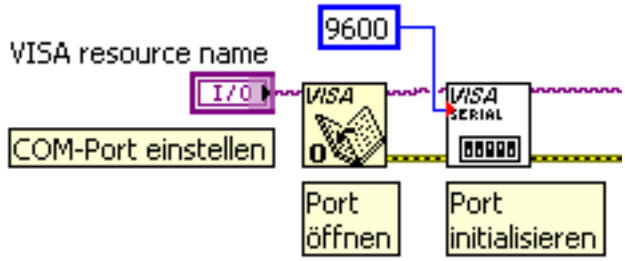
Im Folgenden wird der sogenannte „G-Code“ des Programmes erläutert.

Datei Oszipoti.vi:

Übersicht über den G-Code.

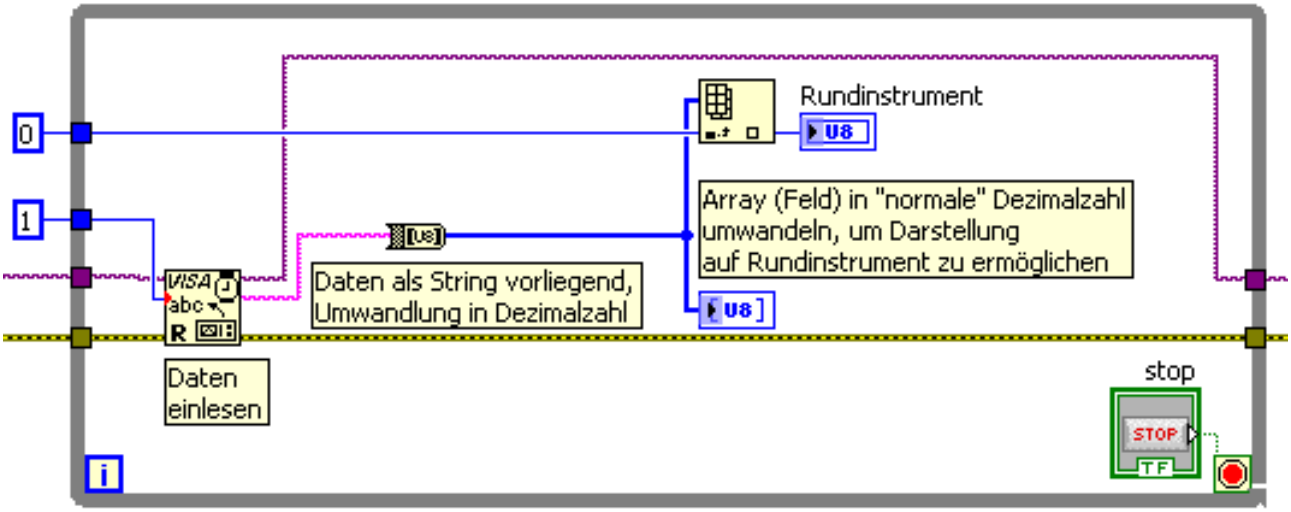


Teil 1



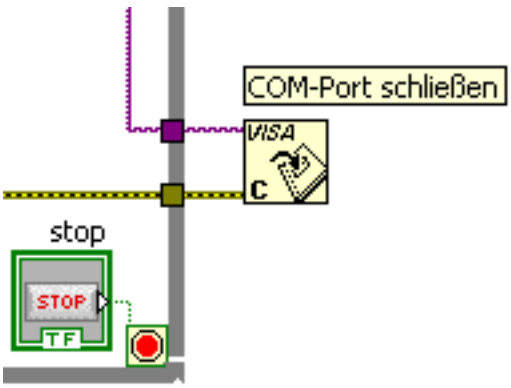
Hier wird der COM-Port geöffnet und die notwendigen Einstellungen festgelegt.
Die Baudrate wird auf 9600 gesetzt (für die restlichen Einstellungen werden die Standardwerte übernommen).

Teil 2



Dies ist der Hauptteil des Programmes. Der graue Rahmen ist eine `while`-Schleife, die das Programm kontinuierlich ausführt. Zuerst werden die Daten von der Schnittstelle eingelesen. Die Daten liegen als Text vor, deshalb müssen diese zur Darstellung auf dem Rundinstrument umgewandelt werden. Weil diese umgewandelten Daten als Feld (Array) vorliegen, muss die Dezimalzahl noch aus dem Array extrahiert werden, um auf dem Rundinstrument dargestellt zu werden. Der Signalgraph hat keine Probleme bei der Verarbeitung der Dezimalzahl als Array und kann daher direkt angeschlossen werden. Sobald auf die Stop-Schaltfläche auf dem Frontpanel gedrückt wird, wird die `while`-Schleife abgebrochen.

Teil 3



Nach Beendigung der `while`-Schleife wird der COM-Port geschlossen.
Achtung: Beim Klicken auf „Ausführung abbrechen“ geschieht dies nicht.