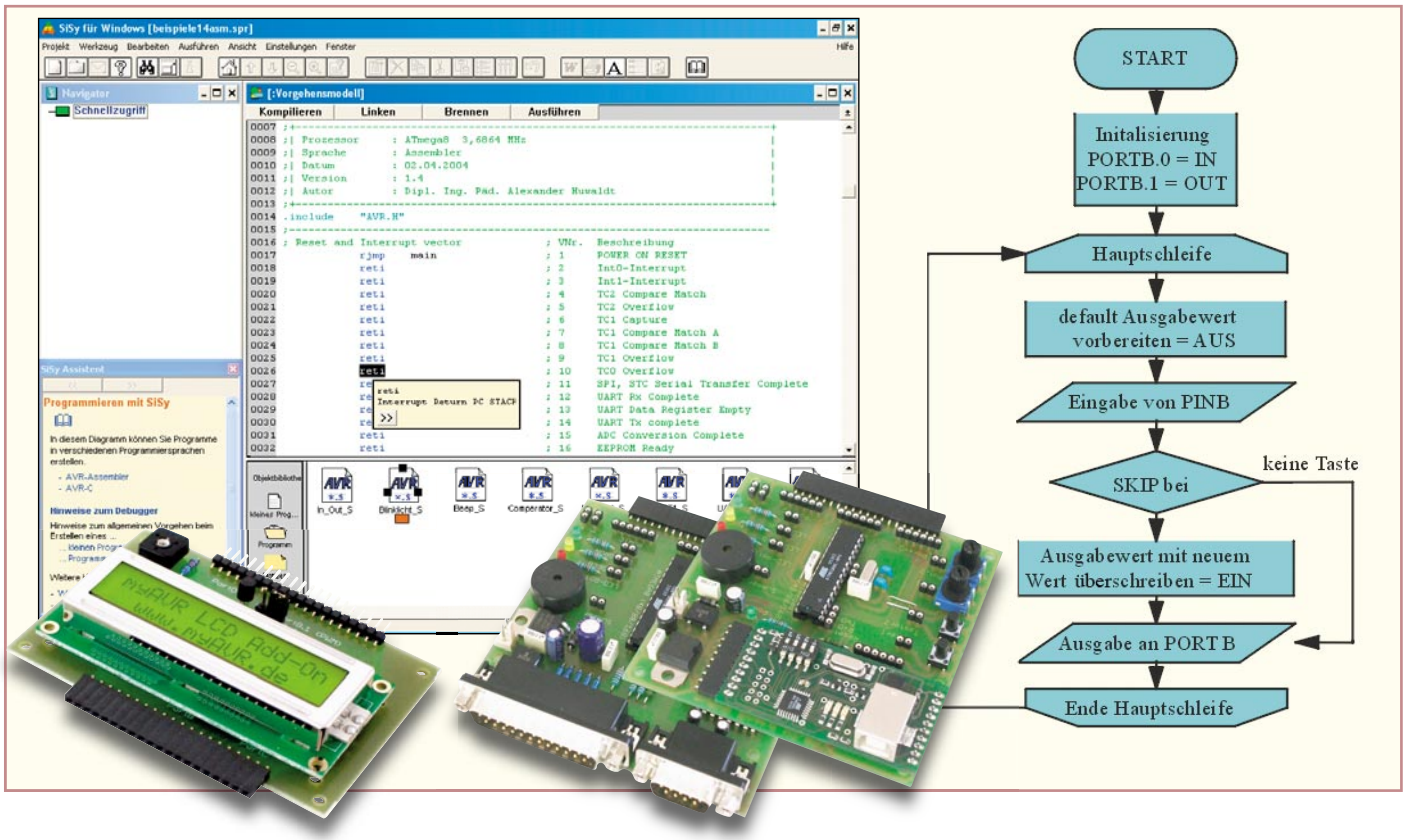

Mikrocontroller-Einstieg mit myAVR Teil 5

Keine Angst vor dem Einstieg in die Welt der Mikrocontroller-Programmierung! Die myAVR-Sets enthalten alles Nötige für den schnellen und fundierten Beginn der Programmierer-Karriere – Experimentierboard mit ATMEL-Controller, Lehrbuch, Softwarepaket, Kabel, sämtlichem Zubehör. Im fünften Teil unserer Serie zur Programmierung des AVR beschäftigen wir uns mit der Kommunikation zwischen AVR und seiner Umgebung über die serielle UART-Schnittstelle.

UART – seriell und asynchron

Der Austausch von Daten mit anderen Systemen ist eine wichtige Voraussetzung für die Arbeit mit Mikrocontrollern. Der Kommunikations-„Partner" muss nicht immer ein PC sein, auch Tastaturen, Displays und externe Speicher müssen vom Prozessor abgefragt bzw. mit Daten versorgt werden. Dabei hat sich in Mikrocontrollern die serielle Schnittstelle bis heute bewährt. Sie ist einfach aufgebaut, entsprechend einfach handhabbar und in den Controllern implementiert. Beim AVR-Controller finden wir neben den seriellen Interfaces I^2C und SPI die serielle Schnittstelle in Form eines integrierten UART-Bausteins (Universal Asynchron Receiver Transmitter). Dieser spezielle Schnittstellenbaustein unterstützt die serielle Datenübertragung nach der weitverbreiteten RS232-Norm und stellt damit eine einfache Möglichkeit dar, Daten zu versenden und zu empfangen.

Dies erfolgt über die Zweidrahtschnittstelle des AVR. Diese allerdings arbeitet mit den in Mikrocontrollersystemen üblichen TTL-Pegeln. Das ist in Ordnung, solange über kurze Strecken etwa mit einem angeschlossenen Speicherbaustein direkt kommuniziert werden soll.

Will man aber eine RS232-Verbindung zur seriellen Schnittstelle eines PCs auf-

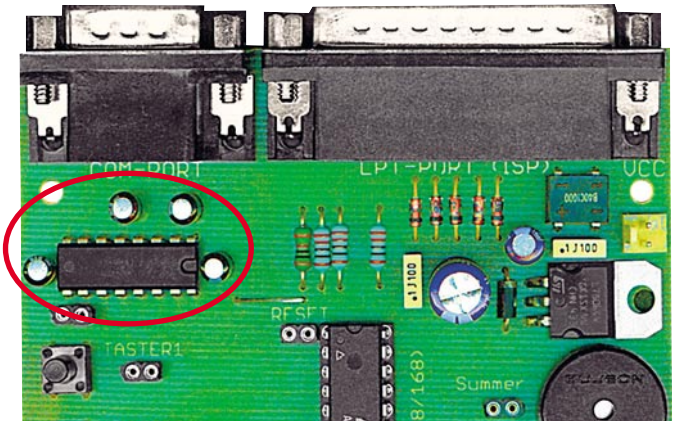

Bild 42: Der MAX 232 sorgt für die normgerechte Erzeugung der Pegel für die RS232-Schnittstelle aus der 5-V-Betriebsspannung. Darüber ist der RS232-Port zu sehen.

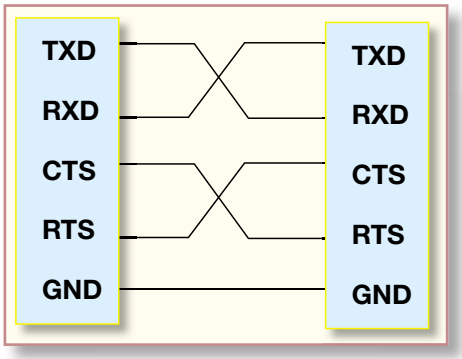

Bild 43: Die vollständige RS232-Verbindung mit Steuerleitungen.

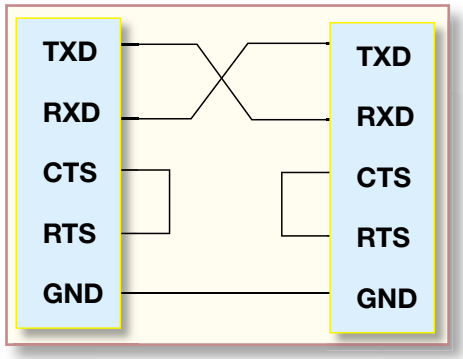

Bild 44: Die vereinfachte RS232-Verbindung als Nullmodem-Verbindung.

bauen, sind die Signalpegel an die RS232-Konventionen anzupassen. Dazu befindet sich auf dem myAVR-Board der bekannte Schnittstellen-Baustein MAX 232 (Abbildung 42), der eine Umwandlung der TTL-Pegel (0/5 V) in die RS232-Signalpegel ±12 V vornimmt. Der Vorteil dieses Bausteins: Man benötigt nur eine Versorgungsspannung von 5 V, die Umwandlung erfolgt durch interne Spannungswandler.

Damit und mit dem 9-poligen Sub-D-Steckverbinder auf dem Board sind die hardwaremäßigen Voraussetzungen erfüllt, um mit einer RS232-Standard-Schnittstelle, wie sie die serielle PC-Schnittstelle darstellt, kommunizieren zu können. Bleibt noch die eigentliche Verbindung. Abbildung 43 zeigt die vollständige Verbindung zwischen zwei RS232-Schnittstellen. Zusätzlich zu den Datenleitungen RXD (Empfangen) und TXD (Senden) finden wir hier noch einige Steuerleitungen, die jeweils die Bereitschaft von Sender und Empfänger zur Datenübertragung signalisieren. Für eine einfache Datenübertragung nutzt man allerdings oft die in Abbildung 44 dargestellte 3-Draht-Nullmodem-Verbindung. Ein so beschaltetes Nullmodem-Kabel befindet sich im Lieferumfang des myAVR-Sets, womit alle hardwareseitigen Voraussetzungen für den Datenaustausch zwischen PC und AVR erfüllt sind.

Diese geringe Leitungsanzahl (je Richtung eine plus Masse) wird durch das angewendete serielle, asynchrone Datenübertragungsverfahren mit einer sehr einfachen Datenstruktur ermöglicht. Asynchron heißt, dass mit den Daten kein Taktsignal übertragen wird und es keine Rückfrage über den ordnungsgemäßen Empfang der Daten gibt. Somit verliert man keine Zeit mit der Kontrolle der Datenübertragung. Allerdings müssen dazu Sender und Empfänger mit exakt der gleichen Übertragungsrate (Baudrate) arbeiten bzw. darauf eingestellt sein. Und beide Stellen müssen sich gegenseitig signalisieren, wenn die eigentliche Datenübertragung beginnt (Start-Bit) und endet (Stopp-Bit). Entsprechend sind die Datenbits eines Datenwortes jeweils von je einem Start- und Stopp-Bit eingerahmt.

So erreicht man eine relativ schnelle und sehr einfach aufzubauende Datenverbindung.

UART inside

Verschaffen wir uns zum Verständnis der Programmierung des AVR-UARTs zunächst mit Abbildung 45 einen Überblick über seinen Aufbau. Auch hier erfolgt die Programmierung über Register, die der Steuerung, der Festlegung der Baudrate und der Verwaltung der eigentlichen Daten dienen. Die Übertragungsrate wird direkt aus dem quarzstabilen Systemtakt generiert, die Genauigkeit ist für diese Art der Datenverbindung völlig ausreichend. Die „INT"-Anschlüsse weisen darauf hin: Der UART-Baustein ist interruptfähig, ein Blick in die Vektortabelle (Abbildung 46) bestätigt die drei möglichen Interrupt-Arten. Schließlich realisieren die Empfangs- und Sendebausteine die normgerechte RS232-Kommunikation nach außen.

UART initialisieren

Wollen wir uns nun mit der Programmierung des UARTs befassen, um schließlich Daten an einen angeschlossenen PC zu senden.

Baudrate einstellen

Wie gesagt, die Baudrate wird direkt aus dem Systemtakt abgeleitet. Die RS232-Konventionen sehen eine Reihe definierter Baudraten, z. B. 1200, 2400, 9600, 57 600

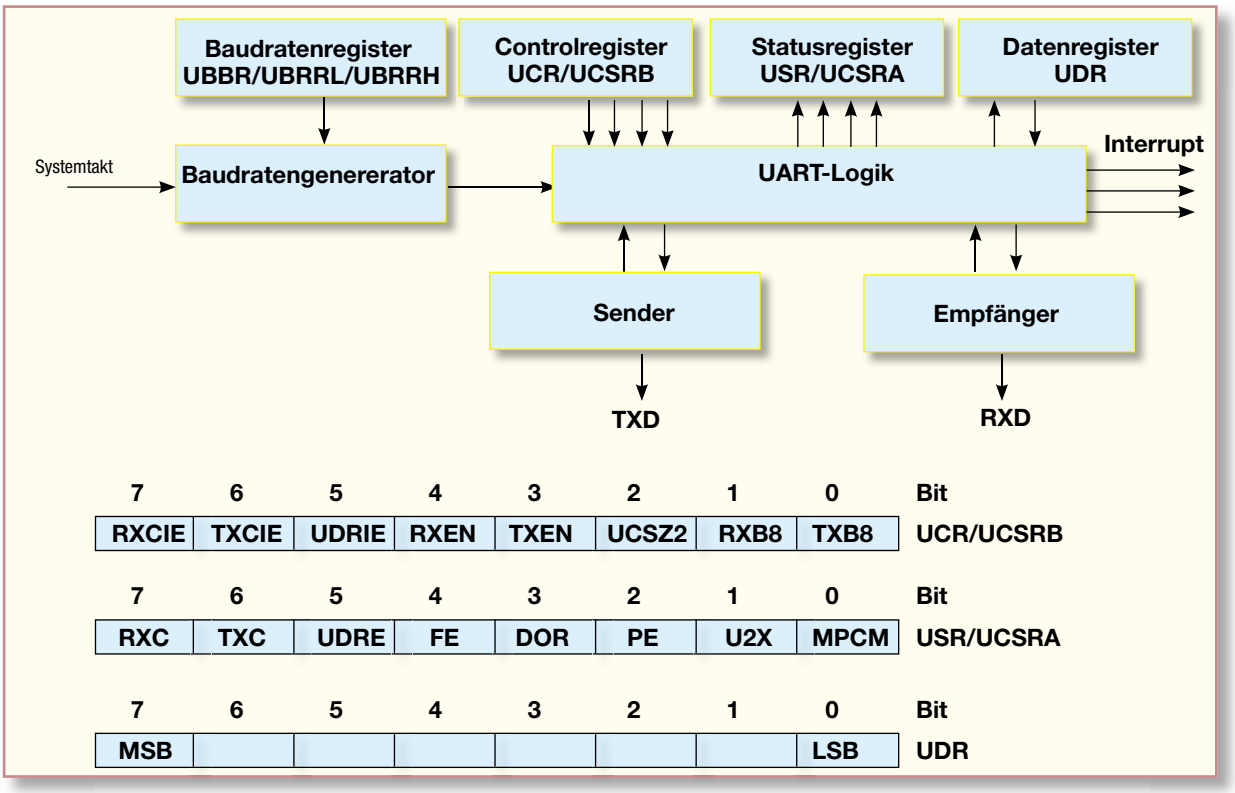

Bild 45: Der Aufbau des UART und seine Registerbelegung. Die Registerbezeichnungen können je nach Controllertyp variieren, die Funktionen sind jedoch kompatibel.

```
VNr.  Beschreibung
1     POWER ON RESET
2     Int0-Interrupt
3     Int1-Interrupt
4     TC2 Compare Match
5     TC2 Overflow
6     TC1 Capture
7     TC1 Compare Match A
8     TC1 Compare Match B
9     TC1 Overflow
10    TC0 Overflow
11    SPI, STC Serial Transfer Complete
12    UART Rx Complete
13    UART Data Register Empty
14    UART Tx complete
15    ADC Conversion Complete
16    EEPROM Ready
17    Analog Comparator
18    TWI (I²C) Serial Interface
19    Store Program Memory Ready
```

Bild 46: Der UART besitzt drei Interrupt-Quellen, die alternativ zum Pollingverfahren einsetzbar sind.

oder 115 200 Baud vor. Entsprechend ist ein Faktor in das Register UBRR zu laden. Wollen wir z. B. mit der gängigen Baudrate 9600 Baud übertragen, errechnet sich dieser Faktor wie folgt:

$$UBRR = \frac{\text{Taktfrequenz}}{16 \text{ x Baudrate}} - 1$$

also für 9600 Baud:

$$UBRR = \frac{3\,686\,400 \text{ Hz}}{16 \text{ x } 9600} - 1 = 23$$

Im späteren Programm kann man den Faktor entweder als Formel oder gleich das Ergebnis eintragen.

Empfänger/Sender initialisieren

Nach dem Einschalten des AVR sind die Empfänger- und Senderbausteine ausgeschaltet, einmal um Strom zu sparen und zum anderen um ungewollte Übertragungen und Störeffekte zu unterbinden.

Daher muss je nach Aufgabe der Sender und/oder Empfänger des UART aktiviert bzw. deaktiviert werden.

Dies erfolgt über das Kontroll-Register UCR/UCSRB (siehe Abbildung 45) und dort über die Bits 3 (Sender (TX) ein-/ausschalten) und 4 (Empfänger (RX) ein-/ausschalten).

Daten empfangen

Nachdem nun die Initialisierung abgeschlossen ist, kann der UART-Baustein Daten empfangen. Die Empfangssteuerung erfolgt über die beiden Register USR/UCSRA und UDR (siehe Abbildung 45). Dabei wird über das Statusregister USR/UCSRA über das Bit 7 (RXC, Receive complete) durch ständige Abfrage (Polling) überwacht, ob das Stopp-Bit gesendet wurde. Erst dann werden die empfangenen Daten im Datenregister UDR abgelegt und liegen hier zur Auswertung

```
getChar sbis USR,7       ; UCSRA=0x0B RXC= Bit 7
        rjmp getChar     ; warten, bis UDR bereit
        in   r16,UDR     ; Daten einladen
        ret              ; zurück zum Hauptprogramm
```

Bild 47: Das Unterprogramm zum Warten auf den Empfang eines Bytes.

```
putChar sbis USR,5       ; USR Bit 5 = UDRC
        rjmp putChar     ; warten, bis UDR bereit
        out  UDR, r16    ; Ausgabe
        ret              ; zurück zum Hauptprogramm
```

Bild 48: Das Unterprogramm der Senderoutine im Polling-Modus.

bereit. Abbildung 47 zeigt die zugehörige Programmierung über die „getChar“-Routine (Zeichen via Polling empfangen). Der Befehl sbis ist wieder einer der bereits bekannten Skip-Befehle. Er sorgt für die Abfrage des Zustands im Register USR/UCSRA. Ist das Bit 7 dort gesetzt, also der Empfang abgeschlossen, wird der fol-

```
;+------------------------------------------------------------------------+
;| Titel         : Beispiel UART senden für SiSy AVR-Board                |
;+------------------------------------------------------------------------+
;| Funktion      : sendet Test mit 9600Baud per serieller                 |
;|               : Schnittstelle                                          |
;| Schaltung     :                                                        |
;+------------------------------------------------------------------------+
;| Prozessor     : ATmega8  3,6864 MHz                                    |
;| Sprache       : Assembler                                              |
;| Datum         : 2.04.2004                                              |
;| Version       : 1.2                                                    |
;| Autor         : Dipl. Ing. Päd. Alexander Huwaldt                      |
;+------------------------------------------------------------------------+
.include    "AVR.H"
;-------------------------------------------------------------------------
; Reset and Interrupt vector              ; VNr.  Beschreibung
            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 (I²C) Serial Interface
            reti                          ; 19   Store Program Memory Ready
;-------------------------------------------------------------------------
; Start, Power ON, Reset
main:       ldi     r16,    hi8(RAMEND)
            out     SPH,    r16
            ldi     r16,    lo8(RAMEND)   ; Stack Initialisierung
            out     SPL,    r16           ; Init Stackpointer
            sbi     UCSRB, 3              ; USR=UCSRB=0x0A TXEN=Bit3 TX aktivieren
            ldi     r16,    23            ; Baudrate 9600 einstellen
            out     UBRRL, r16            ; Baudrate ins Register
mainloop:   ldi     r16,    'T'
            rcall   putChar
            ldi     r16,    'E'
            rcall   putChar
            ldi     r16,    'S'
            rcall   putChar
            ldi     r16,    'T'
            rcall   putChar
            ldi     r16,    '!'
            rcall   putChar
            rjmp    mainloop
;-------------------------------------------------------------------------
putChar:    sbis    UCSRA, 5              ; USR=UCSRA=0x0B UDRE=Bit5
            rjmp    putChar               ; warten, bis UDR bereit
            out     UDR,    r16           ; Ausgabe
            ret                           ; zurück zum Hauptprogramm
;-------------------------------------------------------------------------
```

Bild 49: Der Quellcode für die serielle Datenübertragung des Wortes TEST! an den PC.

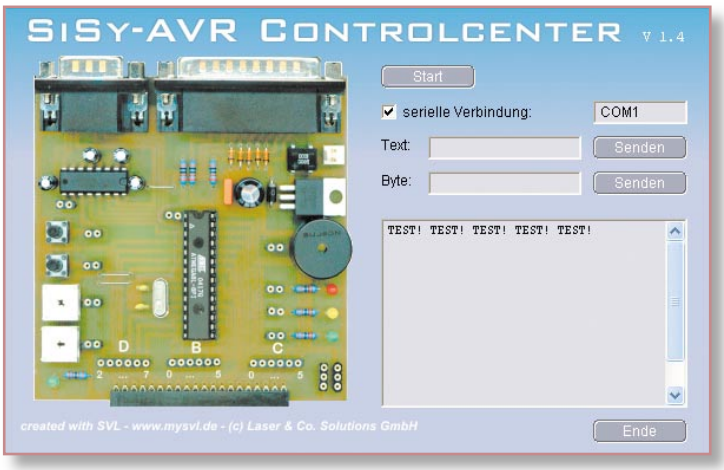

Bild 50: Unser Text ist im PC angekommen! Er wird über das Terminalprogramm des Controlcenters ausgegeben.

gende Rücksprungbefehl ignoriert, und es erfolgt das Einschreiben der Daten, die sich im UDR-Register befinden, in das Zwischenregister r16.

Daten senden

Ganz ähnlich läuft das Senden ab. Der UART wird durch das Eintreffen eines Bytes im Datenregister dazu aufgefordert, dieses auch zu senden – aber erst, wenn das Senden des vorhergehenden Bytes abgeschlossen ist. Die zugehörige Polling-Routine „putChar“ (Abbildung 48) ähnelt der Empfangsroutine, hier wird aber das Bit 5 des Registers USR/UCSRA abgefragt, bevor der Inhalt des Zwischenregisters in das Datenregister und damit an den Sender gegeben wird.

Die zu sendenden Daten werden Byte für Byte über das Hauptprogramm in das Zwischenregister geladen und hier eines nach dem anderen über Rücksprung zum Unterprogramm „putChar“ gesendet.

Wollen wir uns ansehen, wie das praktisch funktioniert.

Das Programm

Das einfache Programm zur Ausgabe eines Wortes über die serielle Schnittstelle ist schnell erklärt, da wir alle wesentlichen Bausteine ja bereits kennen.

Der Quellcode (Abbildung 49) für unser einfaches Sendeprogramm zeigt, dass in der Vektortabelle außer dem Power-on-Reset keine Interrupt-Quelle genutzt wird, da wir ja das Polling-Verfahren zur Abfrage der Sendebereitschaft verwenden.

Die Initialisierungsroutine wird hier ergänzt durch das Setzen des Registers UCR/UCSRB, hier wird, wie beschrieben, der Sendebaustein aktiviert. Danach erfolgt das Generieren der Baudrate.

Im Hauptprogramm finden wir die beschriebenen Routinen zum Aufrufen der auszugebenden Zeichen über das schließlich folgende Ausgabe-Unterprogramm „putChar“. Kompilieren, linken und brennen wir das Programm nun wie üblich über das Parallelkabel in unseren AVR. Erst, wenn das abgeschlossen ist, verbinden wir eine serielle Schnittstelle des Rechners (COMx) über das beschriebene Nullmodem-Kabel mit der seriellen Schnittstelle des myAVR-Boards.

Nach dem Öffnen des SiSy-AVR Controlcenters über den Button „Ausführen“, dem Herstellen der seriellen Verbindung und dem Starten des AVR über den Start-Button sendet dieser ständig die programmierten Zeichen in das Protokollfenster (Abbildung 50), bis das Programm wieder angehalten wird.

Das hier aufgeführte Programm ist natürlich nur als einfach zu lernende Programmiersequenz aufzufassen, die u. a. die Grundzüge der Ausgabe auf eine Anzeige beschreibt. Dass es noch wesentlich eleganter geht, ganze Zeichenketten „auf einen Schlag“ zu verschicken und zu empfangen, Nachrichten in den einzelnen Speicherarten, insbesondere dauerhaft im EEPROM zu speichern und aus diesen auszulesen, beschreibt der myAVR-Lehrgang ebenso ausführlich wie die Programmierung des UART als Empfänger.

Wir wollen uns zum Abschluss der Einführung in die AVR-Assemblerprogrammierung noch einmal kurz mit einer weiteren Möglichkeit befassen, wie man den AVR nutzen kann – die Verarbeitung analoger Signale über die analogen Eingänge des Controllers.

Analoge Signalverarbeitung

Erinnern wir uns noch an den Beginn unserer Serie, als wir Aufbau und Pin-Out des AVR-Controllers besprochen haben. Hier fallen uns auch diverse analoge Ports auf, darunter mehrere A/D-Wandler-Eingänge und ein Komparator. Die Nutzung des Letzteren wollen wir in unserem kurzen Beispiel genauer erarbeiten. Die ausführliche Nutzung der A/D-Wandler, die einen analogen Spannungswert in eine digitale Größe umwandeln, überlassen wir dem ausführlichen myAVR-Lehrgang.

Der Komparator des ATmega

Ein Komparator vergleicht zwei analoge Spannungen in einem bestimmten Spannungsbereich und gibt ein Signal aus, wenn eine der Spannungen größer oder kleiner als die andere (meist ein fester Referenzwert) ist.

Der Komparator des ATmega auf unserem myAVR-Board verbirgt sich hinter den Pins 12 (AIN 0, analog input 0) und Pin 13 (AIN 1, analog input 1). AIN 0 ist der Eingang für die Referenzspannung. Er ist so programmierbar, dass entweder eine externe oder intern erzeugte Referenzspannung nutzbar ist (Abbildung 51). Hat die zu kontrollierende Spannung an AIN 1 einen Wert unter der Referenzspannung, so erscheint am Ausgang des Komparators der Zustand „0“. Übersteigt der Spannungswert

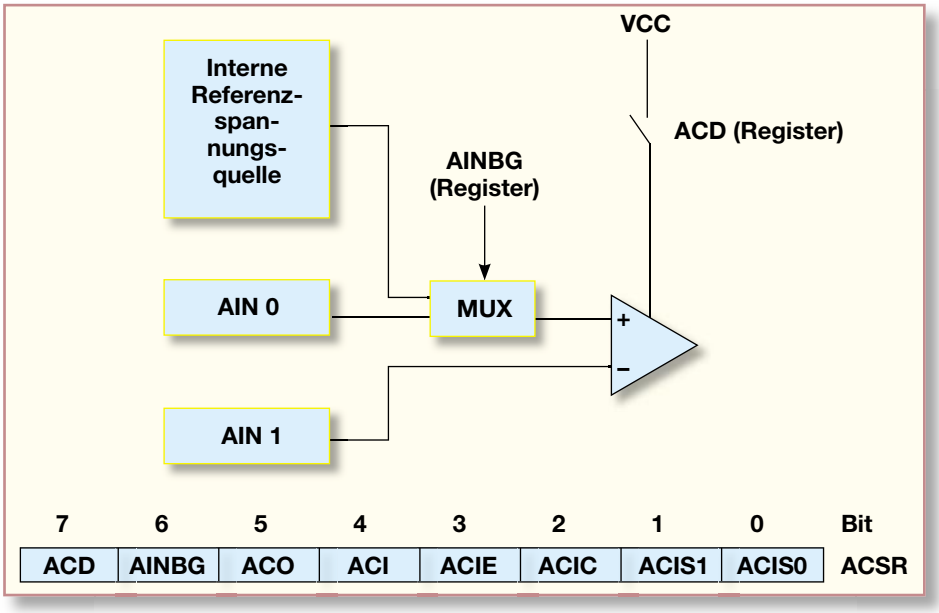

Bild 51: Der prinzipielle Schaltungsaufbau des Komparators und das zugehörige Steuerregister ACSR.

```
;+------------------------------------------------------------------------+
;| Titel          : Beispiel Komparator für SiSy AVR-Board                |
;+------------------------------------------------------------------------+
;| Funktion       : Zwei Analogwerte werden verglichen                    |
;| Schaltung      : PortD.6 & 7 an Poti, rote LED an PortB.0              |
;+------------------------------------------------------------------------+
;| Prozessor      : ATmega8  3,6864 MHz                                   |
;| Sprache        : Assembler                                             |
;| Datum          : 02.06.2004                                            |
;| Version        : 1.5                                                   |
;| Autor          : Dipl. Ing. Päd. Alexander Huwaldt                     |
;+------------------------------------------------------------------------+
.include    "AVR.H"
;-------------------------------------------------------------------------
; Reset and Interrupt vector              ; VNr.  Beschreibung
            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 (I²C) Serial Interface
            reti                          ; 19    Store Program Memory Ready
;-------------------------------------------------------------------------
; Start, Power ON, Reset
main:       ldi     r16,    hi8(RAMEND)
            out     SPH,    r16
            ldi     r16,    lo8(RAMEND)   ; Stack Initialisierung
            out     SPL,    r16
            cbi     ACSR,   7             ; Komparator ON
            cbi     ACSR,   6             ; Externe Referenzspannung
            sbi     DDRB,   0             ; PortB.0 Ausgang rote LED

;-------------------------------------------------------------------------
mainloop:   wdr
            in      r24,    ACSR          ; Komparatorstatus einlesen
            ldi     r25,    0b00000000    ; Ausgabewert LOW
            out     PORTB, r25
            sbrs    r24,    5             ; Wenn PD.6 / 7 Uin > Uref (ACO=high)
            ldi     r25,    0b00000001    ; Ausgabewert high
            out     PORTB, r25
            rjmp    mainloop
;-------------------------------------------------------------------------
```

Bild 52: Der Quellcode zum Komparatorprogramm.

die Referenzspannung, kippt der Komparator, und es erscheint der logische Zustand „1“ an seinem Ausgang. Genau das kann der Controller auswerten und z. B. eine Warnanzeige ausgeben (oder auch, in Erweiterung des vorherigen Kapitels, eine Meldung an den PC ausgeben bzw. auch eine Regelschaltung in Gang setzen). Die praktischen Anwendungen sind vielfältig, man denke nur an die Überwachung einer Temperatur.

Das Steuerregister

Das in Abbildung 51 bereits abgebildete Steuerregister des Komparators bietet zahlreiche Optionen zur Steuerung des Komparators. So kann er durch Setzen des Bits 7 wie die UART-Sender/-Empfänger ein- und ausgeschaltet werden. Durch Bit 6 ist die Umschaltung zwischen einer externen oder der internen Referenzspannung (fest: 1,22 V) möglich. Die Auswertung von Bit 5 ermöglicht es, den aktuellen Ausgangswert des Komparators auszulesen.

Und da wir den Komparator auch in der Interrupt-Vektortabelle als Interrupt-Quelle finden, zeigt Bit 4 des Steuerregisters an, ob ein Komparator-Interrupt-Ereignis vorliegt oder nicht. So kann der Komparator bei entsprechender ISR einen Interrupt auslösen – sofern man über Bit 2 des Steuerregisters diese Betriebsart freigibt und über Bit 1 und 0 das Interrupt-Ereignis definiert (Interrupt bei jedem Wechsel des Komparatorausgangs, bei fallender oder bei steigender Flanke).

In unserem folgenden Programmierbeispiel wollen wir den Komparator dazu nutzen, zwei analoge Spannungen an AIN 0 und AIN 1 zu überwachen und, wenn die Spannung an AIN 1 die an AIN 0 übersteigt, eine LED zu schalten. Dazu lassen wir die Interrupt-Möglichkeit wieder außen vor und überwachen den Zustand des Bits 5 im Komparatorregister ACSR im Polling-Betrieb, ähnlich unserem Vorgehen bei der seriellen Datenübertragung. Als Spannungsquellen nutzen wir praktischerweise die beiden Potentiometer des myAVR-Boards, die den Komparatoreingängen Spannungen zwischen 0 und +5 V liefern können.

Das Programm

Im Quellcode (Abbildung 52) sehen wir, dass gar nicht so viel Aufwand für die Lösung dieser einfachen Aufgabe getrieben werden muss.

Nach Interrupt-Vektortabelle und Stack-Initialisierung finden wir hier zunächst die beschriebene Aktivierung des Komparators über sein Register ACSR und dessen Bit 7. Danach folgt die Aktivierung des Bits 6, was bedeutet, dass die externe Referenzspannung genutzt werden soll. Und schließlich finden wir einen alten Bekannten wieder: Port B0 wird als Ausgabeport für die rote LED definiert.

Im Hauptprogramm folgen nun der Auslesebefehl des Komparator-Statusregisters, das Laden des Ausgaberegisters r25 mit „0“, also LED aus. Jetzt wird es spannender – wurde das Komparator-Statusregister (Bit 5) mit „1“ ausgelesen, bedeutet dies, dass die Eingangsspannung an Pin AIN 1 höher ist als die an Pin AIN 0. Jetzt wird Register r25 mit „1“ geladen und dies an Port B ausgegeben – die LED wird eingeschaltet! Abbildung 53 zeigt die Verdrahtung des myAVR-Boards für diese Aufgabe.

Das war’s – und das war es auch mit unserem kleinen Assembler-Lehrgang zum AVR-Controller. Wie gesagt, das Lehrgangsmaterial zum myAVR-Board enthält noch viele, vor allem weiterführende Beispiele und interessante Lösungen bestimmter Aufgaben. Wir gehen in der nächsten Folge zu einer Kurzeinführung der AVR-Programmierung in den Programmiersprachen C und BASCOM über. ELV

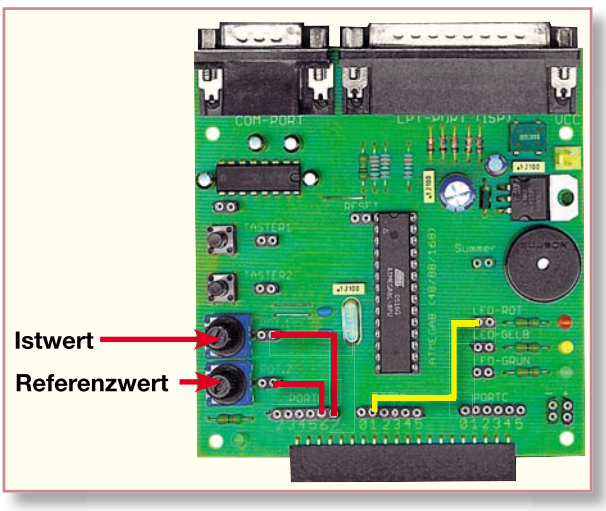

Bild 53: Die Verdrahtung des myAVR-Boards.