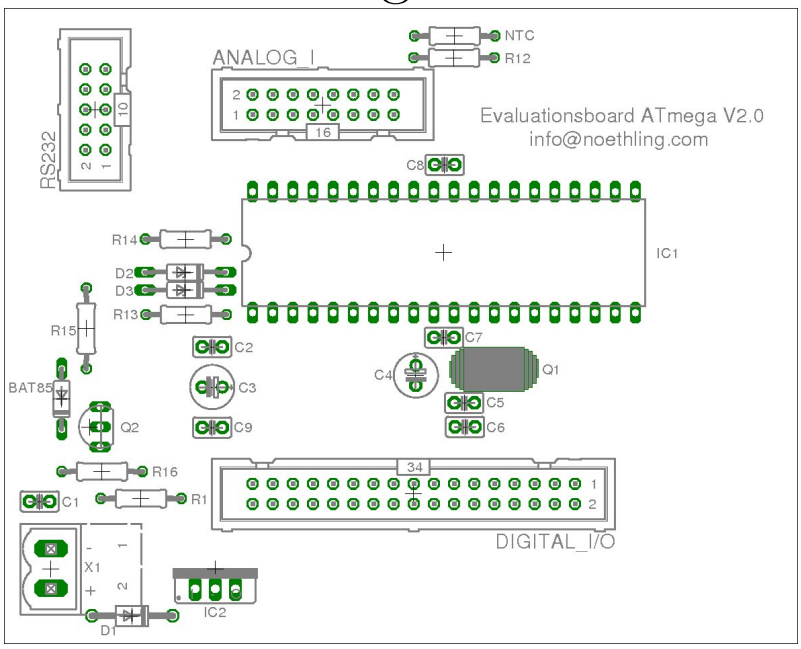
# Dokumentation

# Mikrocontroller-Evaluationboard ATmega V2.0




http://www.s-jordan.de
info@noethling.com

Stand: 29. Juli 2008

# Inhaltsverzeichnis

# 1 Installation der erforderlichen Software

Das vorliegende Mikrocontroller-Board kann wahlweise unter den Betriebssystemen Microsoft Windows 9x/NT/2000/XP oder unter Linux programmiert werden. Für beide Betriebssysteme wird die komplette Software im Internet zum Herunterladen angeboten. Zuerst wird die Installation unter Windows beschrieben. Anschließend werden für Linux-Benutzer entsprechende Quellen genannt.

## 1.1 Installation unter MS Windows 9x/NT/2000/XP

Um Ihren Mikrocontroller unter MS Windows 9x/NT/2000/XP programmieren zu können, benötigen Sie folgende Softwaretools, die alle im Internet kostenlos zum Herunterladen angeboten werden:

- WinAVR (Enthält den Compiler avr-gcc)
- AVR Studio 4 (Entwicklungsumgebung)
- PonyProg (Flash-Tool)

Selbstverständlich können auch andere Tools verwendet werden. Beachten Sie aber, dass nur die oben aufgelisteten Programme getestet wurden und für andere Software kein Support angeboten werden kann.

Im Folgenden wird beschrieben, wo Sie die benötigte Software herunterladen können und wie sie installiert wird.

### 1.1.1 WinAVR

Sie benötigen das Programmpaket „WinAVR". Es enthält den Compiler avr-gcc und wird daher für das AVR Studio 4 benötigt. Laden Sie daher folgende Datei aus dem Internet herunter:

- WinAVR
  `http://kent.dl.sourceforge.net/sourceforge/winavr/` ↩
  `WinAVR-20070525-install.exe`

Sollte die Download-Quelle nicht verfügbar sein, besuchen Sie bitte die Internetseite von WinAVR unter `http://winavr.sourceforge.net` und suchen die aktuelle Version von WinAVR im Download-Bereich. Anschließend installieren Sie das Programm bitte auf Ihrem Computer.

### 1.1.2 AVR Studio 4

Als Entwicklungsumgebung empfehlen wir das „AVR Studio 4", das vom Mikrocontroller-Hersteller Atmel direkt angeboten wird.

Laden Sie die folgende Datei von der Hersteller-Homepage herunter:

- AVR Studio 4.13
  `http://www.atmel.com/dyn/resources/prod_documents/aStudio4b528.exe`

Es ist möglich, dass die oben genannte Download-Quelle nicht mehr aktuell oder verfügbar ist, wenn Sie diese Dokumentation lesen, da der Hersteller die Software weiter entwickelt. Schauen Sie in diesem Fall, oder wenn Sie weitere Informationen zur Software erhalten möchten, direkt auf der Homepage unter `http://www.atmel.com/products/avr` im Bereich „Tools & Software"nach der jeweils aktuellen Version des AVR Studio.

Installieren Sie anschließend das AVR Studio.

### 1.1.3 PonyProg2000

Um den von der Entwicklungsumgebung generierten Maschinencode über die In-System-Programming(ISP)-Schnittstelle auf den Mikrocontroller zu schreiben, wird noch das Programm PonyProg benötigt. Laden Sie folgende Datei herunter:

- PonyProg v2.06f
  `http://www.lancos.com/e2p/V2_06/ponyprogV206f.zip`

Auch hier kann es passieren, dass die Version nicht mehr aktuell ist. Den jeweils neuesten Stand finden Sie auf folgender Website: `http://www.lancos.com/ppwin95.html`

Entpacken und installieren Sie das Programm auf Ihrem Computer.

## 1.2 Installation unter Linux

Um Ihren Mikrocontroller unter Linux programmieren zu können, benötigen Sie folgende Softwaretools, die alle im Internet kostenlos zum Herunterladen angeboten werden:

- Einen Editor Ihrer Wahl
- CDK4AVR (Enthält den Compiler avr-gcc)
- PonyProg (Flash-Tool)

### 1.2.1 CDK4AVR

Es gibt unter Linux verschiedene Softwarepakete, die den Compiler avr-gcc enthalten. Wir empfehlen das Paket CDK4AVR. Laden Sie sich die Software von folgender Homepage herunter:

- CDK4AVR
  `http://cdk4avr.sourceforge.net/`

Bitte installieren Sie die Pakete auf Ihrem System.

### 1.2.2 PonyProg2000

Um den vom Compiler generierten Maschinencode über die In-System-Programming(ISP)-Schnittstelle auf den Mikrocontroller zu schreiben, wird noch das Programm PonyProg benötigt. Laden Sie folgende Datei herunter:

- PonyProg2000 v2.06c BETA
  `http://www.lancos.com/e2p/V2_06/ponyprog-2.06c-rh70.tar.gz`

Hier kann es passieren, dass die Version nicht mehr aktuell ist. Den jeweils neuesten Stand finden Sie auf folgender Website: `http://www.lancos.com/ppwin95.html`

Entpacken Sie das Programm. Führen Sie das Programm als User root aus oder stellen Sie sicher, dass Ihr Benutzer Schreibrechte für die serielle Schnittstelle besitzt.

# 2 Erstellen eines Mikrocontroller-Programms

Nachdem die Entwicklungstools nun auf Ihrem Rechner installiert sind, können Sie mit dem Erstellen Ihres ersten Programms für den ATmega32 beginnen.

## 2.1 Anlegen eines neuen Projektes im AVR Studio 4

Unter Windows starten Sie das AVR Studio 4. Das Startfenster „Welcome to AVR Studio 4“ zeigt Ihnen jetzt die zuletzt bearbeiteten Projekte an. Beim ersten Starten wird diese Liste leer sein, später können Sie eines Ihrer zuletzt bearbeiteten Projekte aber damit bequem per Doppelklick auf den Projektnamen öffnen und sofort weiter bearbeiten. Zum Anlegen eines neuen Projektes klicken Sie nun aber einmal auf die Schaltfläche „New Project“. Unter „Project type:“ markieren Sie bitte den Eintrag „AVR GCC“. Wählen Sie dann per Klick auf die Schaltfläche „...“ den Speicherort auf der Festplatte des Rechners aus. Merken Sie sich diesen Ort und wählen ihn mit „Select“ aus. In das Feld „Project name:“ tragen Sie einen Namen ein, beispielsweise „Test“. Aktivieren Sie die Checkbox „Create initial file“. Die Datei mit dem C-Quellcode erhält automatisch denselben Namen wie der Projektname, was auch so übernommen werden kann. Klicken Sie nun auf „Next >>“. Wählen Sie im nächsten Fenster aus der linken Liste den „AVR Simulator“ aus und als Device aus der rechten Liste den „ATmega32“. Klicken Sie jetzt auf „Finish“ und Sie können im rechten oberen Fenster mit der Eingabe Ihres Programmcodes in C beginnen.

Unter Linux starten Sie einfach einen Texteditor Ihrer Wahl.

## 2.2 Erstellen des C-Programms

Sämtliche Funktionen und Register des Mikrocontrollers sind im Handbuch des Herstellers beschrieben. Dieses wird unbedingt empfohlen. Sie können es unter folgender Adresse herunterladen:

- `http://www.atmel.com/dyn/resources/prod_documents/doc2503.pdf`

Sehr hilfreiche Informationen können Sie auch dem AVR-GCC-Tutorial von mikrocontroller.net entnehmen:

- `http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial`

### 2.2.1 Header-Dateien

Damit Sie die Ein- / Ausgänge und Register des Mikrocontrollers in Ihrem Programm mit den selben Namen ansprechen können, wie sie auch im Handbuch beschrieben sind, sollten Sie zu Beginn Ihres Programms folgende Headerdateien einbinden:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdint.h>
#include <stdio.h>
```

### 2.2.2 Datentypen

In Tabelle 2.1 sind die elementaren ganzzahligen Datentypen dargestellt.

| Datentyp | Speichergröße | Wertebereich |
|---|---|---|
| signed char | 8 Bit | -127 ... +127 |
| unsigned char | 8 Bit | 0 ... 255 |
| short | 16 Bit | -32.767 ... 32.767 |
| unsigned short | 16 Bit | 0 ... 65.535 |
| long | 32 Bit | -2.147.483.647 ... +2.147.483.647 |
| unsigned long | 32 Bit | 0 ... 4.294.967.295 |
| long long | 64 Bit | $-9 \cdot 10^{18}$ ... $+9 \cdot 10^{18}$ |
| unsigned long long | 64 Bit | 0 ... $18 \cdot 10^{18}$ |

**Tabelle 2.1:** Elementare ganzzahlige Datentypen

### 2.2.3 Programmaufbau

Neben der in C immer notwendigen main-Funktion kann der Mikrocontroller auch sogenannte Interrupts auslösen. Beim Auftreten eines Interrupts wird automatisch die Hauptfunktion unterbrochen und in die entsprechende Interrupt-Subroutine gesprungen. Von der Syntax her sind diese Subroutinen normale Funktionen, nur dass diese nicht von einer anderen Funktion aufgerufen werden, sondern eben von einem eintretenden Ereignis. Prioritäten für verschiedene Interrupts können Sie beim ATmega32 leider nicht selbst festlegen. Als Beispiel sehen Sie einmal eine Interrupt-Subroutine (ISR), die aufgerufen wird, wenn die Analog-/Digitalwandlung abgeschlossen ist:

```
ISR (ADC_vect)
{
  analogwert = ADCL;
  analogwert += (unsigned long int)ADCH<<8;
}
```

Diese liest z.B. die Ergebnisregister (zuerst die unteren 8 Bit, danach die oberen 2 Bit) des A/D-Wandlers aus und speichert diese in der globalen Variable analogwert (Datentyp: unsigned short), die dann in der Hauptfunktion weiter verwendet werden kann. Bei solch kurzen Berechnungen kann die Reaktion auf den Interrupt direkt in der Interrupt-Subroutine erfolgen. Werden umfangreichere Aktionen nötig, ist es empfehlenswert, in der ISR lediglich einen Merker zu setzen, und die Aktion in der Hauptfunktion durchzuführen, damit der Mikrocontroller weiter in der Lage bleibt, auf andere auftretende Interrupts zu reagieren. Um die Interrupts zu aktivieren, müssen aber häufig erst bestimmte Register beschrieben werden. Die Anweisung

```
TIMSK=(1<<TOIE0)|(1<<TOIE1);
```

sorgt beispielsweise dafür, dass beim Überlauf der Timer 0 und 1 ein Interrupt ausgelöst wird. Konkret bedeutet die Anweisung, dass in das Register TIMSK die Bits für TOIE0 und TOIE1 auf 1 gesetzt werden, die restlichen Bits werden auf 0 gesetzt. Die Bedeutung der einzelnen Bits der Register können Sie dem Handbuch entnehmen. Andere Register werden analog zu diesem Beispiel konfiguriert.

Um die Interrupts aber auch wirklich global einzuschalten, ist noch folgender Aufruf nötig:

```
sei();
```

Mit folgender Anweisung lassen sich die Interrupts wieder global ausschalten:

```
cli();
```

Zum Beispiel ist es notwendig, vor dem Schreiben oder Lesen des EEPROMs die Interrupts zu deaktivieren und erst danach wieder zu aktivieren, weil eine Unterbrechung des Schreib- / Lesezugriffs zu undefinierten Zuständen (Absturz) führen kann.

Mit entsprechenden Registern können Sie z.B. auch festlegen, ob einzelne Anschlüsse Ein- oder Ausgänge sein sollen.

Nach dem Setzen der Register zu Beginn der main-Funktion ist es empfehlenswert, die Hauptfunktion durch eine Endlosschleife aktiv zu halten. In dieser Endlosschleife finden dann die regelmäßig zu wiederholenden Programmanweisungen Platz und auch die Reaktionen auf die in Interrupt-Subroutinen gesetzten Merker. Beispielsweise kann dies durch

```
while(1)
{
  ... Programmanweisungen ...
}
```

erreicht werden.

Ein umfangreich kommentiertes Beispielprogramm finden Sie im Anhang dieser Dokumentation und sollte Ihnen die Programmierung des ATmega32 an einer konkreten Anwendung verdeutlichen.

## 2.3 Compilieren des Programms

Vergessen Sie nicht, Ihr Programm regelmäßig per „File“, „Save“ zu sichern.

Wenn Sie mit dem Programm – bzw. einer Programmänderung – fertig sind, können Sie das Programm compilieren.

### 2.3.1 Complilieren unter Windows

Im AVR Studio unter Windows funktioniert das Compilieren mittels „Build“, „Build“ oder ganz einfach per Druck auf die Taste „F7“. Sie erhalten jetzt einige Compiler-Meldungen im unteren Bildschirmbereich. Warnungen werden dabei mit einem gelben und Fehler mit einem roten Punkt in der entsprechenden Meldungszeile gekennzeichnet. Wenn Sie

alles richtig gemacht haben (syntaktisch zumindest), sehen Sie nur grüne Meldungen und die Zeile „Build succeeded with 0 Warnings...“. Ansonsten helfen Ihnen hoffentlich die Meldungen bei der Fehlersuche weiter. Weiterhin sehen Sie die Speicherausnutzung des Mikrocontrollers in Bezug auf den Programm- und Datenspeicher in Prozent. Nach dem erfolgreichen Compilieren (es sind höchstens Warnungen, aber keine Fehler aufgetreten) können Sie nun das Programm auf den Mikrocontroller laden.

### 2.3.2 Compilieren unter Linux

Unter Linux sind folgende Schritte zum Erzeugen des hex-Files notwendig:

- Öffnen Sie eine Konsole
- Wechseln Sie in das Verzeichnis, in dem die C-Quellcode-Datei liegt
- Führen Sie den folgenden Befehl aus:
  `/opt/cdk4avr/bin/avr-gcc-3.4.5 NAME.c -o NAME.elf -mmcu=atmega32`
- Führen Sie den folgenden Befehl aus:
  `/opt/cdk4avr/bin/avr-objcopy -O ihex -j .text -j .data ↩`
  `NAME.elf NAME.hex`

Um Informationen zum Speicherbedarf des Programms zu erhalten, verwenden Sie folgenden Befehl: `/opt/cdk4avr/bin/avr-size NAME.elf`

Das lässt sich selbstverständlich auch mittels make oder einem Shell-Skript automatisieren.

# 3 Anschließen des Mikrocontroller-Evaluationboards an den PC

Verbinden Sie das Mikrocontroller-Evaluationboard zunächst über die zweipolige Anschlussklemme mit der Betriebsspannung von 7 ... 30 V Gleichspannung. Halten Sie dabei unbedingt die korrekte Polarität ein!

Danach verbinden Sie die 10-polige Pfostenbuchse des Sub-D-Adapterkabels (siehe Abb. 3.1) mit dem 10-poligen Wannenstecker (mit „RS232“ beschriftet) auf dem Evaluationboard. Die Sub-D-Buchse verbinden Sie über ein 1:1-Verlängerungskabel mit der seriellen Schnittstelle Ihres PCs.

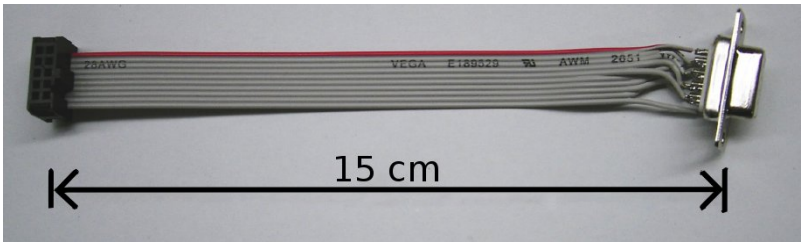



**Abbildung 3.1:** Sub-D-Adapterkabel

# 4 Flashen des Mikrocontrollers

## 4.1 Starten des Programms „PonyProg2000“

Das Flashen des Mikrocontrollers (das Übertragen des compilierten Programmcodes in den Flash-Speicher des Mikrocontrollers) erfolgt unter Windows und unter Linux mit dem Programm „PonyProg2000“. Starten Sie dieses Programm und schließen Sie das Begrüßungsfenster „About PonyProg2000“ mit einem Klick auf „OK“. Die beiden nachfolgenden Notizen ebenfalls mit „OK“ bestätigen.

## 4.2 Device auswählen

Wählen Sie nun als erstes den richtigen Mikrocontroller unter „Device“ - „AVR micro“ - „ATmega32“ aus.

## 4.3 Schnittstelle einrichten

Als nächstes wird über „Setup“ - „Interface Setup...“ der entsprechende Anschluss des Computers ausgewählt. Wählen Sie dazu „Serial“ sowie aus dem Drop-Down-Feld den Wert „SI Prog I/O“. Wählen Sie die COM-Schnittstelle aus, an die der RS232/ISP-Adapter angeschlossen ist und lassen Sie die Checkboxen unter „Select Polarity of the Control lines“ alle deaktiviert. Ein Klick auf „Probe“ sollte die Meldung „Test Ok“ bewirken. Schließen Sie diese sowie das „I/O port setup“ jeweils mit einem Klick auf „OK“.

Über „Setup“ - „Calibration“ wird anschließend die serielle Übertragung kalibriert. Beachten und bestätigen Sie die angezeigte Meldung mit „Yes“ und nach einigen Sekunden wird die Meldung „Calibration OK“ angezeigt, die wieder mit „OK“ quittiert werden kann.

## 4.4 Fuses bearbeiten

Als nächstes wählen Sie bitte „Command“ - „Security and Configuration Bits...“ aus und klicken nun auf „Read“. Hier ist eine erhöhte Vorsicht geboten, denn fehlerhaft gesetzte Fuses können dazu führen, dass der Mikrocontroller z.B. nicht mehr startet und somit nicht mehr angesprochen werden kann. Beachten Sie deshalb auch die Hinweise im ATmega Handbuch.

Damit der Mikrocontroller mit dem externen Quarzoszillator mit 16 MHz läuft, setzt man die Häkchen wie in Abb. 4.1. Klicken Sie dann auf „Write“, um die Werte zum Mikrocontroller zu übertragen.

## 4.5 Software übertragen

Nun kann das compilierte Programm, das als HEX-File vorliegt, auf den Mikrocontroller geschrieben werden. Öffnen Sie zunächst das File, indem Sie auf „File“ - „Open Program (FLASH) File...“ klicken. Wählen Sie als Dateityp „*.hex“ und suchen Sie nun in Ihrer Verzeichnisstruktur das HEX-File. Gewöhnlich findet sich dieses im Projektverzeichnis,

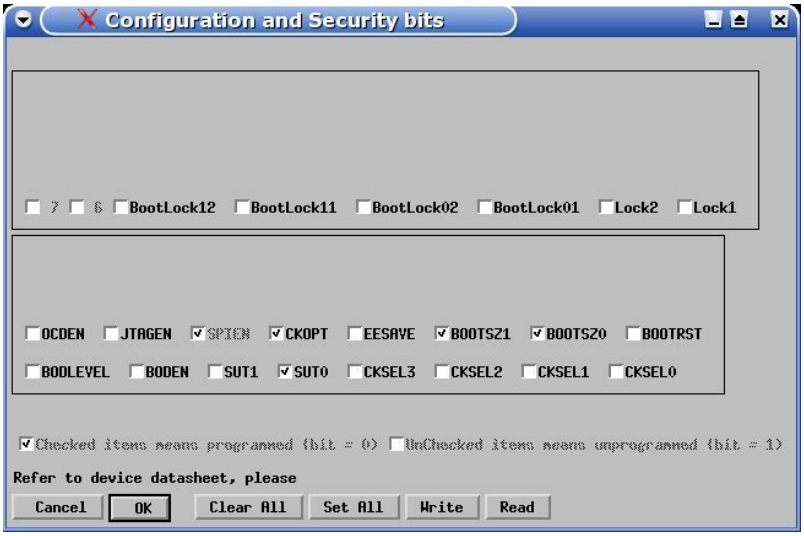



**Abbildung 4.1:** Configuration and Security bits - Betrieb mit externem Quarz und deaktiviertem JTAG-Interface

welches Sie im „AVR Studio“ ausgewählt haben, und dort im Unterverzeichnis „default“. Unter Linux finden Sie das HEX-File im selben Verzeichnis wie die C-Quelldatei auch. Sie sehen nun Ihre Programmanweisungen im hexadezimalen Code. Diese können Sie jetzt auf den Mikrocontroller schreiben. Das funktioniert mit der Funktion „Command“ - „Write Program (FLASH)“. Den Hinweis, dass das alte Programm auf dem Mikrocontroller verloren gehen wird, bestätigen Sie mit „Yes“. Im Folgenden wird das Programm zuerst auf den Mikrocontroller geschrieben und anschließend noch einmal ausgelesen und verifiziert. Die Meldung „Write successful“ können Sie erneut mit „OK“ quittieren. Der Mikrocontroller wird automatisch neu gestartet und Ihr neues Programm wird ausgeführt. Bei manchen PCs ist es allerdings nötig, nach dem Flashen die serielle Datenverbindung zwischen Rechner und Mikrocontroller wieder zu unterbrechen, damit der Mikrocontroller wieder anläuft.

## 4.6 Hinweis

Da das „PonyProg2000“ Ihre Einstellungen nach dem Schließen beibehält, können Sie beim nächsten Flashen die Schritte „Device auswählen“ und „Schnittstelle einrichten“ auslassen. Die restlichen Schritte sind obligatorisch.

# 5 Temperaturmessung

Auf Wunsch kann das Evaluationboard mit einer Temperaturmessung ausgestattet werden. Sind der Widerstand R12 sowie der NTC-Widerstand bestückt, ergibt sich die elektrische Verschaltung gemäß Abb. 5.1.

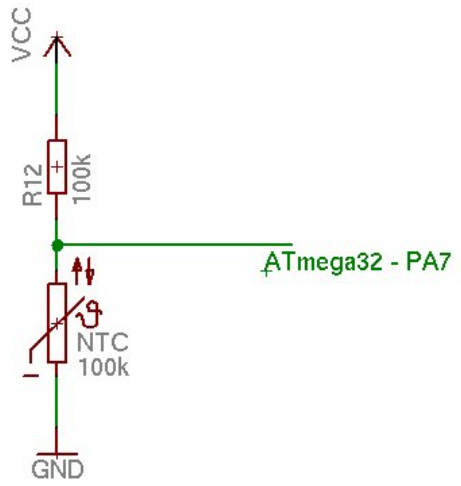



**Abbildung 5.1:** Elektrische Verschaltung des Temperatursensors

Gemäß der Spannungsteilerregel ergibt sich die Spannung am Analogeingang (0 ... 5 V) und damit der gewandelte Wert $adc$ (0 ... 1023) in Abhängigkeit des Wertes vom Temperaturwiderstand $R_{NTC}$:

$$adc = \frac{1023}{R_{NTC} + 100k\Omega} * R_{NTC} \tag{5.1}$$

Da für die Berechnung der Temperatur in K oder °C der Widerstandswert des NTCs benötigt wird, muss die Formel also nach $R_{NTC}$ umgestellt werden:

$$R_{NTC} = \frac{adc * 100k\Omega}{1023 - adc} \tag{5.2}$$

Nach dem Datenblatt des NTC-Thermistors lässt sich nun aus dem Widerstand des NTCs die Temperatur $T$ in Kelvin berechnen:

$$T = \frac{1}{A + B \cdot ln(\frac{R_{NTC}}{100k\Omega}) + C \cdot ln^2(\frac{R_{NTC}}{100k\Omega}) + D \cdot ln^3(\frac{R_{NTC}}{100k\Omega})} \tag{5.3}$$

mit:

$$A = 3,354016 \cdot 10^{-3}; B = 2,46038 \cdot 10^{-4}; C = 3,40538 \cdot 10^{-6}; D = 1,03424 \cdot 10^{-7}$$

# A Stückliste

Falls das Evaluationboard als Bausatz erworben wurde, sind die Bauteile, mit denen die Platine zu bestücken ist, der Tabelle A.1 zu entnehmen.

| Postition | Symbol(e) | Bezeichnung | Anzahl |
|---|---|---|---|
| 1 | - | Platine: Evaluationboard V2.0 | 1 |
| 2 | IC1 | ATmega32, DIL40 | 1 |
| 3 | - | Sockel für IC1, 40-polig | 1 |
| 4 | IC2 | Spannungsregler, $\mu$A7805, 5 V, 1 A | 1 |
| 5 | - | Kühlkörper für IC2 | 1 |
| 6 | RS232 | Wannenstecker, 10-polig, gerade | 1 |
| 7 | ANALOG_I | Wannenstecker, 16-polig, gerade | 1 |
| 8 | DIGITAL_I/O | Wannenstecker, 34-polig, gerade | 1 |
| 9 | X1 | Anschlussklemme, 2-polig, RM 5.08 mm | 1 |
| 10 | Q1 | Quarz, 16 MHz | 1 |
| 11 | Q2 | NPN-Transistor, BC546B | 1 |
| 12 | D1 | Diode, 1N4007 | 1 |
| 13 | D2, D3 | Z-Diode, 5.1 V, 0.5 W | 2 |
| 14 | BAT85 | Schottky-Diode, 30 V, 0.2 A | 1 |
| 15 | R1, R15 | Widerstand, 10 kΩ | 2 |
| 16 | R13, R14 | Widerstand, 4.7 kΩ | 2 |
| 17 | R16 | Widerstand, 33 kΩ | 1 |
| 18 | C1 | Folienkondensator, 330 nF, 50 V | 1 |
| 19 | C2, C7, C8, C9 | Folienkondensator, 100 nF, 63 V | 4 |
| 20 | C3, C4 | Elektrolytkondensator, 47 $\mu$F, 35 V | 2 |
| 21 | C5, C6 | Keramikkondensator, 22 pF, 63 V | 2 |
| Optional: | | | |
| 9 | X1 | WAGO Lötstifte | 2 |
| 22 | NTC | Temperatursensor, NTC, 100 kΩ | 1 |
| 23 | R12 | Widerstand, 100 kΩ | 1 |

**Tabelle A.1:** Stückliste des Evaluationboards

# B Anschlussbelegungen

Die Anschlussbelegung des Wannensteckers „DIGITAL_I/O“ ist in Tab. B.1 zu sehen.

| Anschluss Wannenstecker | Anschluss ATmega | Funktion |
|---|---|---|
| 1 | 19 | PD5 (OC1A) |
| 2 | 20 | PD6 (ICP1) |
| 3 | 18 | PD4 (OC1B) |
| 4 | - | +7...30V |
| 5 | 17 | PD3 (INT1) |
| 6 | - | +7...30V |
| 7 | 16 | PD2 (INT0) |
| 8 | - | +7...30V |
| 9 | 15 | PD1 (TXD) |
| 10 | - | GND |
| 11 | 14 | PD0 (RXD) |
| 12 | - | GND |
| 13 | - | GND |
| 14 | - | GND |
| 15 | - | +5V |
| 16 | 21 | PD7 (OC2) |
| 17 | - | +5V |
| 18 | - | +5V |
| 19 | 8 | PB7 (SCK) |
| 20 | 22 | PC0 (SCL) |
| 21 | 7 | PB6 (MISO) |
| 22 | 23 | PC1 (SDA) |
| 23 | 6 | PB5 (MOSI) |
| 24 | 24 | PC2 (TCK) |
| 25 | 5 | PB4 (SS) |
| 26 | 25 | PC3 (TMS) |
| 27 | 4 | PB3 (OC0/AIN1) |
| 28 | 26 | PC4 (TDO) |
| 29 | 3 | PB2 (INT2/AIN0) |
| 30 | 27 | PC5 (TDI) |
| 31 | 2 | PB1 (T1) |
| 32 | 28 | PC6 (TOSC1) |
| 33 | 1 | PB0 (XCK/T0) |
| 34 | 29 | PC7 (TOSC2) |

**Tabelle B.1:** Anschlussbelegung des Wannensteckers „DIGITAL_I/O“

Die Anschlussbelegung des Wannensteckers „ANALOG_I“ ist in Tab. B.2 zu sehen.

| Anschluss Wannenstecker | Anschluss ATmega | Funktion |
|---|---|---|
| 1 | 40 | PA0 (ADC0) |
| 2 | - | +5V |
| 3 | 39 | PA1 (ADC1) |
| 4 | - | +5V |
| 5 | 38 | PA2 (ADC2) |
| 6 | - | GND |
| 7 | 37 | PA3 (ADC3) |
| 8 | - | GND |
| 9 | 36 | PA4 (ADC4) |
| 10 | - | GND |
| 11 | 35 | PA5 (ADC5) |
| 12 | - | GND |
| 13 | 34 | PA6 (ADC6) |
| 14 | - | +7...30V |
| 15 | 33 | PA7 (ADC7) |
| 16 | - | +7...30V |

**Tabelle B.2:** Anschlussbelegung des Wannensteckers „ANALOG_I“

Der Wannenstecker „RS232“ wird 1:1 auf eine 9-polige D-Sub-Buchse aufgelegt. Anschluss 10 hat GND-Potenzial und kann für die Abschirmung der Verbindungsleitung zum Rechner verwendet werden. Die Belegung ist Tabelle B.3 zu entnehmen.

| Anschluss Wannenstecker | Anschluss D-Sub-Buchse |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| 10 | Gehäuse |

**Tabelle B.3:** Anschlussbelegung des Wannensteckers „RS232“

# C Beispielprogramme

An dieser Stelle finden Sie ein paar kurze Beispielprogramme, anhand deren die Programmstruktur verdeutlicht werden soll.

Alle Beispielprogramme sowie weitere Hard- und Softwareprojekte im Zusammenhang mit dem Evaluationboard finden Sie im Internet unter der Adresse
`http://avr.s-jordan.de`

## C.1 Lauflicht

Im folgenden Beispielprogramm handelt es sich um ein Lauflicht, das die Ausgänge PA0...PA7 der Reihe nach ansteuert.

```
//-------------------------------------------------------------------
// Headerdateien
//-------------------------------------------------------------------

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdint.h>
#include <stdio.h>

//-------------------------------------------------------------------
// Globale Variablen
//-------------------------------------------------------------------

volatile unsigned int takt_1s = 0;     // Wird alle 1s von Timer-ISR auf
                                       // 1 gesetzt und in main() bearbeitet
volatile unsigned int schritt = 0;     // Aktueller Programmschritt

//-------------------------------------------------------------------
// Hauptprogramm
//-------------------------------------------------------------------

int main(void)
{

  DDRA=0b11111111;                      // Port A alles Ausgaenge

  TIMSK=(1<<TOIE1)|(1<<OCIE1A)|(1<<OCIE1B);  // Timer 1 Overflow und
                                             // Compare Match Interrupts
                                             // einschalten

  TCCR1B=(1<<CS12);                     // Timer 1 mit Prescaler=256
                                        // von int. Clock (bei 16MHz
                                        // wird Timer 1 alle 16us um 1 erhoeht

  OCR1AH=0b00000010;                    // Timer 1 Compare Match A bei 625
  OCR1AL=0b01110001;                    // (erzeugt Takt von 10ms)

  sei();                                // Globale Interrupts einschalten

  while(1)                              // Beginn der Endlosschleife
  {
    if(takt_1s==1)
    {
      takt_1s=0;

      if(schritt==0)      PORTA=0b00000000;
```

```
      else if(schritt==1) PORTA=0b00000001;
      else if(schritt==2) PORTA=0b00000010;
      else if(schritt==3) PORTA=0b00000100;
      else if(schritt==4) PORTA=0b00001000;
      else if(schritt==5) PORTA=0b00010000;
      else if(schritt==6) PORTA=0b00100000;
      else if(schritt==7) PORTA=0b01000000;
      else if(schritt==8) PORTA=0b10000000;

      schritt++;
      if(schritt>=9) schritt=0;
    }
  }
  return(0);
}

//---------------------------------------------------------------------
// Interrupt-Subroutinen
//---------------------------------------------------------------------



ISR (TIMER1_OVF_vect)                         // Timer 1 Ueberlauf
{

}


ISR (TIMER1_COMPA_vect)                       // Timer 1 Compare Match A
{
  static unsigned long int zaehler=0;
  TCNT1H=0;                                   // Timer 1 zuruecksetzen
  TCNT1L=0;
  zaehler++;                                  // Zaehler alle 10ms erhoehen
  if(zaehler>=100)                            // Zaehler >=100 bedeutet 1s
  {
    zaehler=0;
    takt_1s=1;
  }
}

ISR (TIMER1_COMPB_vect)                       // Timer 1 Compare Match B
{

}
```

# D Häufig gestellte Fragen

**Mein Notebook hat keine RS232-Schnittstelle mehr. Kann ich auch einen USB auf RS232 - Adapter verwenden?**

Der Theorie nach sollte das eigentlich funktionieren. Leider hat sich in verschiedenen Praxistests herausgestellt, dass der Flashvorgang mit einem solchen Adapter erheblich länger dauert als direkt per RS232 (Stunden statt Sekunden!). Wenn jemand einen gut funktionierenden Adapter hat, würde ich mich über eine kurze Mitteilung freuen. Erfolgreich getestet werden konnte jedoch eine PCMCIA-RS232-Karte. Diese ist zwar etwas teurer als die USB-Adapter, verhält sich aber wie eine gewöhnliche RS232-Schnittstelle. Falls Sie für Ihr Notebook eine Dockingstation besitzen, könnte es sein, dass diese einen RS232-Anschluss bietet, obwohl sie beim Notebook selbst fehlt.

**Beim Flashen bzw. Auslesen/Schreiben der Fusebits erhalte ich die Fehlermeldung „Device missing or unknown device“ vom Ponyprog. Woran kann das liegen?**

Für diesen Fehler können eine Reihe von Problemen verantwortlich sein:

- Die Stromversorgung des Evaluationboard fehlt oder ist zu niedrig.
- Das RS232-Verbindungskabel zwischen Evaluationboard und PC ist beschädigt.
- An den I/O-Pins PB5, PB6 und / oder PB7 des Controllers ist eine lange Leitung angeschlossen (diese Pins werden beim Flashen verwendet und die Kapazität ist dann evtl. zu hoch, sodass die Flanken nicht mehr richtig erkannt werden).
- Das Ponyprog muss auf dem Rechner als Administrator ausgeführt werden, weil sonst die RS232-Schnittstelle nicht richtig angesprochen werden kann.
- Die Fusebits wurden falsch gesetzt, sodass der Controller nur noch mit einem externen Taktsignal von einem Frequenzgenerator läuft (zwischen Pin 13 des ATmega32 und Masse des Evaluationboard muss ein Rechtecksignal mit einer Amplitude von 5 Volt und einer Frequenz von 1 MHz angelegt werden). Dann können die Fusebits wieder so gesetzt werden, dass der interne oder externe Oszillator verwendet wird.

**Ist es möglich, beim Ponyprog die Überprüfung nach dem Flashen abzuschalten?**

Ja, das ist möglich. In der Konfigurationsdatei
`C:\Programme\Ponyprog2000\Ponyprog2000.ini`
muss dazu der Parameter `VerifyAfterWrite` auf `NO` gesetzt werden. (Dank an Andreas Trebing für diese Information!) Somit lässt sich während der Entwicklungsphase Zeit einsparen. Nach dem letzten Flashen vor der Auslieferung bzw. dem praktischen Einsatz sollte das Verifizieren jedoch auf jeden Fall noch einmal durchgeführt werden.