

**AVR-8-bit-Mikrocontroller**  
**Arbeiten mit CodeVisionAVR C-Compiler**  
**Teil 7.3.1 AVR-Projekt PB\_LED**

## Teil 01 - Einführung

- 1 Eine Einführung in C
  - 1.1 Warum C ?
  - 1.2 Wie entstand C ?
  - 1.3 Der AVR-Mikrocontroller in einem eingebetteten System

## Teil 02 - Erste Schritte

- 2 Grundausrüstung und Installation der Werkzeuge
  - 2.1 Testboard, Mikrocontroller, Programmer
    - 2.1.1 Testboard und Mikrocontroller
    - 2.1.2 ISP-Programmer
  - 2.2 AVR Studio installieren
  - 2.3 Versteht mich der Mikrocontroller ?
  - 2.4 CodeVisionAVR C-Compiler installieren
  - 2.5 Editieren von Quell-Dateien

## Teil 03 - Aufbau eines C-Projektes

- 3 Was ist ein C-Projekt ?
  - 3.1 Erzeugen eines C-Projektes
    - 3.1.1 Ein neues Projekt beginnen
    - 3.1.2 Ein C-Projekt generieren
  - 3.2 Dateistruktur eines C-Projektes
  - 3.3 Einbindung von AVR Studio in den CVAVR
  - 3.4 AVR Studio Debugger
  - 3.5 C-Compiler-Optionen

## Teil 04 - Der Preprozessor

- 4 Preprozessor-Anweisungen
  - 4.1 Struktur der C-Quell-Programme
  - 4.2 `#include`-Anweisung
  - 4.3 `#define`-Anweisung (Makro)
    - 4.3.1 Makros ohne Parameter
    - 4.3.2 Makros mit Parametern
  - 4.4 `#undef`-Anweisung
  - 4.5 `#if`-, `#ifdef`-, `#ifndef`-, `#else`- und `#endif`-Anweisungen
  - 4.6 Andere Preprozessor-Anweisungen

## Teil 05 - Syntax der C-Programme

- 5 Die Syntax der C-Programme
  - 5.1 C-Quell-Programme
    - 5.1.1 Kommentare
    - 5.1.2 Deklarationen (Vereinbarungen)
    - 5.1.3 Die Funktion `main`
    - 5.1.4 Schlüsselwörter (Keywords) des CodeVisionAVR C-Compilers
  - 5.2 Konstanten und Variablen
    - 5.2.1 Zahlensysteme
    - 5.2.2 Datentypen
    - 5.2.3 Konstanten
    - 5.2.4 Variablen
  - 5.3 Operatoren
    - 5.3.1 Arithmetische Operatoren
    - 5.3.2 Relationale Operatoren
    - 5.3.3 Logische und bitweise wirkende Operatoren
    - 5.3.4 Andere Operatoren und Shortcuts
  - 5.4 Komplexe Objekte in C
    - 5.4.1 Funktionen
    - 5.4.2 Funktions-Prototypen
    - 5.4.3 Pointers und Arrays
      - 5.4.3.1 Pointers
        - 5.4.3.1.1 Pointers in Verbindung mit `flash` und `eeprom`
        - 5.4.3.1.2 Pointers in Verbindung mit `typedef`
      - 5.4.3.2 Arrays

**AVR-8-bit-Mikrocontroller**  
**Arbeiten mit CodeVisionAVR C-Compiler**  
**Teil 7.3.1 AVR-Projekt PB\_LED**

- 5.4.3.2.1 Ein-dimensionale Arrays
- 5.4.3.2.2 Zwei-dimensionale Arrays
- 5.4.3.2.3 Drei-dimensionale Arrays
- 5.4.3.3 Benutzen der Array-Namen als Pointers
- 5.4.3.4 Arrays von Pointers
- 5.4.3.5 Pointers auf Funktionen (Funktionszeiger)
- 5.4.3.6 Funktionen in Verbindung mit `typedef`
- 5.4.4 Strukturen und Unionen
  - 5.4.4.1 Strukturen
  - 5.4.4.2 Unionen
- 5.4.5 Komplexe Typen (eine Zusammenfassung)
- 5.5 Steuerung des Programmablaufs
  - 5.5.1 Anweisungsblöcke { . . . }
  - 5.5.2 Die Anweisung `if`
  - 5.5.3 Die Anweisung `if` in Verbindung mit `else`
  - 5.5.4 Die Fallunterscheidung `switch`
  - 5.5.5 Die Schleife `for`
  - 5.5.6 Die Schleife `while`
  - 5.5.7 Die Schleife `do` in Verbindung mit `while`
- 5.6 Arbeiten mit den Ein-/Ausgabe-Ports

**Teil 06 - Modularer Aufbau der AVR-Projekte**

6 Modularer Aufbau

- 6.1 Das Konzept
- 6.2 Nomenklatur
- 6.3 Die globalen Header-Dateien
  - 6.3.1 Die globale Header-Datei `typedefs.h` (Typ- und Bit-Definitionen)
  - 6.3.2 Die globale Header-Datei `iomx.h` (Definitionen aller Register-Bits)
  - 6.3.3 Die globale Header-Datei `macros.h` (Definitionen von Makros)
  - 6.3.4 Die globale Header-Datei `switches.h` (Definitionen von Schalter-Makros)
- 6.4 Die Module
- 6.5 Anwendung der Module
  - 6.5.1 Das Modul `application` (Anwendungs-Steuerung)
  - 6.5.2 Das Modul `lcd_2wire` (Ausgabe auf LCD-20x4)
  - 6.5.3 Das Modul `num_conversion` (Typ-Konvertierung nach ASCII)
  - 6.5.4 Das Modul `adc_ref-1-1` (ADC mit interner Referenz 1,1 V)
  - 6.5.5 Das Modul `rc5_decoder` (RC5-IR-Fernsteuerung-Dekoder)
  - 6.5.6 Das Modul `rc5_encoder` (RC5-IR-Fernsteuerung-Enkoder)
  - 6.5.7 Das Modul `usart` (USART-Steuerung)
  - 6.5.8 Das Modul `twi_master` (I<sup>2</sup>C- bzw. TWI-Steuerung)
  - 6.5.9 Das Modul `timer0_pwm` (TIMER0-Steuerung)

**Teil 07 - Anhang**

7 Anhang

- 7.1 Begriffe und Definitionen
- 7.2 Eigene Bibliothek
- 7.3 AVR-Projekte (Programmbeschreibungen)
  - 7.3.1 AVR-Projekt PB\_LED**
  - 7.3.2 AVR-Projekt 2\_Draht\_LCD
  - 7.3.3 AVR-Projekt IRDMS

**AVR-8-bit-Mikrocontroller**  
**Arbeiten mit CodeVisionAVR C-Compiler**  
**Teil 7.3.1 AVR-Projekt PB\_LED**

## **Vorbemerkung**

Nichts ist vollkommen - und nichts ist endgültig! So auch nicht dieses Tutorial! Deshalb bitte immer erst nach dem neuesten Datum schauen. Vielleicht gibt es wieder etwas Neues oder eine Fehlerbereinigung oder eine etwas bessere Erklärung. Wer Fehler findet oder Verbesserungen vorzuschlagen hat, bitte melden ([info@alenck.de](mailto:info@alenck.de)).

Immer nach dem Motto: Das Bessere ist Feind des Guten und nichts ist so gut, dass es nicht noch verbessert werden könnte.

Das **Tutorial Teil 07** wurde grundlegend geändert. Da **Teil 07** eine Sammlung von diversen AVR-Projekt-Beschreibungen werden soll und damit nicht bei jeder Ergänzung eine neue Version erscheinen muss, werden die Beschreibungen als eigenständige Teile ausgeführt. Diese Version ist eine Ergänzung zum **Tutorial Teil 07**, in dem ein grundlegendes Projekt ausführlich beschrieben wird.

## **Gravierende Änderungen gegenüber der Vorversion**

1. Neuer Abschnitt **5.6** in der Inhaltsangabe
2. Neues Bild zur Demonstration der Beschaltung des Testboards

# AVR-8-bit-Mikrocontroller

## Arbeiten mit CodeVisionAVR C-Compiler

### Teil 7.3.1 AVR-Projekt PB\_LED

#### 7.3.1 AVR-Projekt PB\_LED

#### ZUR HARDWARE DER LED'S UND TASTER

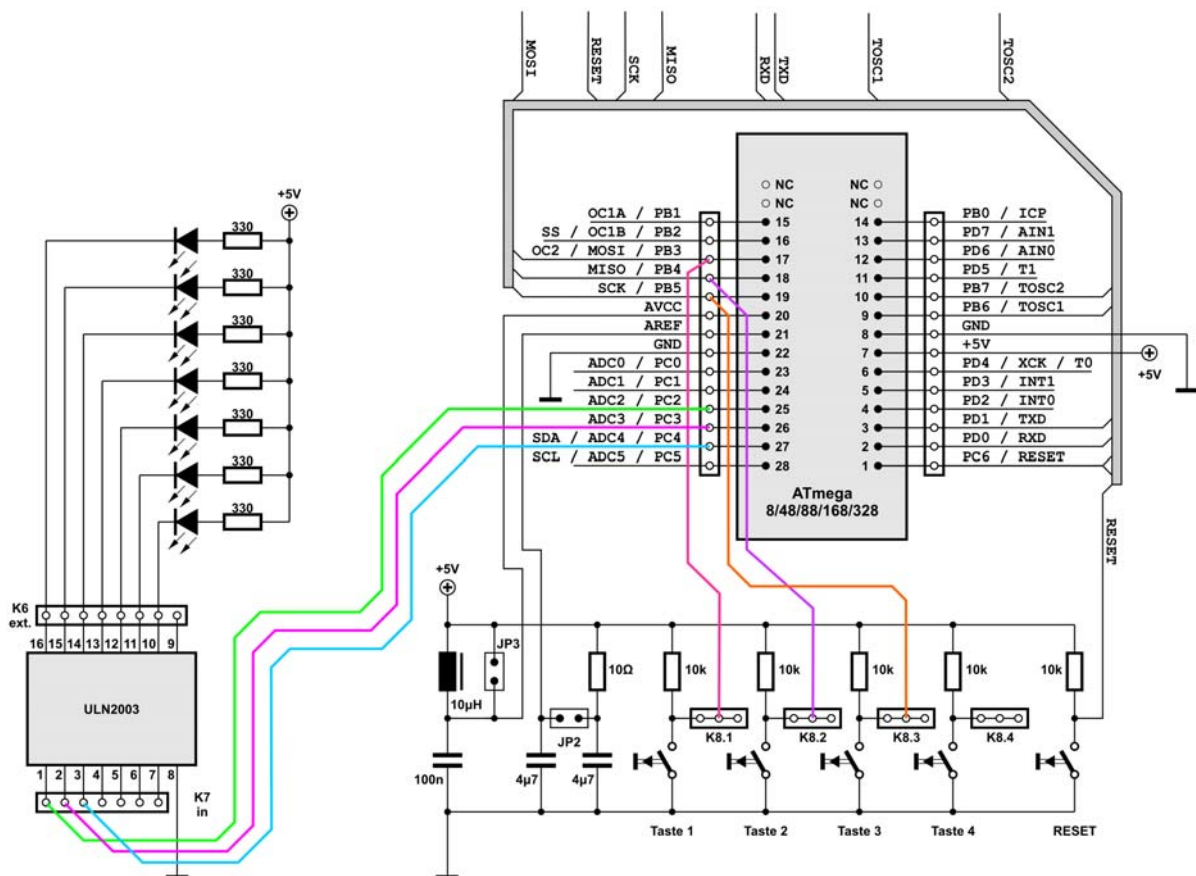
Dieses Projekt stellt eine einfache Applikation zum Gebrauch der LED's und der Taster auf dem Testboard des AVR-Projektes dar. Es ist für viele weitere Projekte geeignet, die eine oder mehrere LED's zur Anzeige bzw. einen oder mehrere Taster benötigen.

Die Taster sind nicht entprellt - können aber universell eingesetzt werden. Sie ziehen beim Betätigen einen angeschlossenen Pin von **High-** auf **Low-**Potential. Pull-Up-Widerstände sorgen für die **High-**Potentiale jedes angeschlossenen Tasters an den Eingangs-Pins des Controllers im Ruhezustand.

Jede LED ist mit ihrer Kathode über einen Vorwiderstand von 330  $\Omega$  an +5 V geschaltet und Ihre Anode ist mit einem Leistungs-Ausgang eines Treiber-Bausteins **ULN2003** verbunden. Die Leistungs-Ausgänge der **ULN2003** sind Open-Collektor-Ausgänge mit einer Belastbarkeit bis zu 500 mA. Damit kann man Klein-Motore, Relais und eben auch kleine LED's schalten:

Ein **High-**Potential an einem Eingang des **ULN2003** schaltet den korrespondierenden Ausgang auf **Low-**Potential und lässt damit eine angeschlossene LED aufleuchten.

#### BESCHALTUNG



**Bild 7.3.1-01: Beschaltung der LEDs und Taster**

Zunächst wird die Beschaltung des Testboards vorgenommen, d.h. die benötigten Pins werden mit dünnen flexiblen Drahtbrücken verbunden:

PC2	mit	LED1	PB3	mit	Taster1
PC3	mit	LED2	PB4	mit	Taster2
PC4	mit	LED3	PB5	mit	Taster3

# AVR-8-bit-Mikrocontroller

## Arbeiten mit CodeVisionAVR C-Compiler

### Teil 7.3.1 AVR-Projekt PB\_LED

#### FUNKTIONSBESCHREIBUNG

- Taster S1 schaltet die LED1 ein, solange der Taster gedrückt ist.
- LED2 ist dauerhaft an und erlischt, solange Taster S2 gedrückt ist.
- Taster S3 schaltet bei jedem Tastendruck die LED3 um ('toggelt' sie).

#### PORT-FUNKTIONEN

Es folgt eine kurze Wiederholung aus dem [Teil 04 Der Preprozessor](#). In diesem Teil wurden die vielen Ersetzungen am Beispiel dieses Projektes anhand von `#define`-Anweisungen beschrieben, so dass man hier vom Ergebnis dieser Ersetzungen ausgehen kann.

#### Die LED's betreffend:

LED1\_DDR, LED2\_DDR und LED3\_DDR als auch LED1\_BIT, LED2\_BIT und LED3\_BIT sind zuständig für das Datenrichtungsregister **DDRC** (Bit2, Bit3 und Bit4).

**Merke:** Ein Datenrichtungsregister (DDR) bestimmt, ob die Pins des betreffenden Ports als Eingänge **oder** als Ausgänge funktionieren sollen. Ein **High**-Potential an den betreffenden Pin des DDR schaltet den korrespondierenden Pin am Port als Ausgang und ein **Low**-Potential an diesem Pin schaltet den korrespondierenden Pin am Port als Eingang. Zum Beispiel:

DDRC = **00001100** bedeutet, dass Bit2 (**PC2**) und Bit3 (**PC3**) vom Port C als Ausgänge geschaltet werden.

Der letzte Schritt des Preprozessors in dieser Angelegenheit ergibt die realen Anweisungen:

```
// Initialisieren der LED's
DDRC |= 0x04;           // Setze LED1-Pin auf Ausgabe
DDRC |= 0x08;           // Setze LED2-Pin auf Ausgabe
DDRC |= 0x10;           // Setze LED3-Pin auf Ausgabe
```

Das sind also bitweise ODER-Verknüpfungen von Bit2, Bit3 und Bit4 im Datenrichtungsregister **DDRC** mit den vorher definierten Konstanten; das Ergebnis wird wieder im Datenrichtungsregister abgespeichert und sorgt so für die Initialisierung dieser Bits für die Ausgabe.

Ergebnis für das Datenrichtungsregister **DDRC** (Ursprungswert von **DDRC** ist nach dem Restart immer binär **00000000**) ergibt sich nach folgender Befehlskette:

```
DDRC = DDRC | 00000100 = 00000100    Bit2 wird als Ausgang konfiguriert
DDRC = DDRC | 00001000 = 00001100    Bit2 und Bit3 werden als Ausgänge konfiguriert
DDRC = DDRC | 00010000 = 00011100    Bit2 (PC2), Bit3 (PC3), Bit4 (PC4) sind Ausgänge !
```

**Anmerkung:** Damit kann über die Bit2 bis Bit4 am **Port C** eine Ausgabe (Schalten der LED's) stattfinden. Eine LED leuchtet erst dann, wenn das betreffende Bit des **Port C** logisch **1 (TRUE)** aufweist (+5 Volt = **High**). Die Zählung der Bits erfolgt von rechts nach links beginnend mit Bit0, Bit1, Bit2 usw.

#### Die Tasten betreffend:

Da die Tasten "Signal-Geber" für den Controller darstellen, werden durch die Ersetzungskette des Preprozessors die Pins des Datenrichtungsregisters **DDRB** auf Null (alle Pins werden als Eingänge konfiguriert) und die Eingabe-Bits Bit3 (PB3) bis Bit5 (PB5) von **PINB (Port B)** auf logisch **1 (TRUE)** gesetzt.

Im Hauptprogramm **main.c** treten die folgenden Initialisierungs-Anweisungen auf:

```
// Initialisieren der Tasten (Pushbuttons)
S1_INIT();
S2_INIT();
S3_INIT();
```

Sie korrespondieren mit den Definitionen in der Header-Datei **switches.h**, mit den Definitionen in der Header-Datei **application.h** und den Ersetzungen von **BIT3**, **BIT4** und **BIT5** in der Header-

## AVR-8-bit-Mikrocontroller

### Arbeiten mit CodeVisionAVR C-Compiler

#### Teil 7.3.1 AVR-Projekt PB\_LED

Datei `typedefs.h` so dass sich das Ergebnis in dieser Anwendung für das Datenrichtungsregister `DDRB` und die Eingabepins von `PINB` schließlich (Ursprungswerte von `DDRB` und `PINB` sind nach dem Restart immer binär `00000000`) nach folgenden Befehlsketten einstellt:

<code>DDRB = DDRB &amp; 11110111 = 00000000</code>	alle Bits sind als Eingänge konfiguriert
<code>DDRB = DDRB &amp; 11101111 = 00000000</code>	alle Bits sind als Eingänge konfiguriert
<code>DDRB = DDRB &amp; 11011111 = 00000000</code>	alle Bits sind als Eingänge konfiguriert
<code>PINB = PINB   00001000 = 00001000</code>	Bit3 ist auf Eingabe aktiviert
<code>PINB = PINB   00010000 = 00011000</code>	Bit3 und Bit4 sind auf Eingabe aktiviert
<code>PINB = PINB   00100000 = 00111000</code>	Bit3 (PB3), Bit4 (PB4), Bit5 (PB5) sind auf Eingabe aktiviert

**Anmerkung:** Das entspricht einer Spannung von jeweils +5 V (**High-Potential**) an den Pins `PB3` bis `PB5`, die erst beim Betätigen einer Taste `s1` bis `s3` auf Masse (**Low-Potential** = 0 V) geschaltet werden.

Der Programm-Quell-Code (C- und Header-Dateien) ist im Abschnitt **AVR\_Programm\_Dateien** unter [0731\\_Programm\\_PB\\_LED](#) zu finden. Weitere Erläuterungen werden im Quell-Programm `main.c` gegeben.