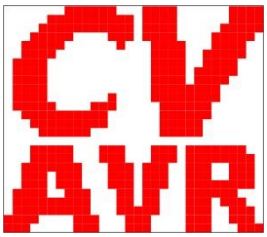
AVR-8-bit-Mikrocontroller
Gruppe 500 - CodeVisionAVR C-Compiler
Teil 502 - Aufbau eines C-Projektes



Teil 501 - Einführung

Teil 502 - Aufbau eines C-Projektes

Teil 503 - Preprozessor

Teil 504 - Syntax der C-Programmiersprache

Teil 505 - Modularer Aufbau der AVR-C-Projekte

Teil 506 - Anhang

AVR-8-bit-Mikrocontroller
Gruppe 500 - CodeVisionAVR C-Compiler
Teil 502 - Aufbau eines C-Projektes

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).

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

Bild-, Beispiel-, Form- und Tabellen-Nummern sind nach folgendem Schema aufgebaut, damit bei Einfügungen/Löschungen nicht alle Nummern wieder geändert werden müssen (hier bunt dargestellt):

Darstellungsart Abschnitt-LfdNummer: Beschreibung	**allgemeines Schema**
• **Bild 5.1.4-02: Daten-Adress-Raum**	Benummerung eines Bildes
• **Beispiel 5.1.4-03: EEPROM-Speicherung**	Benummerung eines Beispiels
• **Form 5.1.3-01: Die `main`-Funktion**	Benummerung einer Formdarstellung
• **Tabelle 5.1.4-01: Schlüsselwörter vom CVAVR**	Benummerung einer Tabelle

Gravierende Änderungen gegenüber der Vorversion

1.
Völlig neue Strukturierung in **Gruppen** und **Teile**, um das Tutorial umfassend ordnen zu können. Die **Abschnitte** in den **Teilen** sind weitgehend erhalten geblieben.

Gruppenbezeichnung	Kurzbezeichnung
Gruppe 100: Technologie der AVR-8-Bit-Mikrocontroller	**Technologie**
Gruppe 200: Einsetzen von AVR-Tools	**Tools**
Gruppe 300: Arbeiten mit AVR-Assembler **3xx_Programm_yyyyy**	**ASM-Programmierung** **ASM-Programm-Beispiel**
Gruppe 400: AVR-ASM-Projekte **4xx_Projekt_yyyyy**	**ASM-Projekte** **ASM-Projekt-Bezeichnung**
Gruppe 500: CodeVisionAVR C-Compiler **5xx_Programm_yyyyy**	**C-Programmierung** **C-Programm-Beispiel**
Gruppe 600: AVR-C-Projekte **6xx_Projekt_yyyyy**	**C-Projekte** **C-Projekt-Bezeichnung**

xx steht für die laufende Nummer innerhalb des **Teils**, in dem das Programm bzw. das Projekt erscheint und **yyyyy** steht für die Programm- bzw. Projekt-Kurz-Bezeichnung.

2.
Notwendige Änderungen auf Grund Neuinstallation von **Windows 7**.

3.
Windows 7 machte eine Installation von **CodeVisionAVR V2.60** als Vollversion notwendig. Daraus leiten sich auch viele Änderungen im Detail für die C-Programmierung (**Gruppe 500**) ab.

4.
Neu-Installation von **AVR Studio Vers. 4.19** unter **Windows 7**

5.
Zur Demonstration des Tools **AVR Studio** ist in **Gruppe 200** eine Trennung in **Teil 205 - Assembler und AVR Studio** und **Teil 206 - C-Compiler und AVR Studio** vorgenommen worden.

6.
ASM- und **C-Projekte** werden jeweils in eigenen Gruppen gesammelt (**Gruppe 400** für Assembler- und **Gruppe 600** für C-Projekte).

Vorbemerkung zu diesem Teil 502

In diesem **Teil 502** wird überwiegend der **Teil 206 - C-Compiler und AVR Studio** eingebettet (d.h. wiederholt), um die **C**-Programmierung mit Hilfe des CodeVisionAVR C-Compilers hier möglichst zusammenfassend darzustellen zu können.

2 Was ist ein C-Projekt ?

Ein **C-Projekt** ist die Summe aller Dateien, die zur Lösung eines Aufgaben-Komplexes (einer Anwendung) mittels **C**-Programmierung notwendig sind.

Es steht hier zunächst **nicht die Lösung** des Problems im Vordergrund und auch **nicht die Aufgabenstellung**, sondern es wird - um den grundsätzlichen Aufbau eines **C**-Projektes darzustellen - von fertigen Quell-Dateien ausgegangen, die bereits fertig erstellt, getestet und erprobt sind. Zur Anschauung wurde das fertige **601_AVR_C_Projekt_PB_LED** (aus der **Gruppe 600**) gewählt. Zur Abkürzung des Namens wird es hier nur mit **AVR_PB_LED** bezeichnet.

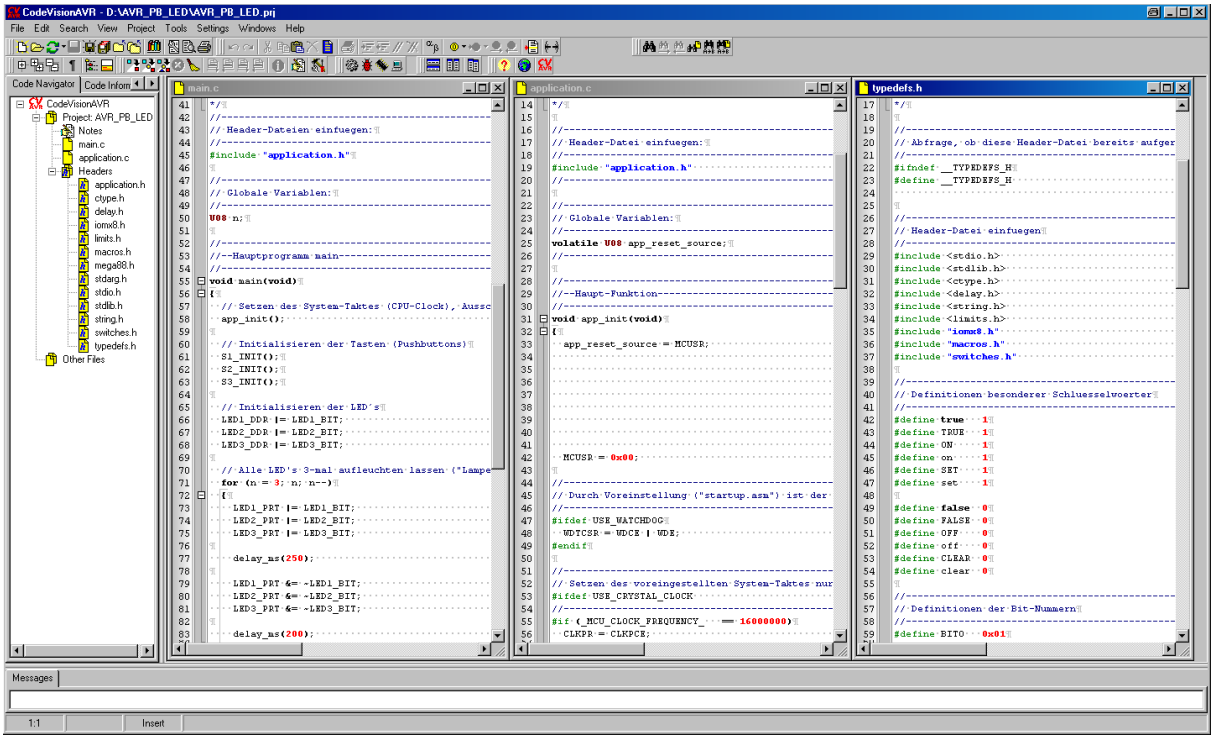


Bild 2-01: Das Projekt AVR_PB_LED (hier: fertig kompiliert)

Im linken Teil des Bildes **Bild 2-01: Das Projekt AVR_PB_LED** ist im **Code Navigator** eine Aufstellung der Quell-Dateien zu sehen. Die Quell-Dateien mit dem Programm-Quell-Code (***.c**-Dateien), d.h. mit den Programm-Befehlen (Anweisungen, Funktions-Aufrufe, Algorithmen usw.), sind:

- `main.c` ist das erste aufgerufene Haupt(**main**)programm mit dem **C**-Programm-Quell-Code, d.h. in der Regel mit vielen Funktionsaufrufen. Im rechten Teil ist der Kode dieses Hauptprogramms aufgerufen und kann dort editiert werden.

- `application.c` das ist die **C**-Quell-Datei mit der Initialisierung der Anwendung, d.h. mit den Anweisungen, die für die Einstellung des Taktes, der Interrupt-Vektoren usw. benötigt werden.

Es folgen die Header-Dateien (***.h-**Dateien). Da sind als erstes die selbst generierten Header-Dateien mit den Beschreibungen der Konstanten, Makros, Datentypen, globalen Variablen und den sog. Funktions-Prototypen zu nennen:

- **`application.h`** Header-Datei zur Definition der im Projekt veränderbaren Parameter
- **`iomx8.h`** Header-Datei zur Definition der Bits des benutzten Mikrocontrollers ATmega88
- **`macros.h`** Header-Datei zur Definition anwendungsspezifischer Makros
- **`switches.h`** Header-Datei zur Definition der Tast-Schalter
- **`typedefs.h`** Header-Datei zur Definition "eigener Datentypen" (Definition äquivalenter Namen zu bestehenden Datentypen)

Und zum guten Schluss noch einige globale Header-Dateien, die ebenfalls in der linken Spalte im **Bild 2-01** auftauchen. Das sind Header-Dateien aus dem "Datei-Fundus", der mit CVAVR ausgeliefert wurde (hierzu mehr im **Teil 503 - Der Preprozessor** und **Teil 504 - Syntax der C-Programme**):

- **`ctype.h`** Header-Datei zur Definition von Funktionen zur Bestimmung von Zeichen-Typen (z.B. alphanumerische, alphabetische, dezimale Zeichen usw.).
- **`delay.h`** Header-Datei zur Definition von Funktionen zur Verzögerung schnell ablaufender Vorgänge.
- **`limits.h`** Header-Datei zur Definition von Minima und Maxima von Datentypen.
- **`mega88.h`** Header-Datei zur Definition von controllerspezifischen Registern, Ports, Timer usw.
- **`stdarg.h`** Header-Datei zur Definition von Makros zur Behandlung variabler Argumenten-Listen.
- **`stdio.h`** Header-Datei zur Definition von Funktionen zur Standard-Eingabe und Standard-Ausgabe.
- **`stdlib.h`** Header-Datei zur Definition von Standard-Bibliotheksfunktionen.
- **`string.h`** Header-Datei zur Definition von Funktionen zur Manipulation von Strings (Zeichenketten).

Wo findet man sie - oder viel wichtiger: wo findet sie der Compiler CVAVR?? Antwort: CVAVR hat sie beim Setup selbst in den von ihm eingerichteten Ordner

```
c:\cvavr2\inc
```

abgelegt und da der Aufruf des Namens mit spitzen Klammern geschieht, z.B. **`<delay.h>`**, schaut der Preprozessor als erstes hier nach:

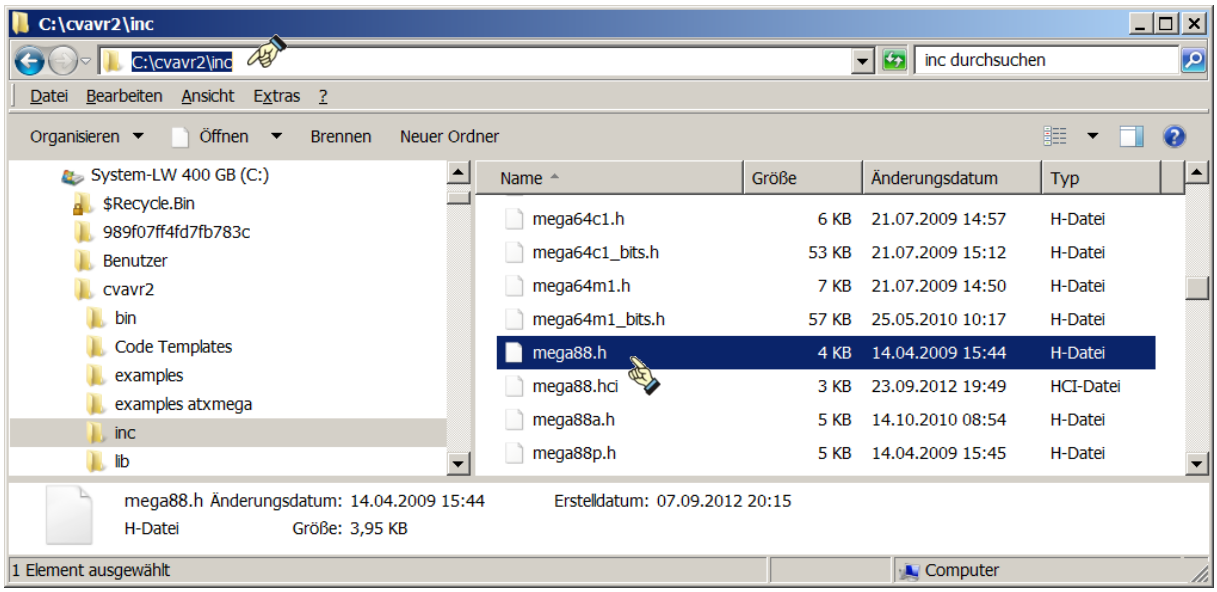


Bild 2-02: Auszug aus dem Datei-Ordner des lokalen Datenträgers (C:)

2.1 Erzeugen eines C-Projektes

Zur Demonstration einer Projekt-Generierung wird ein neuer Projekt-Ordner **`AVR_PB_LED`** angelegt, in dem alle fertigen **`*.c`**- und **`*.h`**-Dateien des fertigen AVR-Projektes **`AVR_PB_LED`** übertragen werden.

Siehe: Teil 601 PB_LED - Einfache Beschaltung von LEDs und Tastern

Der Ordner-Name wurde vom Namen des bestehenden Projektes entlehnt. Dieser Ordner und die folgenden Schritte dienen gleichermaßen zur Demonstration, wie man als "Neuling" fertig erstellte Projekte zum Testen benutzen kann. Wenn man ein neues Projekt beginnt, fängt man ja bei "Adam und Eva" an, d.h. man legt einen neuen Projekt-Ordner an, übernimmt **oder** erstellt darin die Quell-Dateien und marschiert dann durch die hier beschriebenen Instanzen.

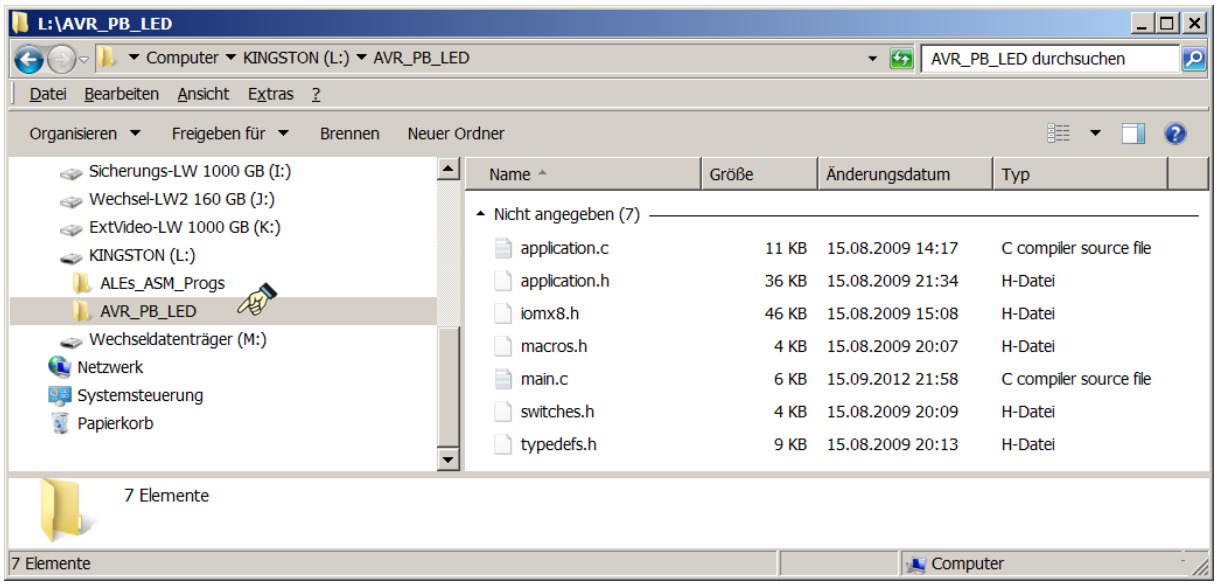


Bild 2.1-01: Einen neuen Projekt-Ordner mit Quell-Dateien anlegen

2.1.1 Ein neues Projekt beginnen

In der **Gruppe 200 - Einsetzen von AVR-Tools** wurde schon das Tool CodeVisionAVR installiert und auch schon ein Hinweis gegeben, wie man mit diesem Tool Quell-Dateien editieren kann. Jetzt wird das erste Projekt (wenn auch nur entlehnt) durch Start dieses Tools erzeugt:

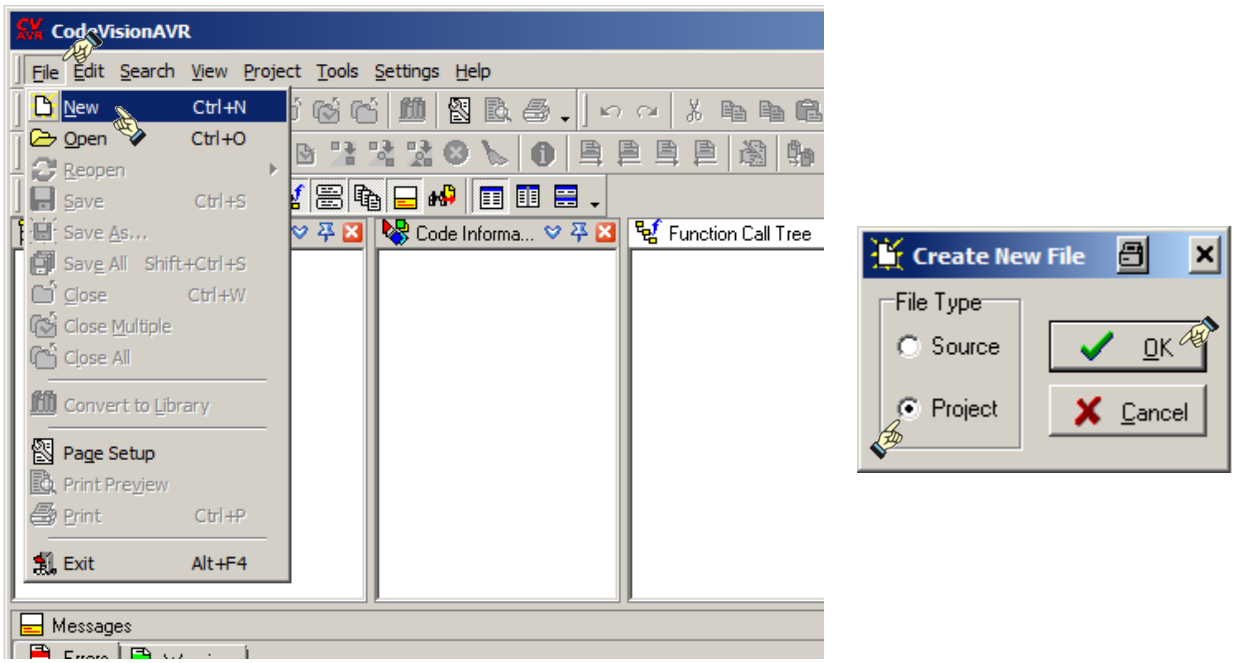


Bild 2.1.1-01: CVAVR für ein neues Projekt starten

File => New => Create New File => Project => OK

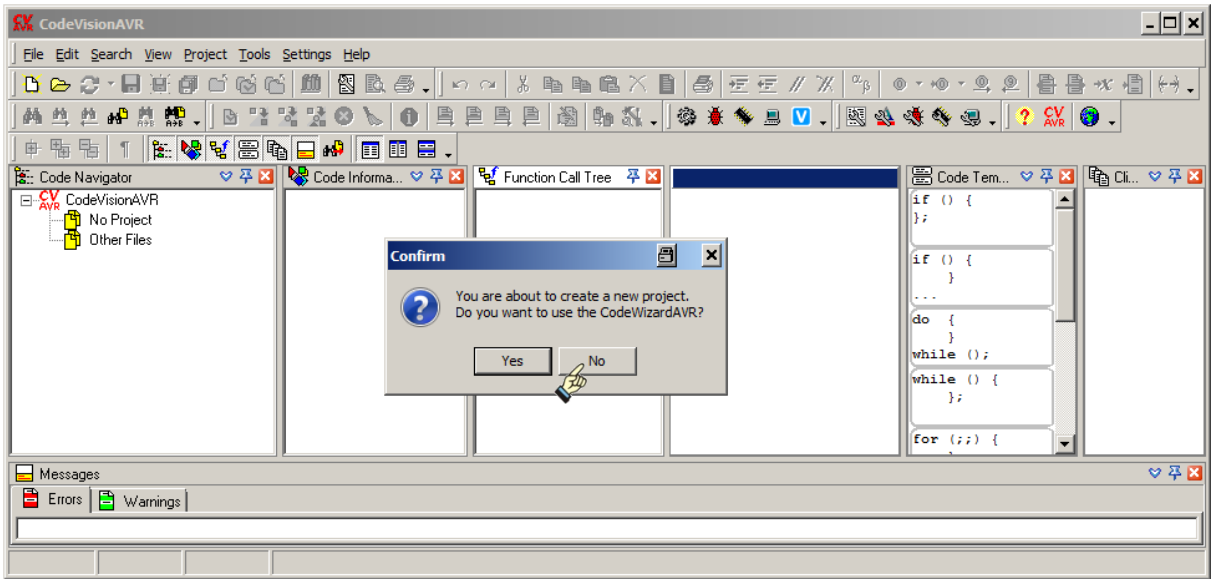


Bild 2.1.1-02: Anfrage, ob der CodeWizardAVR benutzt werden soll

Es soll der **CodeWizardAVR** nicht verwendet werden: **=> No**; es meldet sich das Fenster **Create New Project**. Dieses verweist zunächst auf den Pfad der CVAVR-Installation

```
C:\cvavr2\bin
```

weil der Compiler-Hersteller offenbar davon ausgeht, dass alle Projekte hier konzentriert werden sollen. Da aber das Projekt in einem eigens dafür vorgesehenen Ordner angelegt werden soll, müssen hier neue Einstellungen vorgenommen werden.

Im Feld **Speichern in:** wird der neu eingerichtete Ordner gesucht.

Hier: **`AVR_PB_LED`** **=>** **Speichern**

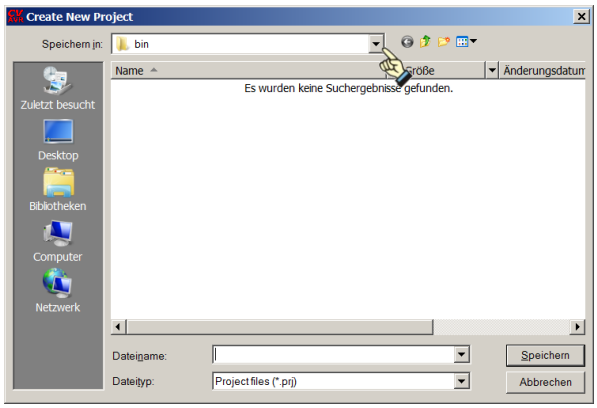


Bild 2.1.1-03: Create New Project

Klick auf ▼ und KINGSTON (L) wählen

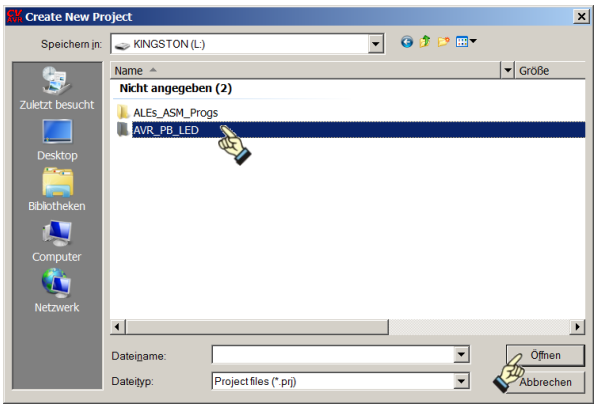


Bild 2.1.1-04: Ordner `AVR_PB_LED` suchen

AVR_PB_LED markieren => Öffnen

Jetzt muss noch der Dateiname für das **Project file (`*.prj`)** benannt werden. Sinnvoll ist es, den gleichen Namen wie für den Projekt-Ordner zu wählen - nämlich **`AVR_PB_LED`**. Nur diese Datei braucht später aufgerufen zu werden, um Änderungen im Projekt vorzunehmen. Sie soll wie folgt abgelegt werden (hier beispielhaft auf dem USB-Stick **`L:`**):

`L:\AVR_PB_LED\AVR_PB_LED.prj`

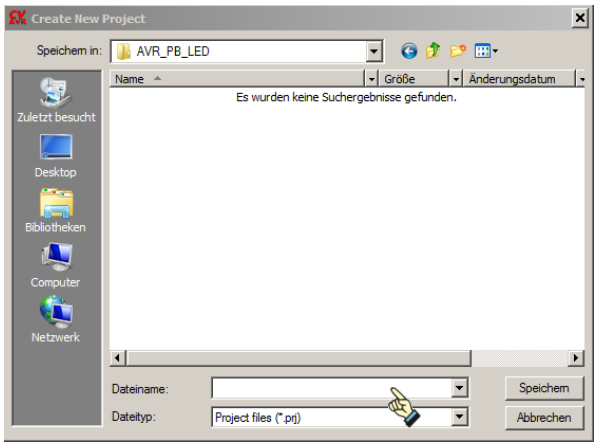


Bild 2.1.1-05: Es wird eine *.prj-Datei angefordert

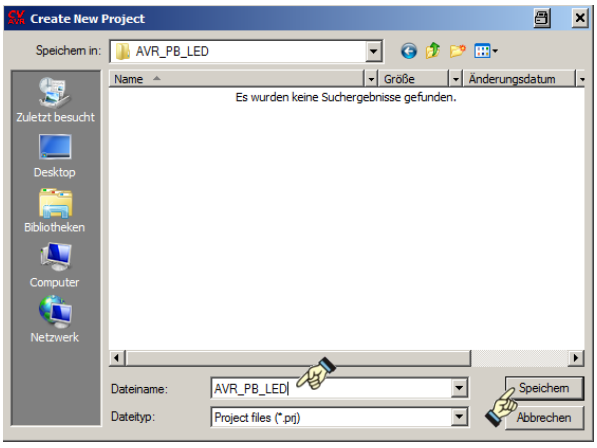
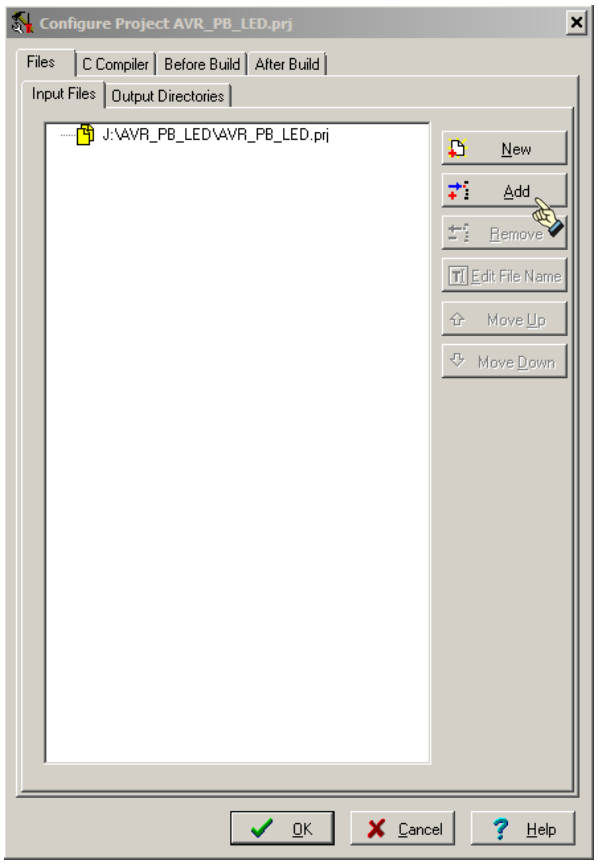


Bild 2.1.1-06: Benennen der *.prj-Datei

Es öffnet sich zusätzlich das Fenster **Configure Project AVR_PB_LED.prj** mit der Aufforderung, dem Projekt nun die Quell-Dateien hinzuzufügen (besser: bekannt zu machen, da sie sich ja bereits im Projekt-Ordner befinden).



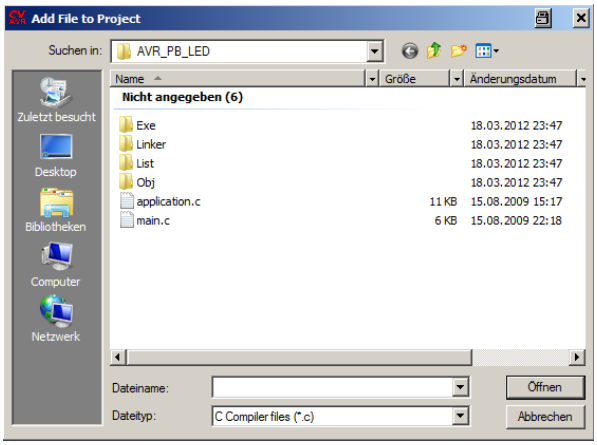


Bild 2.1.1-07: Configure Project Part 1 - Input Files **Add =>** **Bild 2.1.1-08: Neue Projekt-Struktur**

Im Hauptfenster von CVAVR sieht man bereits, dass auf der linken Seite im **Code Navigator** der Projekt-Name **`AVR_PB_LED`** und in der Kopfzeile der Name der Projekt-Datei **`AVR_PB_LED.prj`** eingetragen sind.

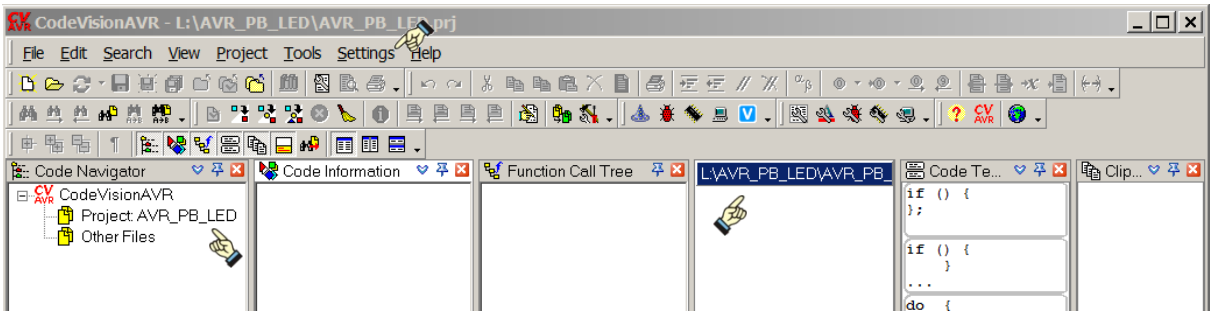


Bild 2.1.1-09: Hauptfenster von CVAVR

Jetzt müssen alle **C**-Dateien zum Kompilieren und Binden dem Projekt bekannt gemacht werden. Das Menü lässt automatisch nur **C**-Dateien zu und weist damit bereits darauf hin, dass nur die **C**-Dateien den anfänglichen Start bestimmen. Die Header-Dateien werden erst während der Kompilierung (genauer: während des Preprozessor-Laufs) aus den **C**-Dateien heraus aufgerufen und in das Coding eingefügt.

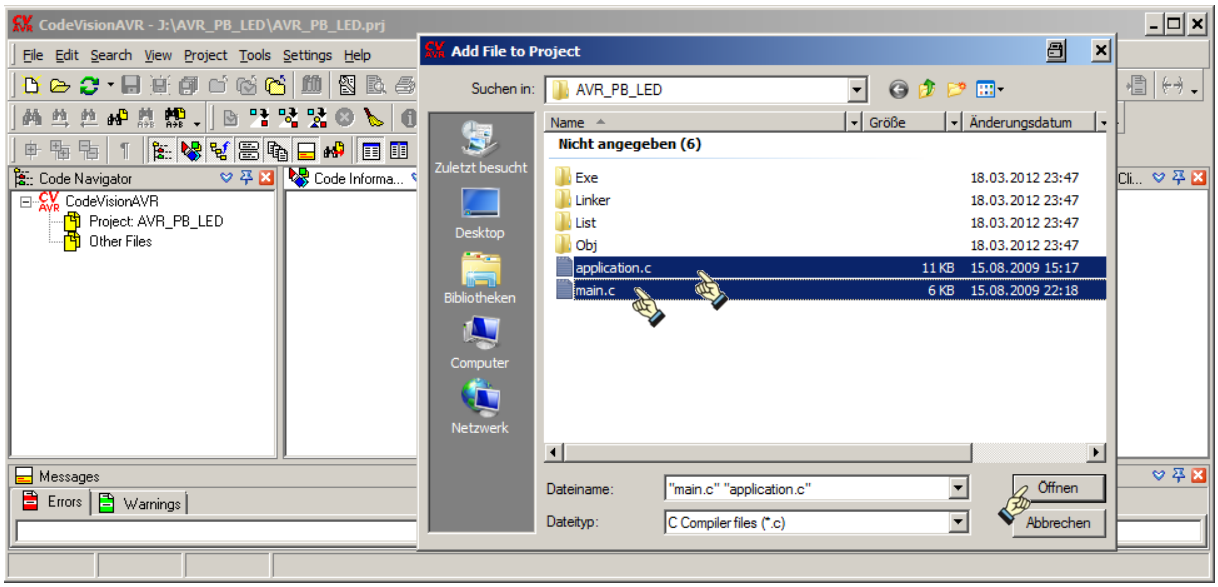


Bild 2.1.1-10: C-Dateien dem Projekt bekannt machen

Alle **C**-Dateien auswählen **=> Öffnen** und . . . . **OK im Configure Project-Menü**

Achtung: Der Vorgang dauert eine Weile - Bitte nicht die Geduld verlieren!

Noch mal der Reihe nach (wegen der Überschneidung der Menüs):

Bild 2.1.1-06: Benennen der *.prj-Datei: AVR_PB_LED.prj => Speichern

Bild 2.1.1-09: Hauptfenster von CVAVR - zur Übersicht

Bild 2.1.1-07: Configure Project - Part 1 (das Menü bleibt zunächst noch sichtbar) **=> Add**

Bild 2.1.1-08: Neue Projekt-Struktur

Bild 2.1.1-10: C-Dateien dem Projekt bekannt machen => *.c-Dateien markieren => Öffnen

Bild 2.1.1-07: Configure Project => OK

Durch **OK** im **Configure Project** werden alle ***.c**-Dateien in das Projekt übernommen.

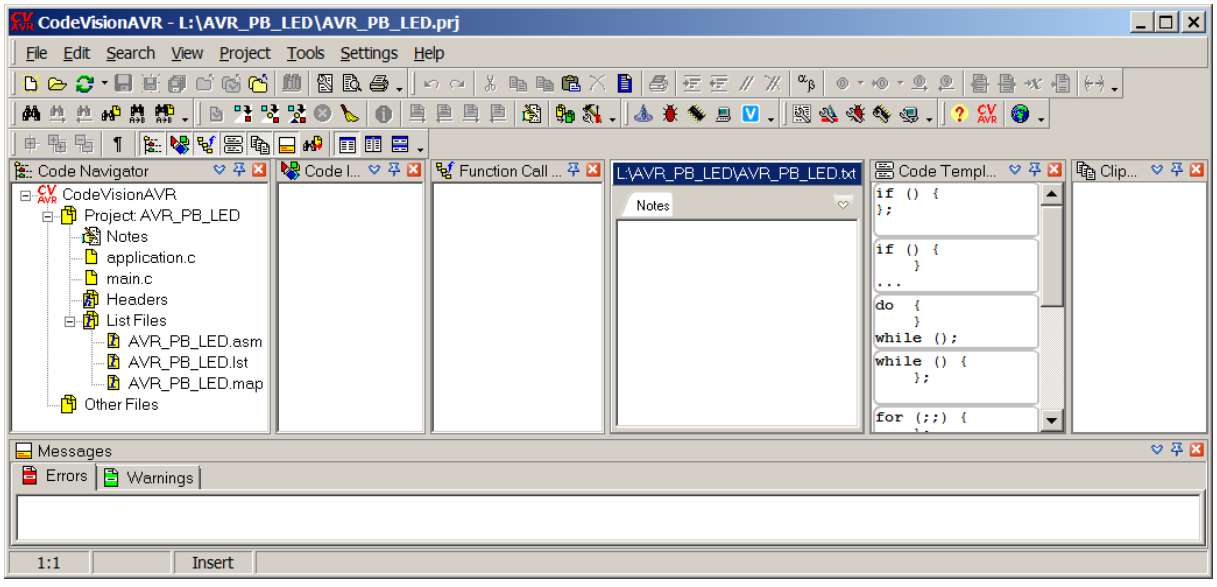


Bild 2.1.1-11: Hauptfenster von CVAVR

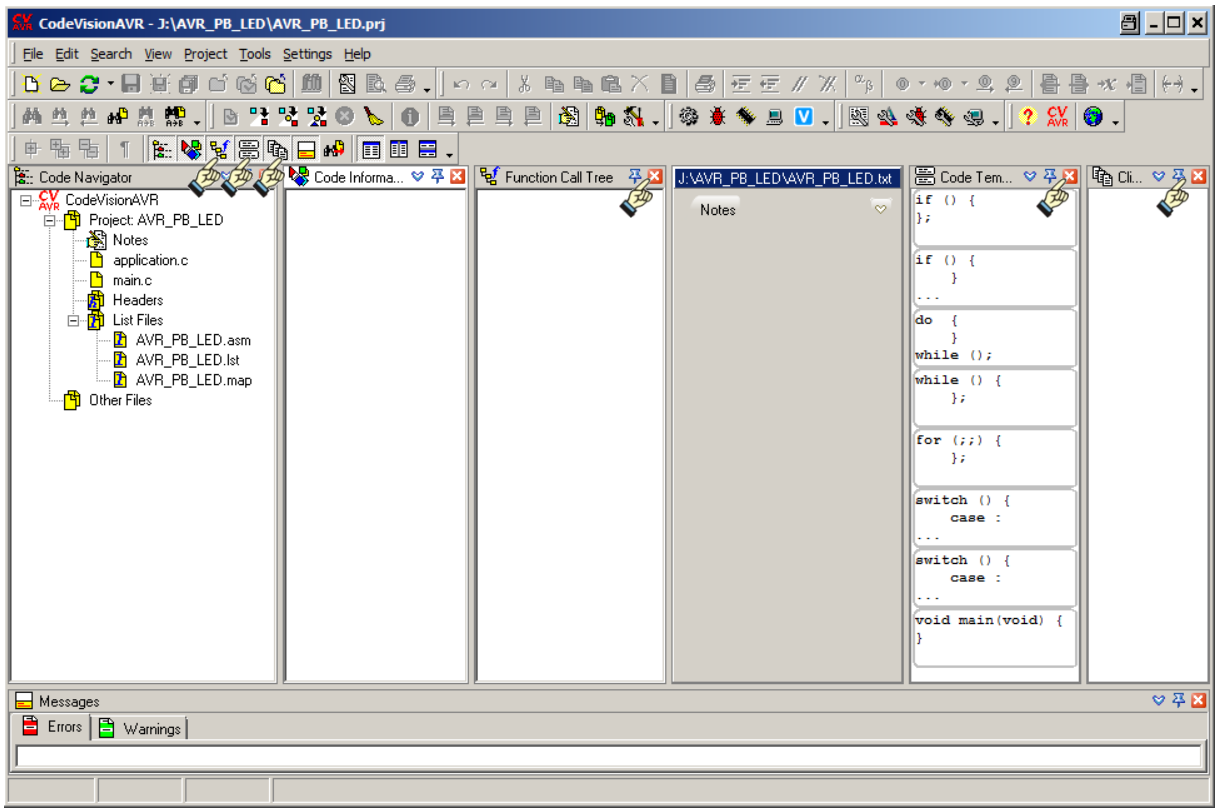


Bild 2.1.1-12: Die Spalten "Function Call Tree", "Code Templates" und "Clipboard History" werden zunächst entfernt, um Platz für die `*.c`-Dateien zu erhalten

In der "File Pane" (Editier-Feld, das ist die 4. Spalte von links) - wo bereits die leere `*.txt`-Datei unter "Notes" vorgemerkt ist - können alle `*.c`-Dateien auch nebeneinander oder untereinander angezeigt werden. In der `AVR_PB_LED.txt`-Datei kann das Projekt unter "Notes" detailliert beschrieben werden - das soll uns hier aber noch nicht interessieren.

Wenn die Felder nicht gleich so erscheinen, wie sie im **Bild 2.1.1-12** dargestellt sind, so ist zu bemerken, dass die Felder - wie in anderen Anwendungen auch - minimiert, maximiert oder "gezogen" werden können, was hier geschehen ist.

Um Platz für die Quelldateien zu schaffen, werden jetzt die Spalten **Function Call Tree, Code Templates** und **Clipboard History** zunächst entfernt (siehe **Bild 2.1.1-12**) und die "File Pane" verbreitert:

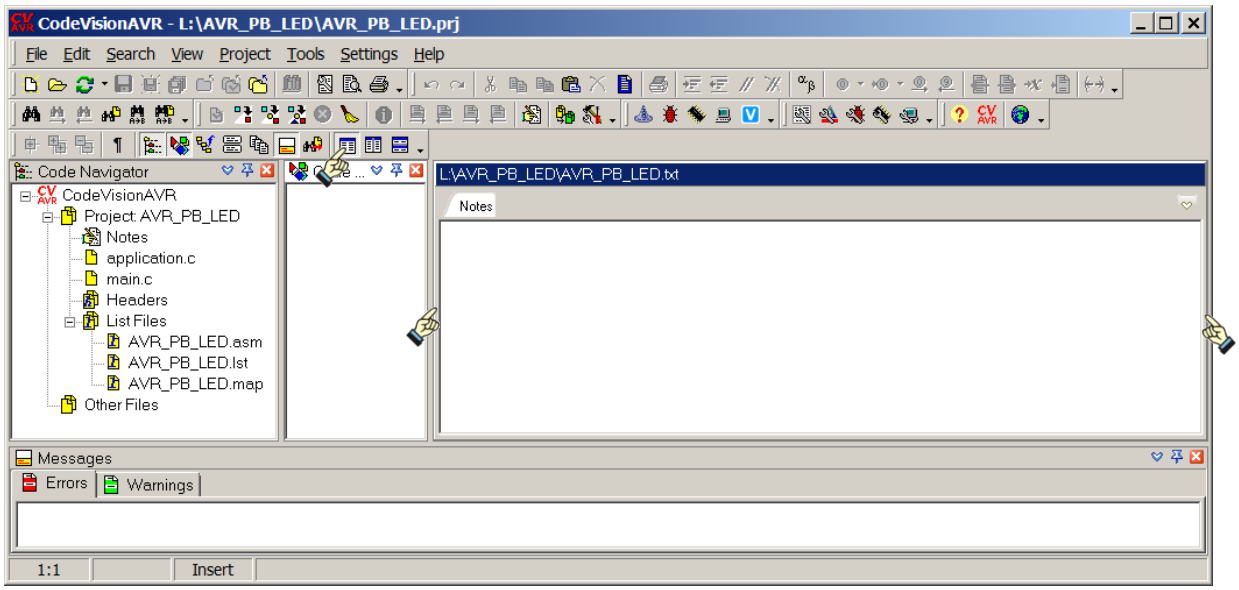


Bild 2.1.1-13: Verbreiterte "File Pane"

Mit ein wenig Jonglieren werden die Spalten gestaltet:

1. Spalten nebeneinander
2. Öffnen von **`main.c`** im **Code Navigator**
3. Öffnen von **`application.c`** im **Code Navigator**

Wie man sieht, kann man im großen Fenster noch eifrig alle Quell-Dateien verändern/editieren. Als erstes wird die Gelegenheit genutzt, dem Projekt eine Kurzbeschreibung in den **Project Notes** voranzustellen. Es wird einfach ein Text in das dafür vorgesehene Datei-Feld eingetragen:

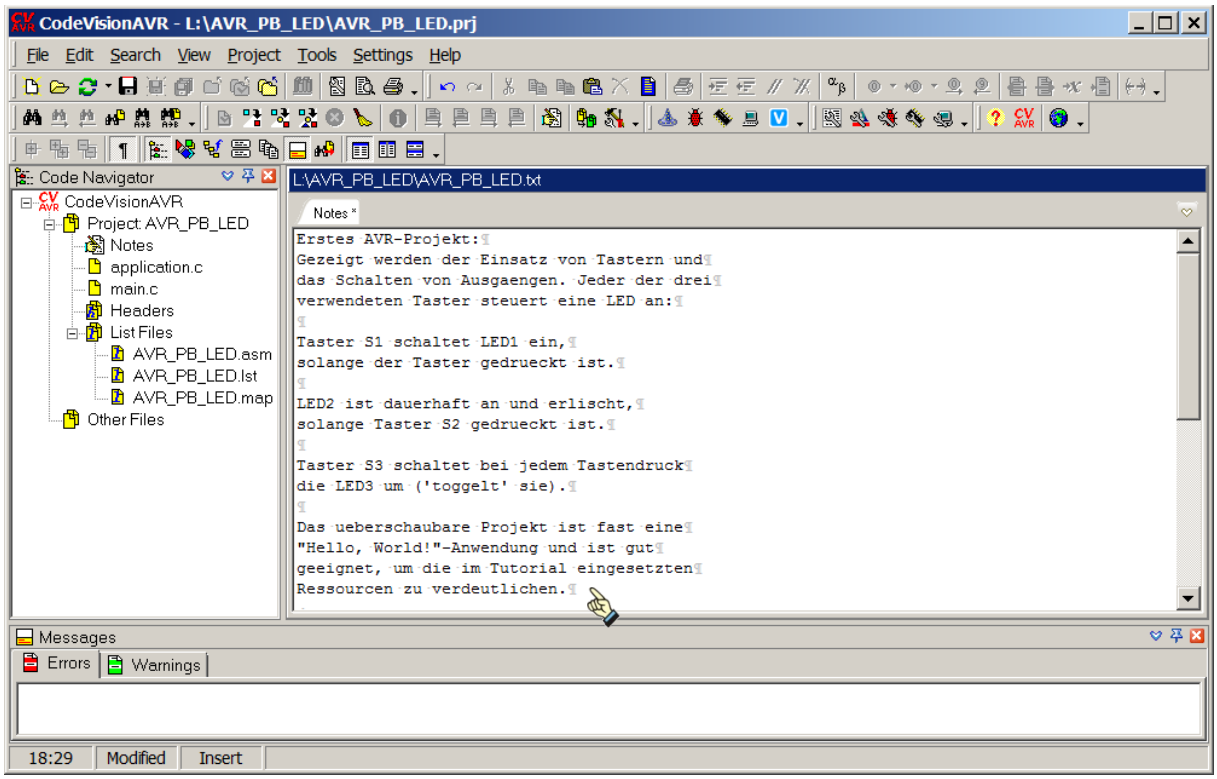


Bild 2.1.1-14: Kurzbeschreibung des Projektes

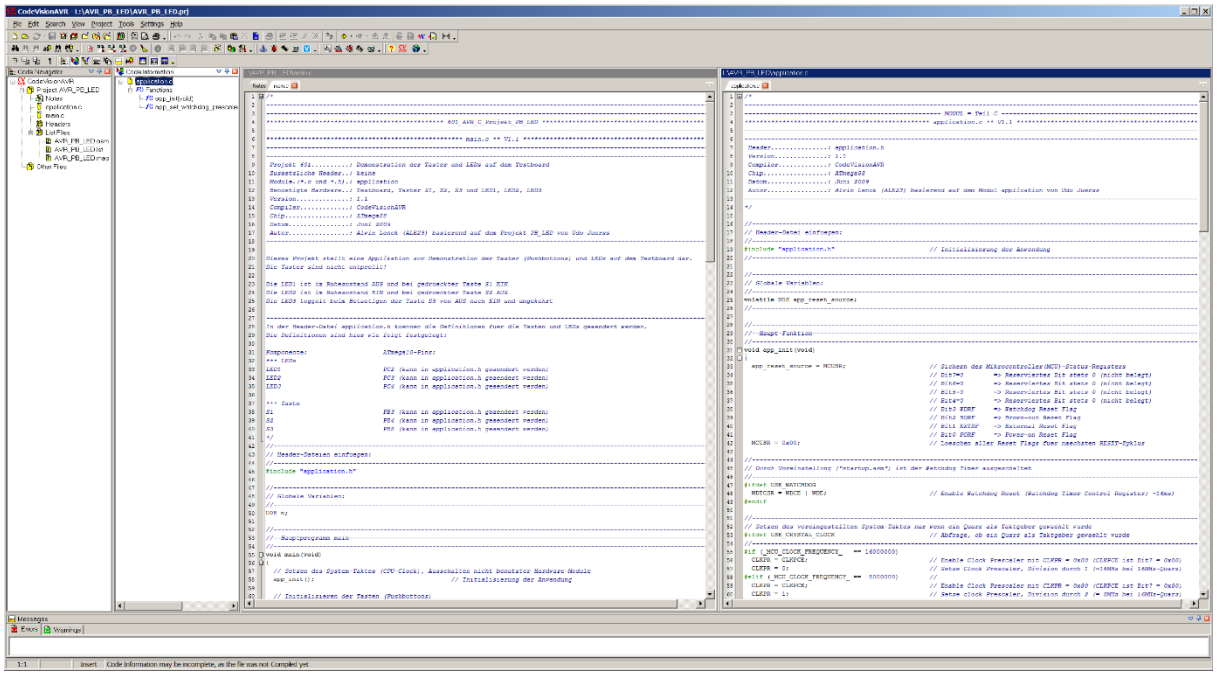


Bild 2.1.1-15: Die `*.c`-Dateien sind bereit zum Editieren (Bildvergrößerung)

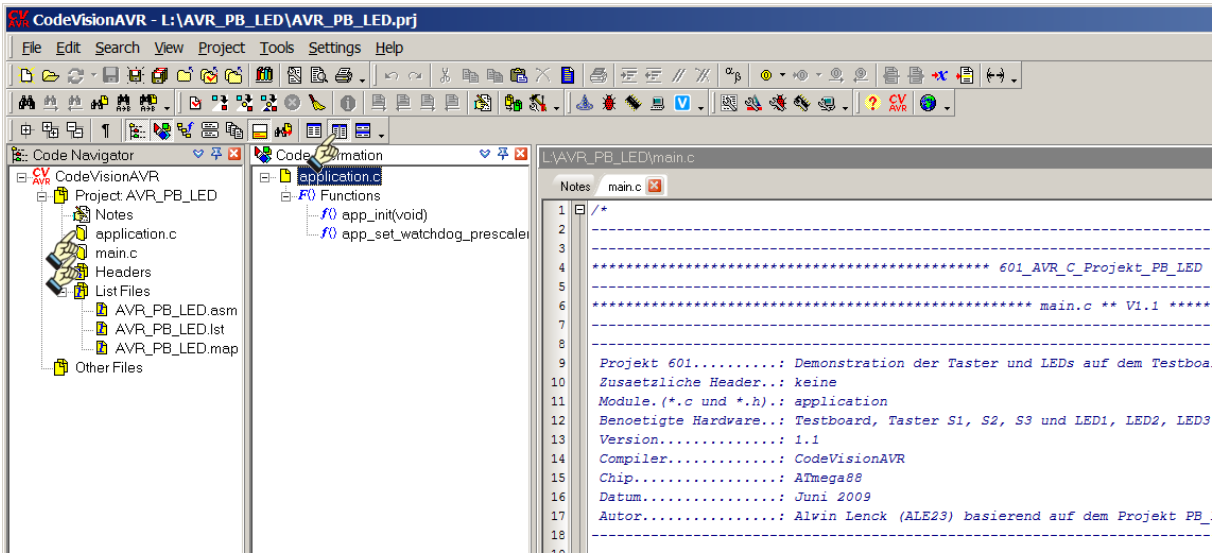


Bild 2.1.1-16: Vergrößerung von Bild 2.1.1-15 - Projekt-Stand vor der Kompilierung

Bevor das Programm kompiliert wird, sollen vorher noch einige andere Listen zur Konfiguration des Projektes genauer betrachtet werden. Dazu wird erst einmal das Konfigurations-Menü erneut aufgerufen:

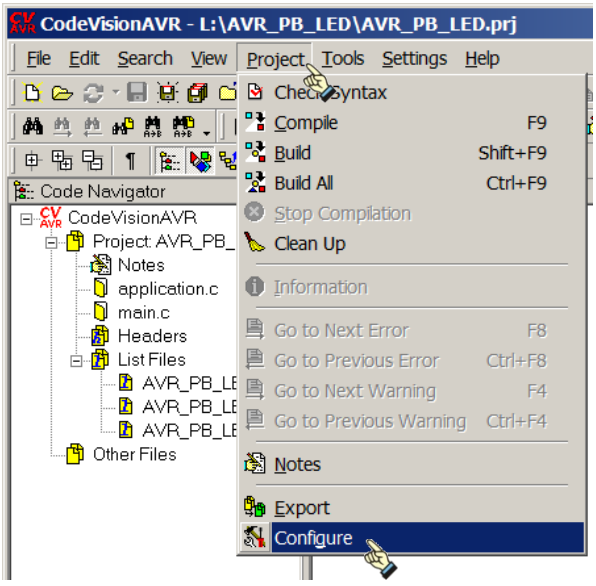


Bild 2.1.1-17: Aufruf des Konfigurations-Menüs

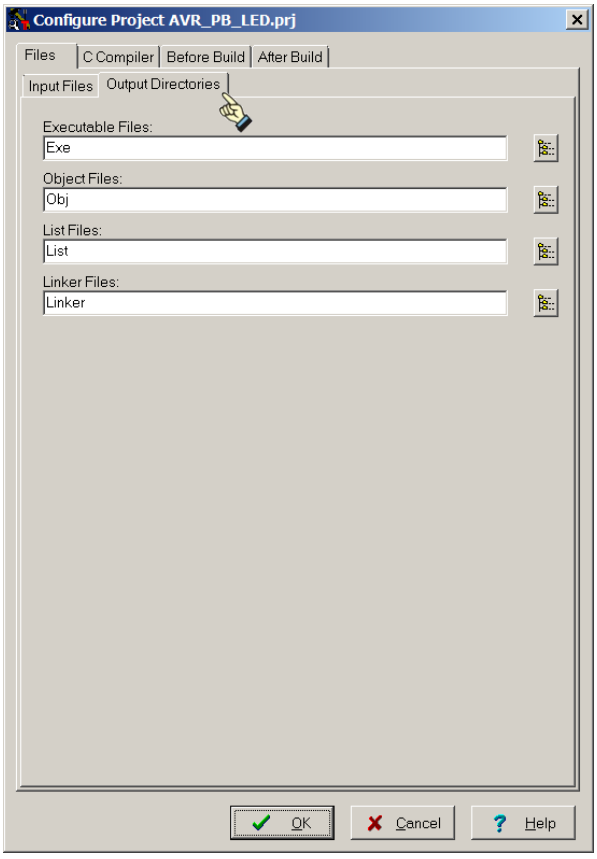


Bild 2.1.1-18: Configure Project Part 2 - Output Directories

Project => Configure =>  **=> Output Directories**

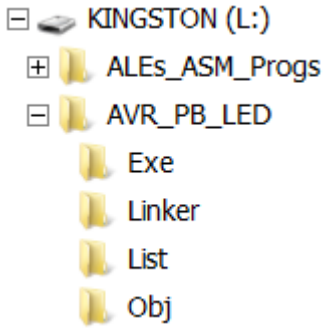
In dieser Abbildung werden die Ausgabe-Ordner angezeigt, in denen bei der Kompilierung die Ergebnis-Dateien abgelegt werden.

Executable Files:
`*.hex`-Dateien vom Linker

Object Files:
`*.obj`-Dateien nach dem Lauf des Assemblers

List Files:
`*.asm`-Dateien nach der Kompilierung

Linker Files:
`*.a`- und `*.o`-Dateien zum Linken/Binden



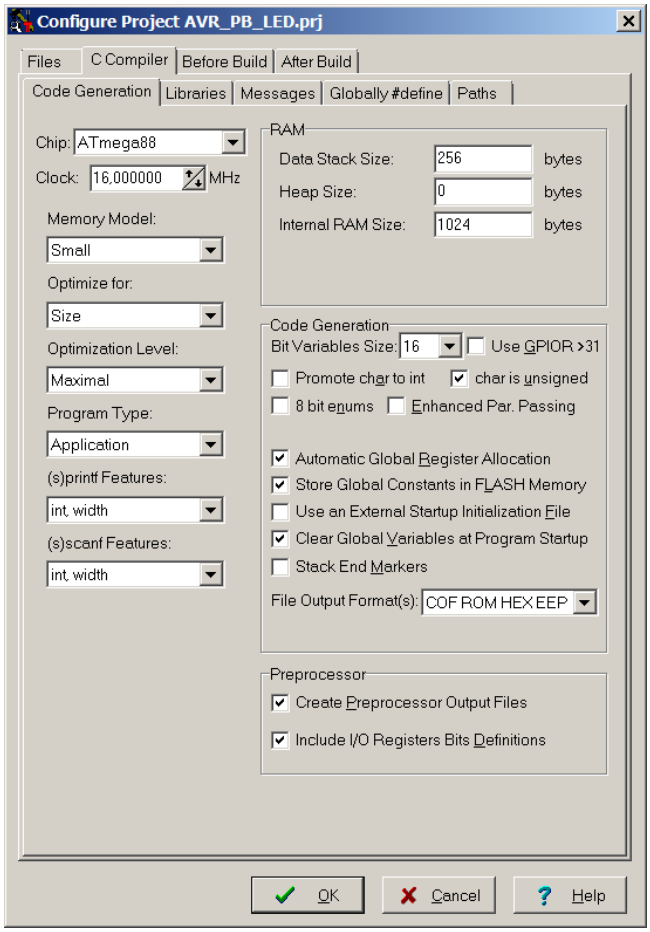


Bild 2.1.1.-19: Configure Project Part 3 - Code Generation

Hier können Einstellungen für den Chip und den Compiler zur Generierung der Ausgabe-Dateien vorgenommen werden. Die Einstellungen werden wie angegeben vorgenommen. Besonders zu beachten sind:

Chip: ATMega88
Clock: 16 MHz
Program Type: Application
Data Stack Size: 256 bytes
Heap Size: 0 Bytes
Internal RAM Size: 1024 bytes
Bit Variable Size: 16
Use GPIOR>31 nicht gesetzt

File Output Format(s):
Formate der Ausgabe-Dateien (hier ist vorrangig das Format **HEX** für ein ausführbares Programm von Interesse).

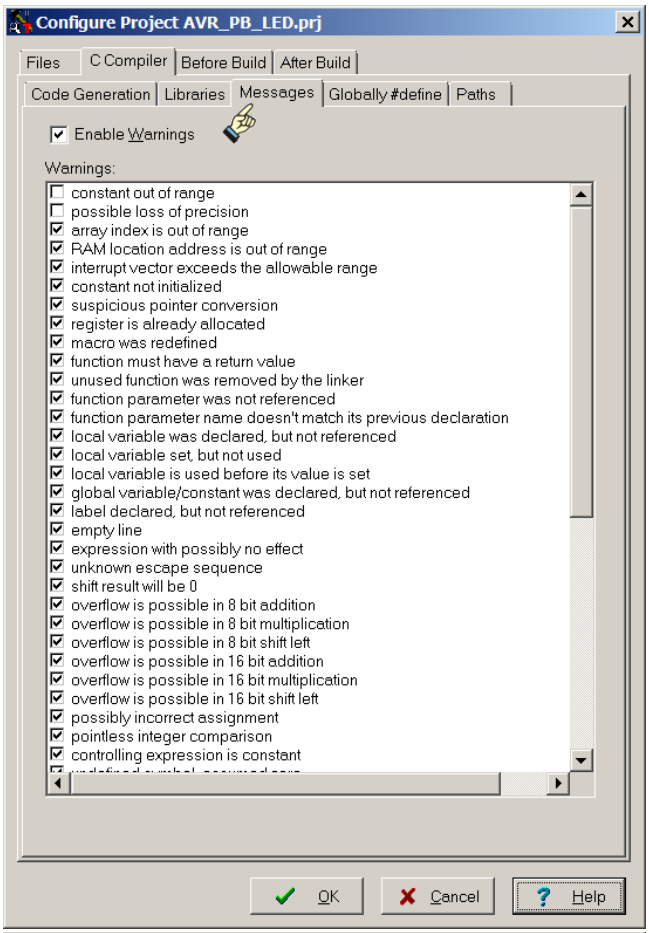


Bild 2.1.1.-20: Configure Project Part 4 - Messages

Das ist die Liste **Messages**, in der man angeben kann, welche Hinweise und Warnungen man erhalten möchte.

Das Setzen bestimmter Warnungen kann beim Testen sehr hilfreich sein, um unerreichbaren Code anzuzeigen oder davor zu warnen, dass Variable deklariert sind, die gar nicht benutzt werden.

Bild 2.1.1.-21: Configure Project Part 5 - Paths

Hier können neue Pfade für zusätzliche `#include`- und library-Dateien angelegt werden:

#include paths (one per line):
für globale Header-Dateien

Library paths (one per line):
für Bibliotheksdateien

Für die hier generierten AVR-Projekte wird von dieser Möglichkeit **kein Gebrauch** gemacht.

Mit **OK** wird das Menü verlassen.

2.1.2 Ein C-Projekt generieren

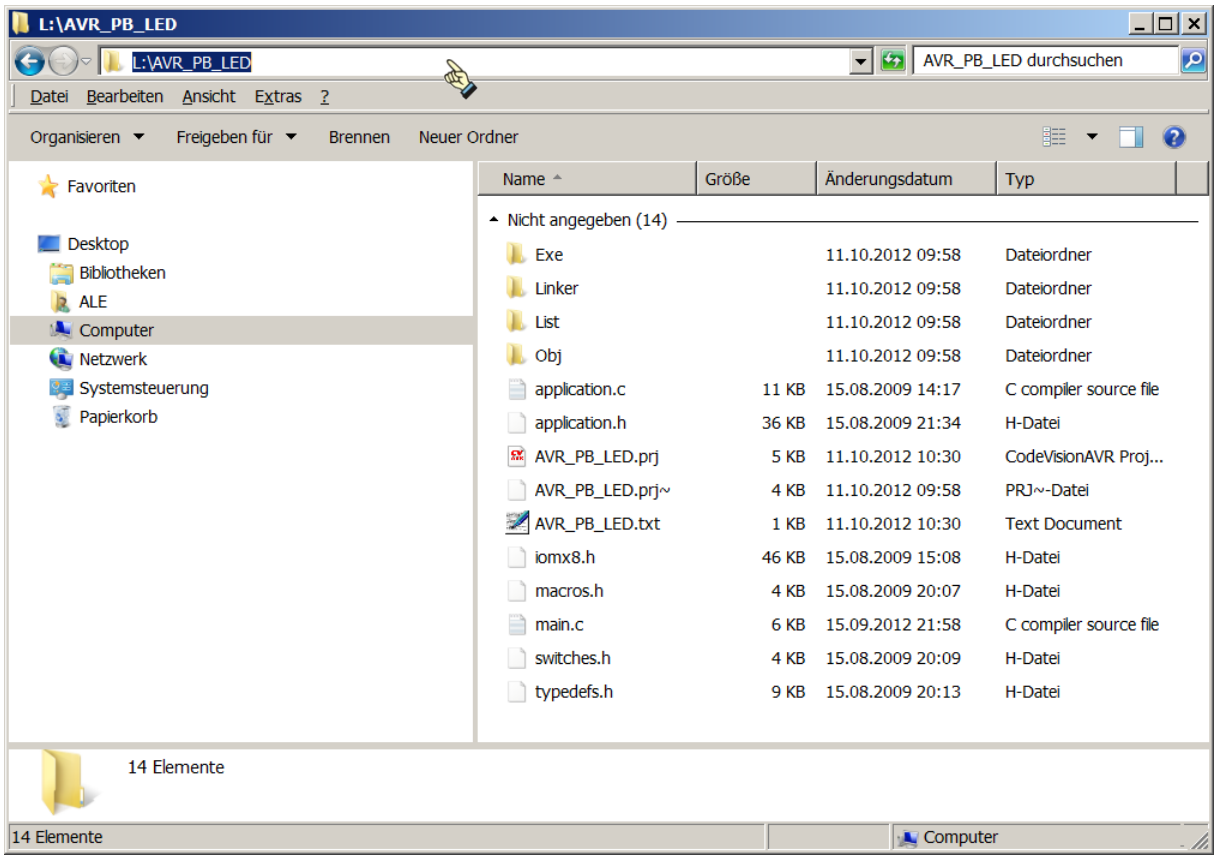


Bild 2.1.2-01: Ausgangslage zur Kompilierung - Stand der Dateien

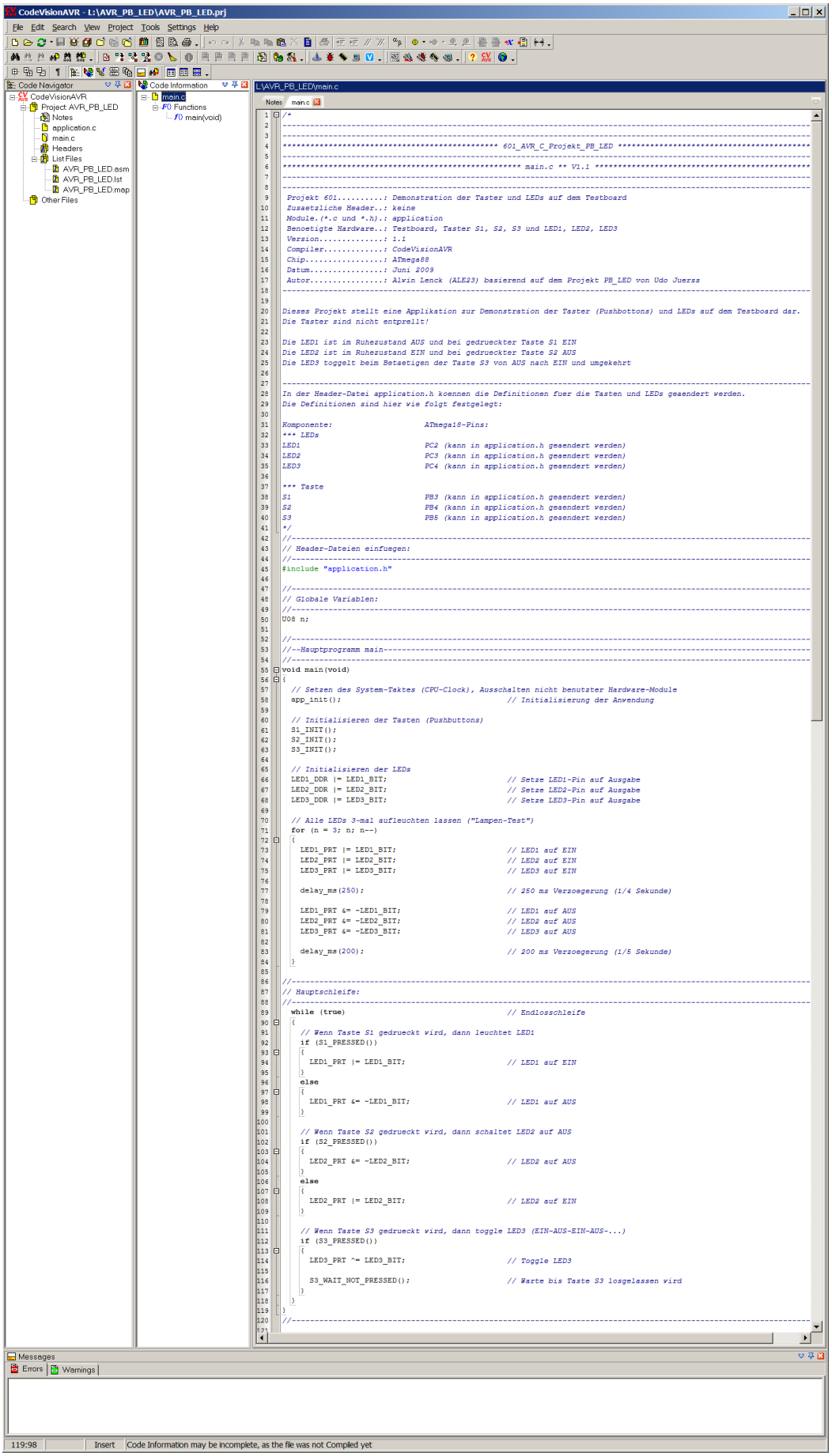


Bild 2.1.2-02: Ausgangslage zur Kompilierung - Hauptprogramm (Bildvergrößerung)

AVR-8-bit-Mikrocontroller
Gruppe 500 - CodeVisionAVR C-Compiler
Teil 502 - Aufbau eines C-Projektes

Die Ausgangslage für die Kompilierung sind in einem **C**-Projekt die sog. Quell-Dateien. In diesem Projekt sind das:

- Das Hauptprogramm **`main.c`**
- Das Modul **`application.c`** incl. **`application.h`**
- Die Header-Datei **`typedefs.h`**
- Die Header-Datei **`iomx8.h`**
- Die Header-Datei **`macros.h`**
- Die Header-Datei **`switches.h`**

Aus ihnen entsteht in grundsätzlich 4 Arbeitsschritten das fertige "Kompilat", das heißt die ausführbare **`*.hex`**-Datei:

1. Arbeitsschritt "Lauf des Preprozessors": Auswerten aller Anweisungen, die mit **#** beginnen. Das sind Anweisungen, die ausschließlich für den Preprozessor bestimmt sind (u. a. Einfügen von Header-Dateien, die mit **`#include "file"`** in den Quell-Dateien aufgeführt sind.

2. Arbeitsschritt "Lauf des C-Compilers" zur Übersetzung der in der Programmiersprache **C** geschriebenen Befehle in Assembler-Anweisungen **=>** **`*.asm`**-Dateien (dieses sind ebenfalls editierbare Text-Dateien).

3. Arbeitsschritt "Lauf des Assemblers" zur Übersetzung der Assembler-Anweisungen in Maschinen-Befehle (Module im Objekt-Code) **=>** **`*.a`**-Dateien und **`*.o`**-Dateien

4. Arbeitsschritt "Lauf des Linkers (Binders)" zur Verbindung aller Objekt-Module zu einem lauffähigen ausführbaren Computer-Programm **=>** **`*.hex`**-Datei

In **CVAVR** kann das **C**-Projekt sequentiell, d.h. in Einzel- bzw. Stufenschritten, generiert werden:

Project => Compile (F9) — **Dieser Schritt** produziert die Objekt-Dateien für den Linker. Die Kompilierung wird nur für die nach dem letzten Durchlauf veränderten Programm-Module durchgeführt.

Project => Build (Shift+F9) — **Dieser Schritt** produziert ein neues Assembler-Quell-Programm mit der Erweiterung **`*.asm`** wobei nur die veränderten Quell-Dateien neu kompiliert werden. Wenn keine Fehler erkannt wurden, wird automatisch der **Atmel-AVR-Assembler** aufgerufen, der die eben erzeugte **`.asm`**-Datei assembliert. Das Ausgabe-File ist eine ausführbare Programm-Datei mit dem Format **`*.hex`**.

Project => Build All (Ctrl+F9) — **Vollständige Kompilierung/Assemblierung/Verlinkung** aller Quellen; Zusammenfassung aller Arbeitsschritte zu einem Arbeitsgang.

Da das Projekt bereits an anderer Stelle getestet wurde, sind keine Fehler (mehr) zu erwarten, so dass sofort mit dem Schritt **Build All (Ctr+F9)** fortgesetzt wird:

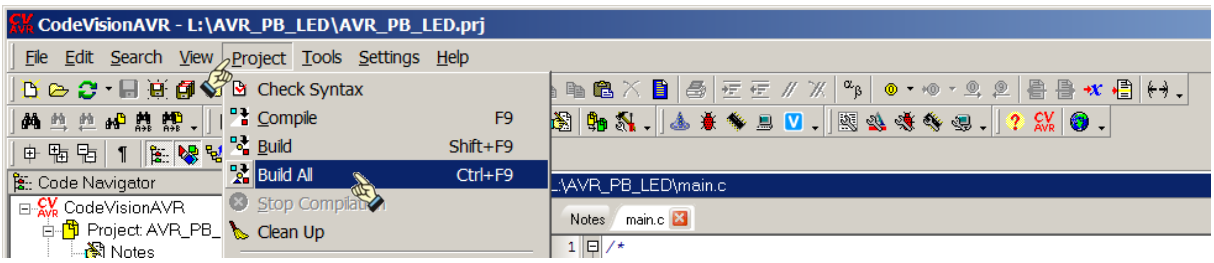


Bild 2.1.2-03: Project => Build All (Ctrl+F9)

Wenn die vollständige Kompilierung fehlerfrei (aber wahrscheinlicht nicht kommentarlos !) gelaufen ist, werden über dem Haupt-Fenster von CVAVR zwei Informations-Listen am Ende der Projekt-Generierung offeriert. Sie zeigen die Ergebnisse für die Kompilierung und die Assemblierung.

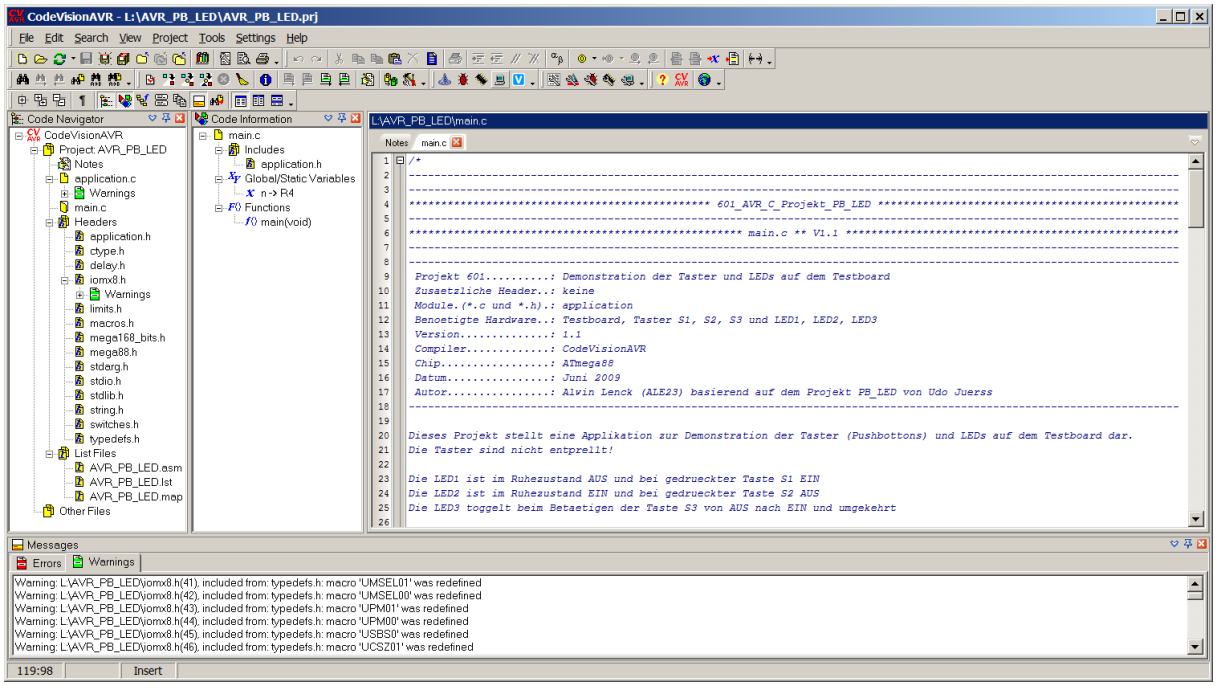


Bild 2.1.2-04: Nach der Kompilierung mit Build All (Ctrl+F9) (Bildvergrößerung)

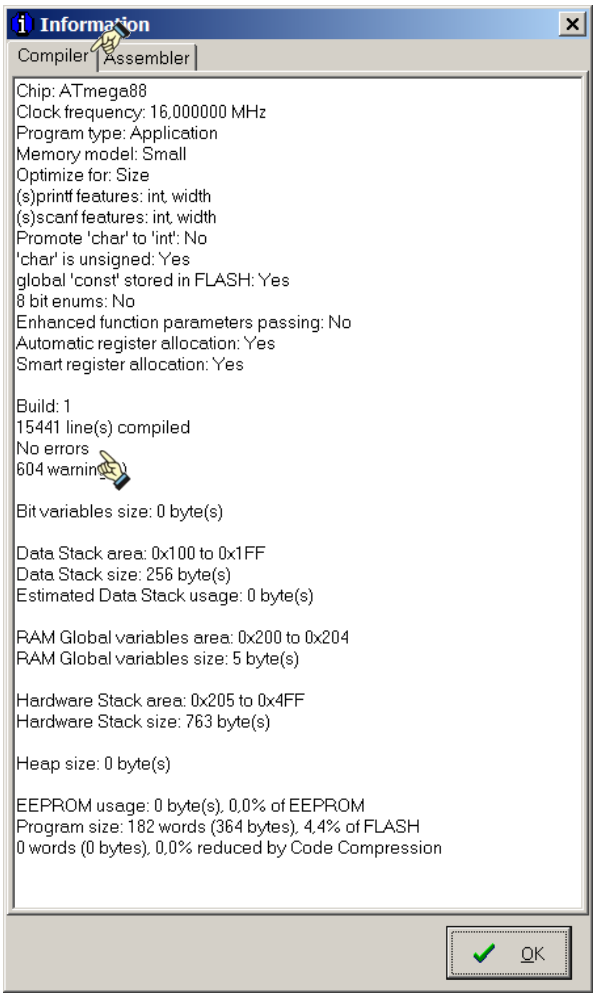


Bild 2.1.2-05: Compiler-Meldungen

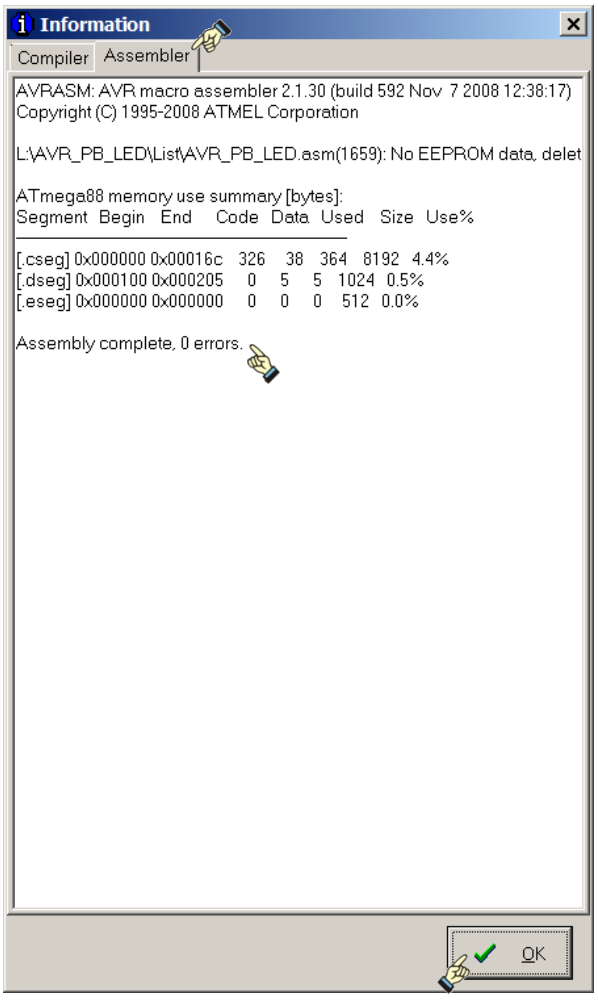


Bild 2.1.2-06: Assembler-Meldungen

Wenn man tiefer in die Materie eingestiegen ist, so geben diese Listen sehr aufschlussreiche Hinweise zum Kompilat und den benutzten Ressourcen (Speicher-Belegungen). An dieser Stelle mögen die Hinweise

No errors und **Assembly complete, 0 errors.**

genügen. Sie zeigen an, dass das Projekt fehlerfrei kompiliert und assembliert wurde und ein ablauffähiges Objekt-Programm erzeugt wurde.

2.2 Dateistruktur eines C-Projektes

Das folgende Bild stellt die ineinander greifende Struktur des oben beispielhaft gewählten **C**-Projektes dar. Die Verschachtelung ergibt sich aus den Quell-Dateien. Weitere Informationen hierzu folgen in **Teil 503 - Der Preprozessor**, **Teil 504 - Syntax der C-Programmiersprache** und **Teil 505 - Modularer Aufbau der AVR-Projekte**.

C-Dateien:
`main.c`, `application.c`

Header-Dateien:
`application.h`, `typedefs.h`, `iomx8.h`, `macros.h`, `switches.h`

Globale Header-Dateien vom CVAVR:
`mega88.h`, `delay.h`, `stdio.h`, `starg.h`, `stdlib.h`, `string.h`, `ctype.h`, `limits.h`

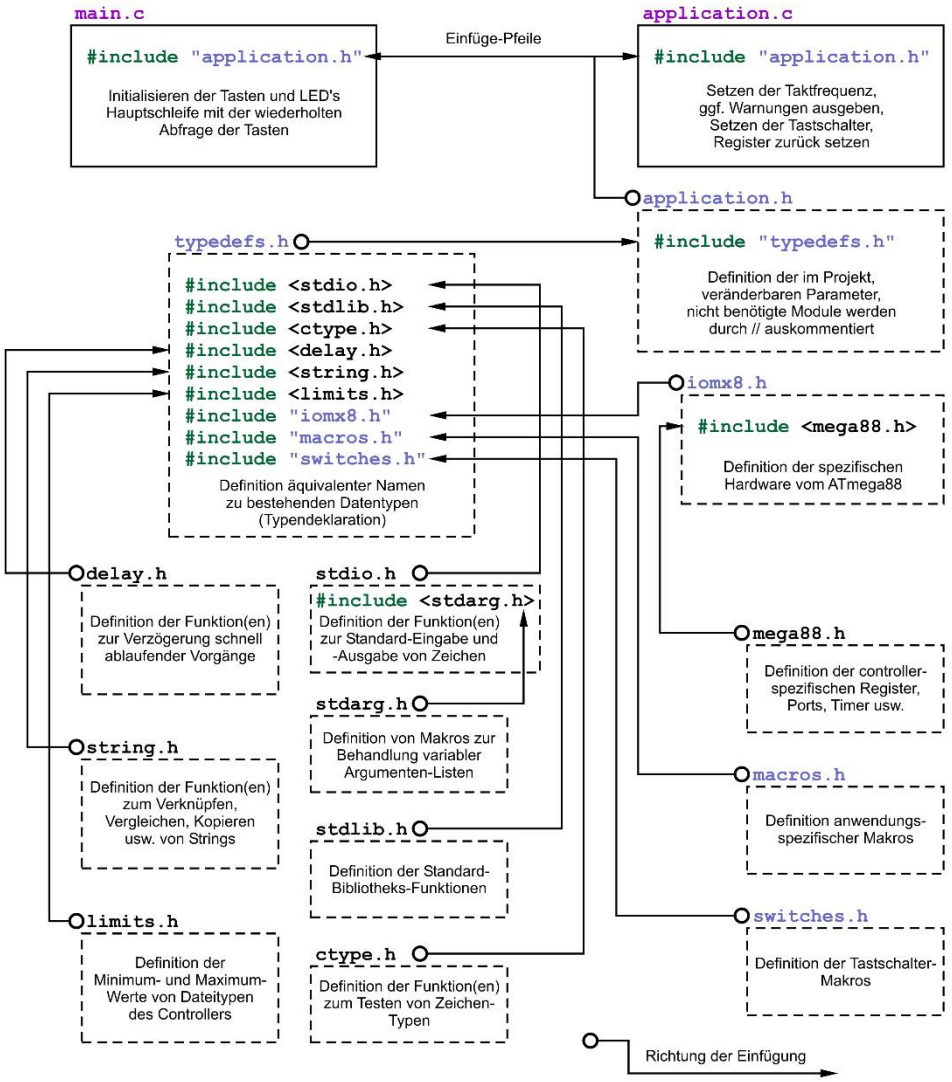


Bild 2.2-01: Datei-Struktur des Projektes AVR_PB_LED (Bildvergrößerung)

Am Ende der Kompilierung sieht der Projekt-Ordner schon viel verwirrender und viel gefüllter aus:

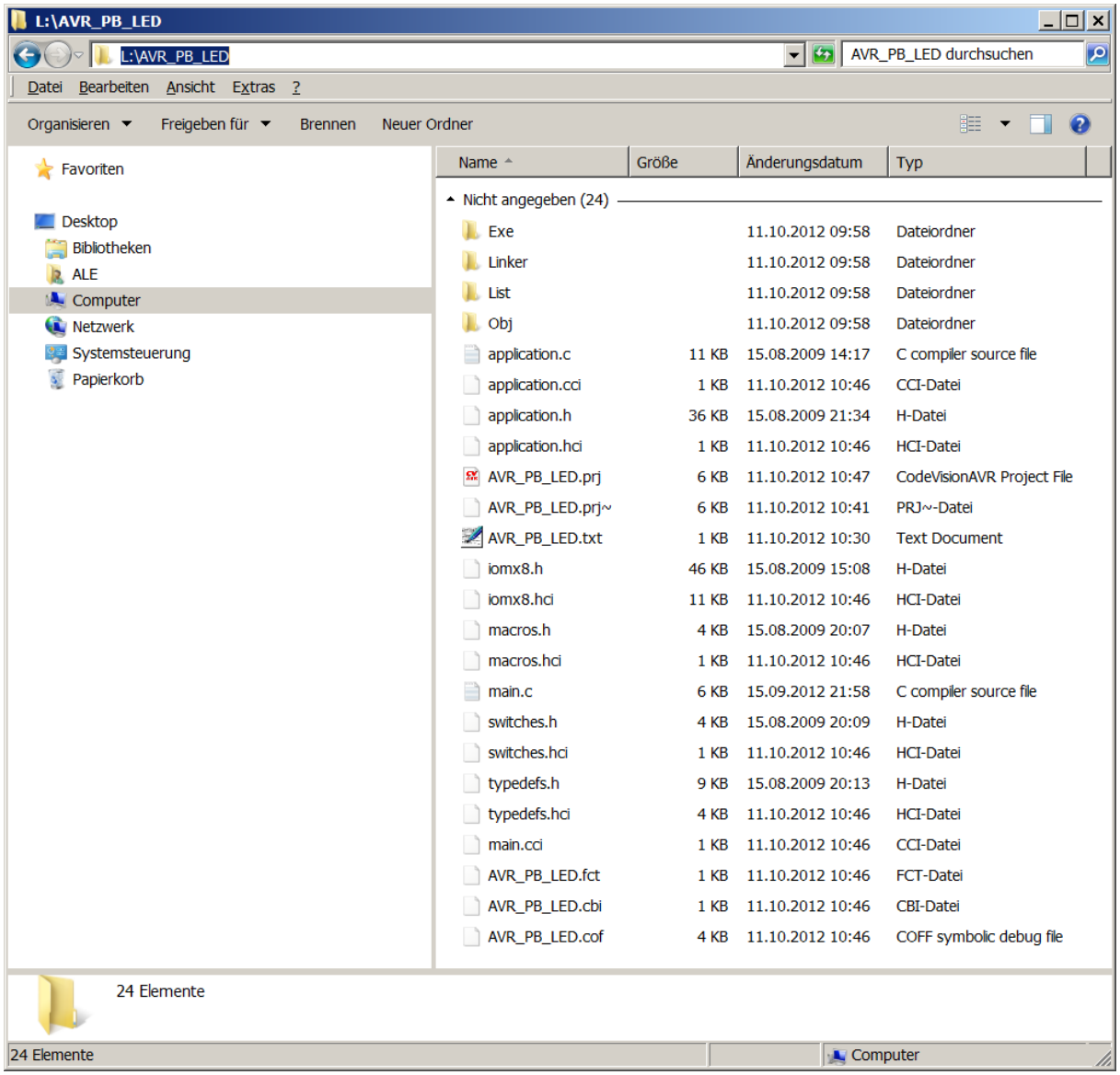


Bild 2.2-02: Die Quell-Dateien und während der Kompilierung angelegte Zwischen-Dateien

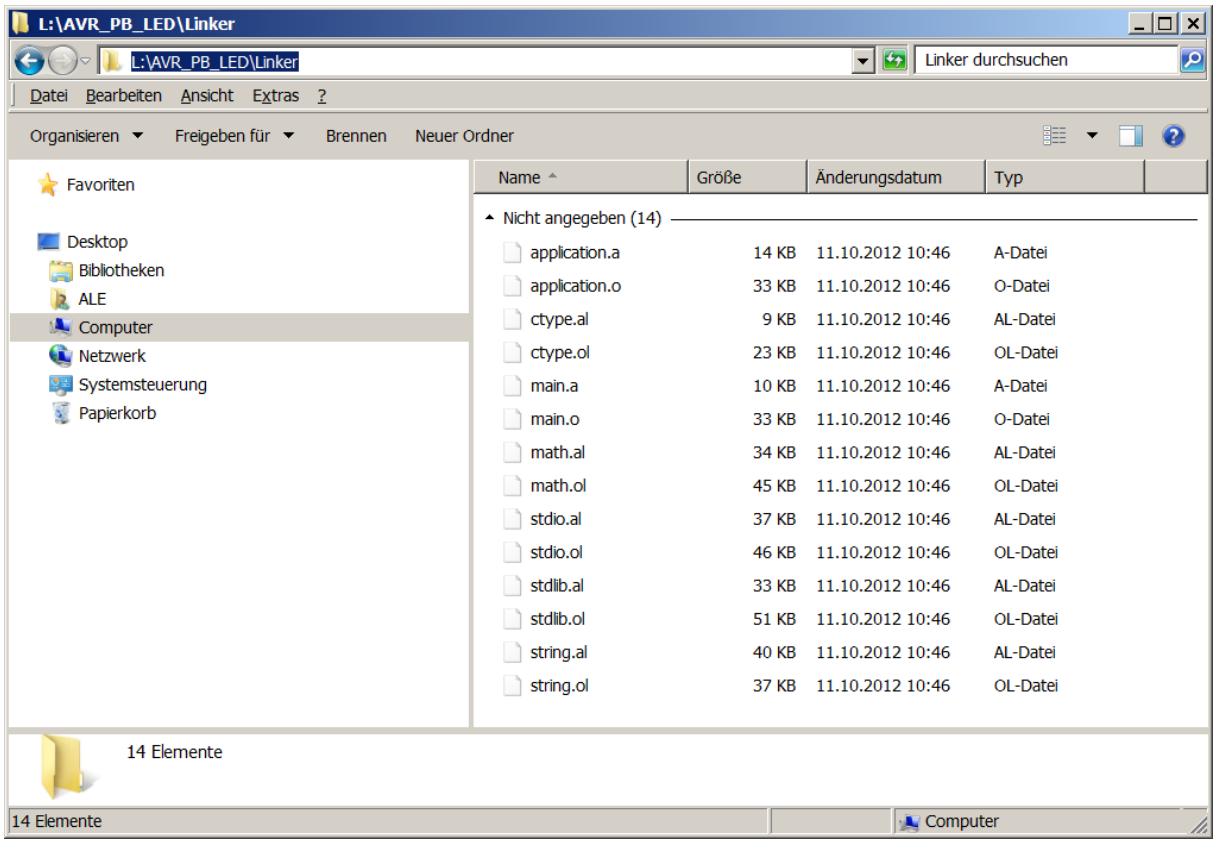


Bild 2.2-03: Die Linker-Dateien

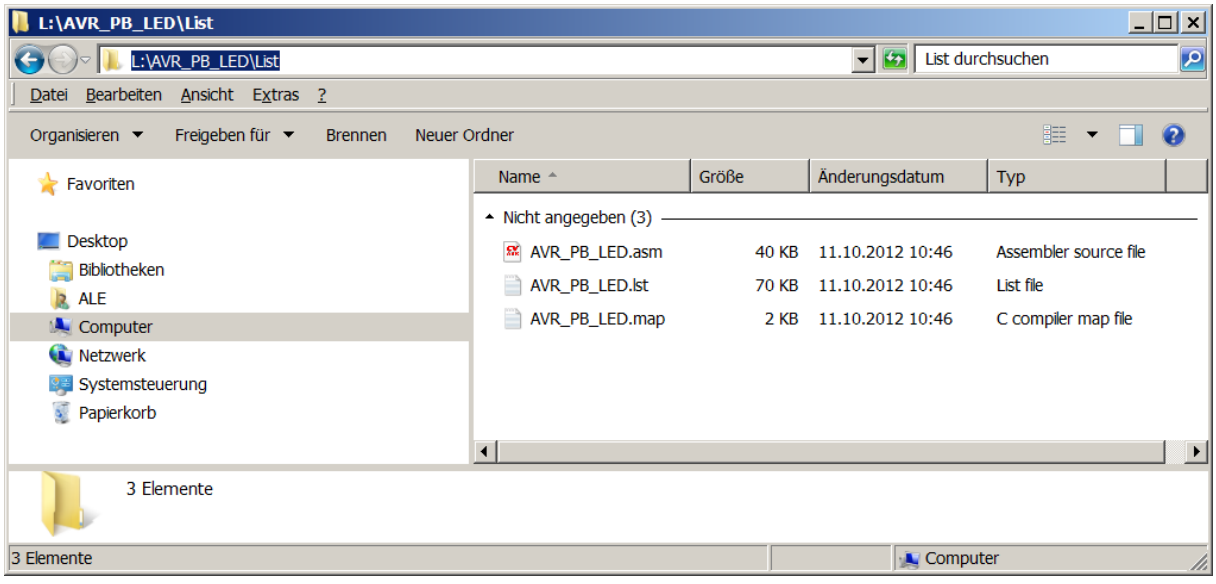


Bild 2.2-04: Die List-Dateien

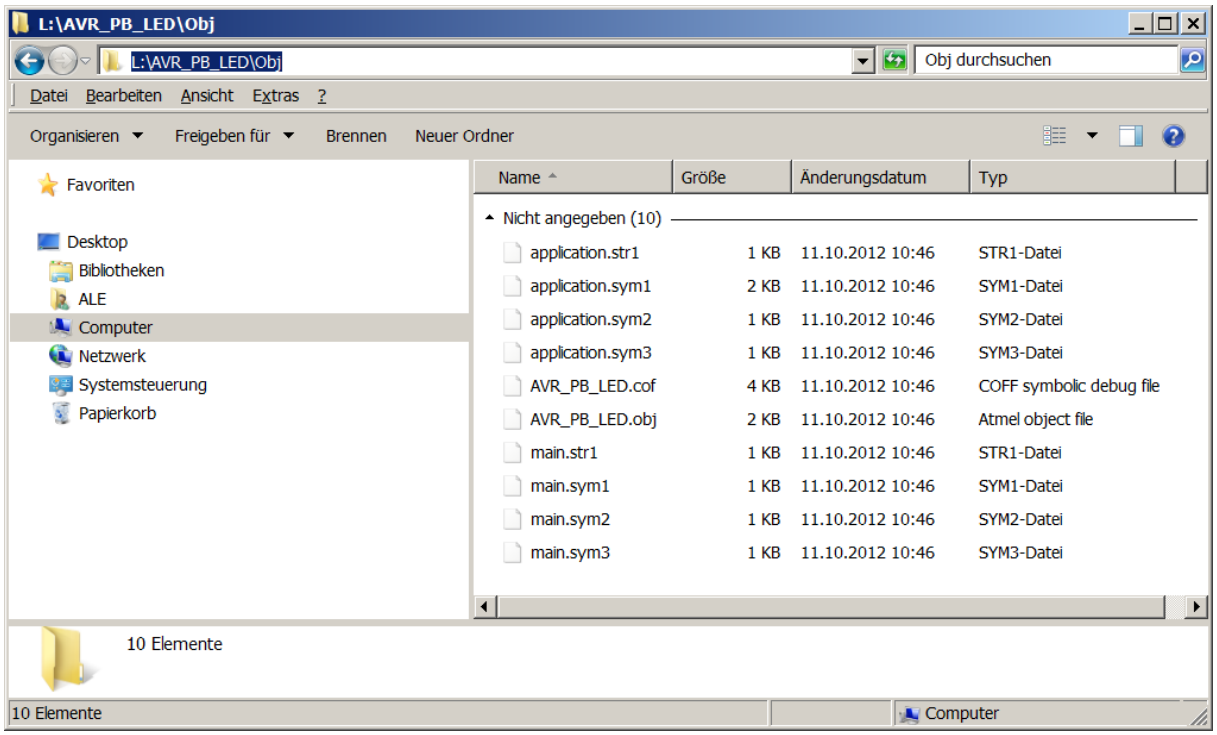


Bild 2.2-05: Die Obj-Dateien

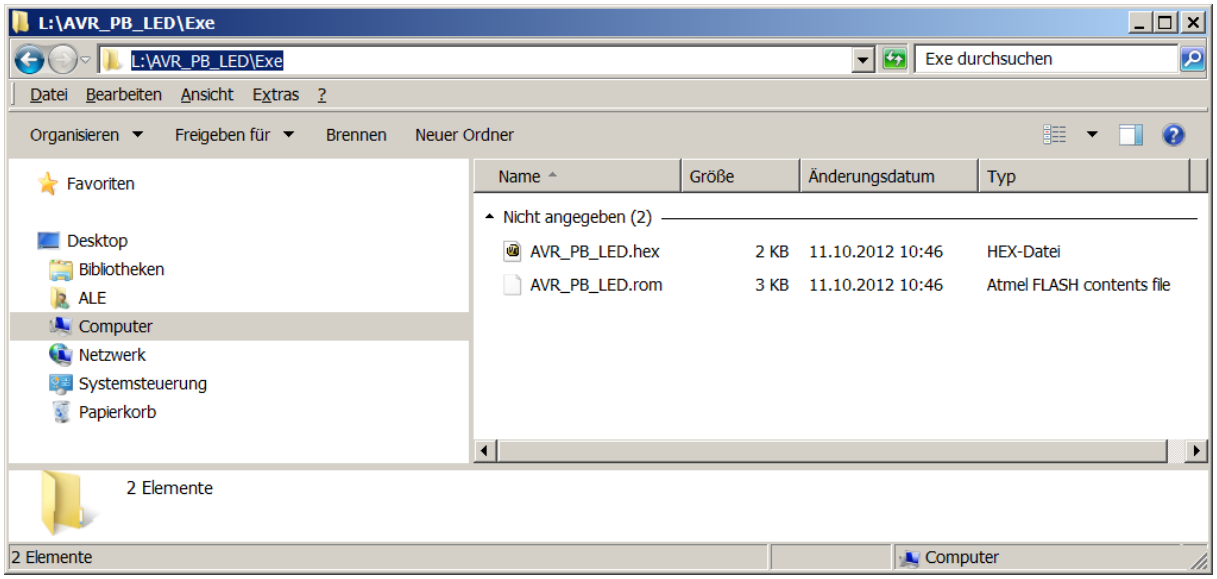


Bild 2.2-06: Die Exe-Dateien

Die Datei **`AVR_PB_LED.hex`** ist schließlich die Datei, die in den Controller geflasht und dann zum Laufen gebracht wird.

2.3 Einbindung von AVR Studio in den CVAVR

Hinweis: Wer gleich ins "Volle" gehen will, der kann bei **2.5 C-Compiler-Optionen (Arbeiten mit dem CVAVR)** gleich weiterarbeiten!

Der Compiler CVAVR ist dafür vorgesehen, mit dem Debugger vom AVR Studio ab der Version 4.13 zu kommunizieren. Dazu ist es notwendig, die Lokalisation von AVR Studio, d.h. den Ordner-Pfad von AVR Studio auf der System-Platte, der Anwendung CVAVR bekannt zu machen. Diese Lokalisation ist (unter der Annahme, dass AVR Studio auf seinem "angestammten" Platz installiert wurde):

`C:\Program Files (x86)\Atmel\AVR Tools\AvrStudio4\AVRStudio.exe`

Der Eintrag in CVAVR wird im Menü **Debugger Settings** vorgenommen:

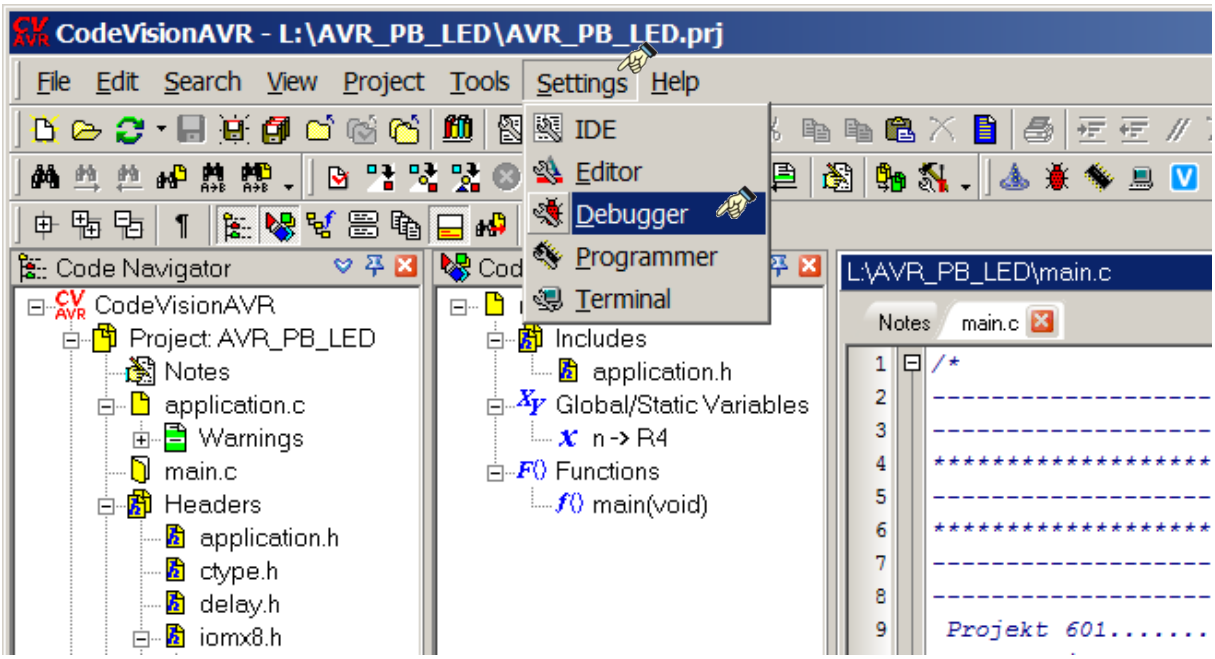


Bild 2.3-01: Debugger-Eintrag in CVAVR vornehmen

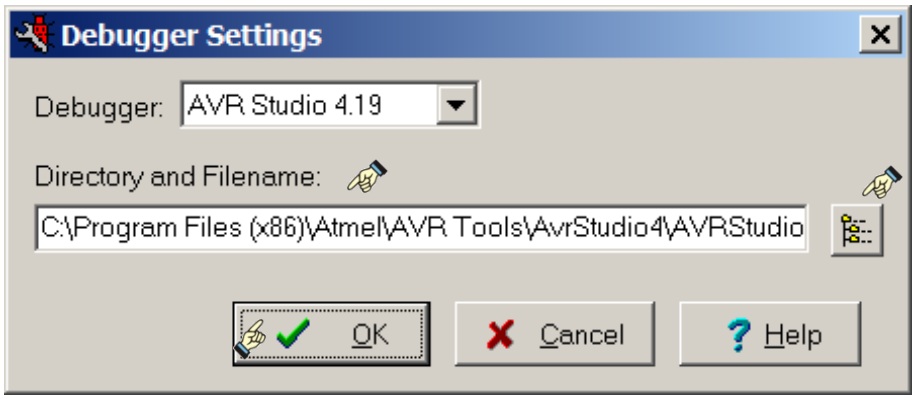


Bild 2.3-02: Einstellen der Datei AVRStudio.exe

Settings => Debugger => Debugger Settings => Directory and Filename:
`C:\Program Files (x86)\Atmel\AVR Tools\AvrStudio4\AVRStudio.exe`
=> OK

Sollte `AVRStudio.exe` nicht an der vorgesehenen Lokalisation installiert worden sein, so kann sie mit der "File-Such-Funktion"  gefunden werden:

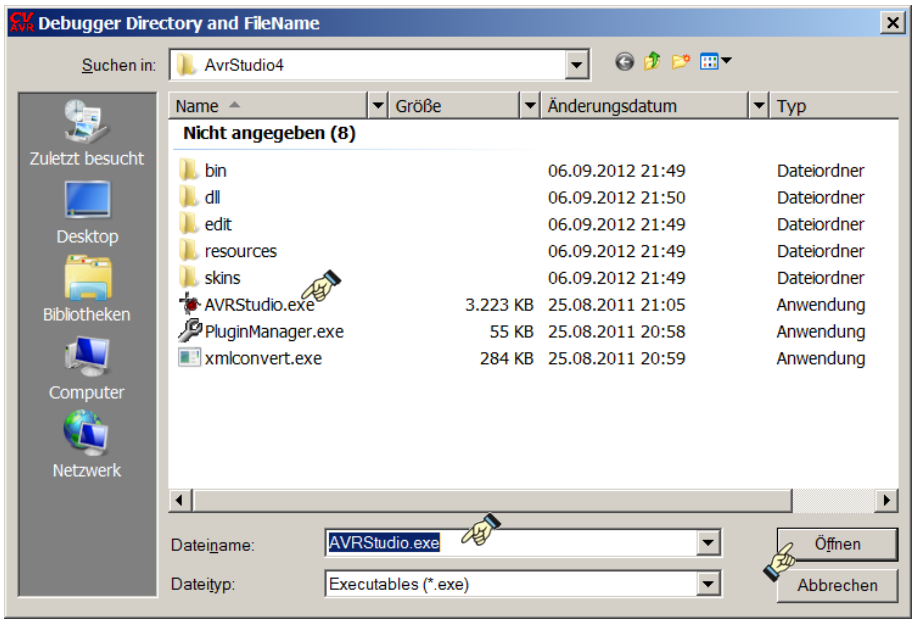


Bild 2.3-03: Suchen der Datei AVRStudio.exe

Hinweis: Hier findet ein gleitender Übergang zum nächsten Abschnitt statt. Man befindet sich scheinbar noch in der Anwendung **CVAVR**. Denn aus dieser heraus wird der **Debugger** aufgerufen, der aber schon Bestandteil des **AVR Studio** ist.

2.4 AVR Studio Debugger

Mit dem Debugger des AVR Studios kann man - bevor man den Objekt-Code des fertig kompilierten Projektes in den Flash-Speicher des Mikrocontrollers überträgt - Schritt für Schritt den Programmablauf simulieren.

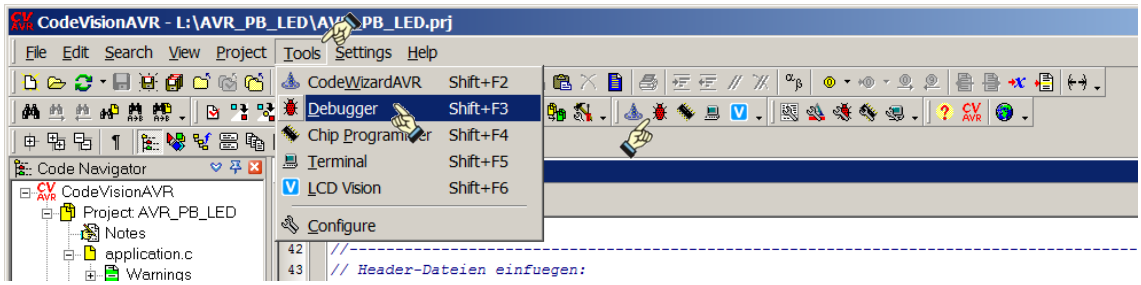


Bild 2.4-01: Aufruf des Debuggers zum Projekt AVR_PB_LED

Tools => Debugger (Shift+F3) => öffnet AVR Studio => File => Open File...(Strg+O)

Es startet AVR Studio (aus CVAVR heraus !):

Dabei wird ggf. das Startfenster mit einem Willkommensdialog geöffnet, wenn die Option **Show this dialog on open** aktiviert war. Da nicht mit **Assembler** oder **AVR GCC** gearbeitet wird, sondern ein AVR-Projekt mit **CVAVR** erstellt werden soll, wird das Häkchen bei **Show dialog at startup** entfernt - und das Fensterchen erscheint zukünftig nicht mehr.

Es ist zu beachten, dass bereits einige Optionen voreingestellt worden sein können, so dass das Abbild von AVR Studio nach der ursprünglichen Installation anders ausgesehen haben könnte.

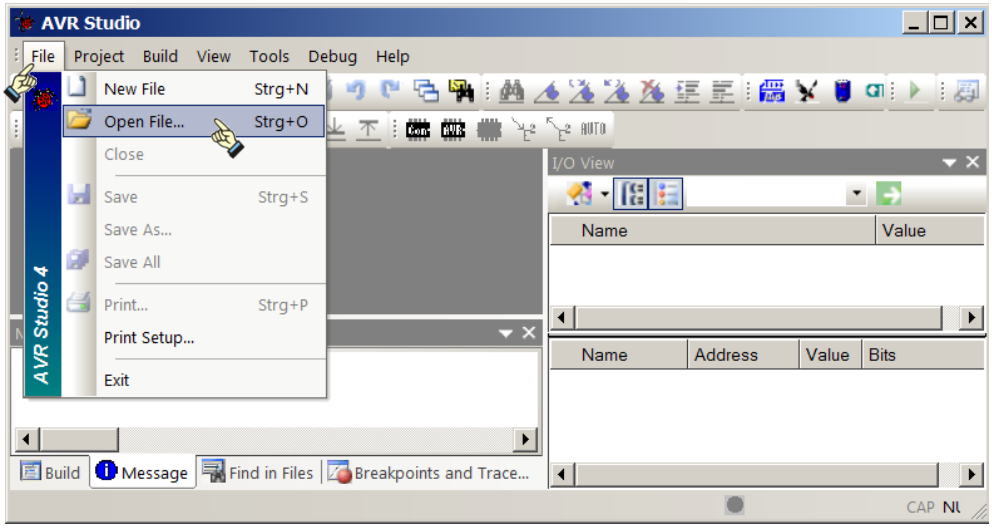


Bild 2.4-02: Öffnen des Datei-Dialogs

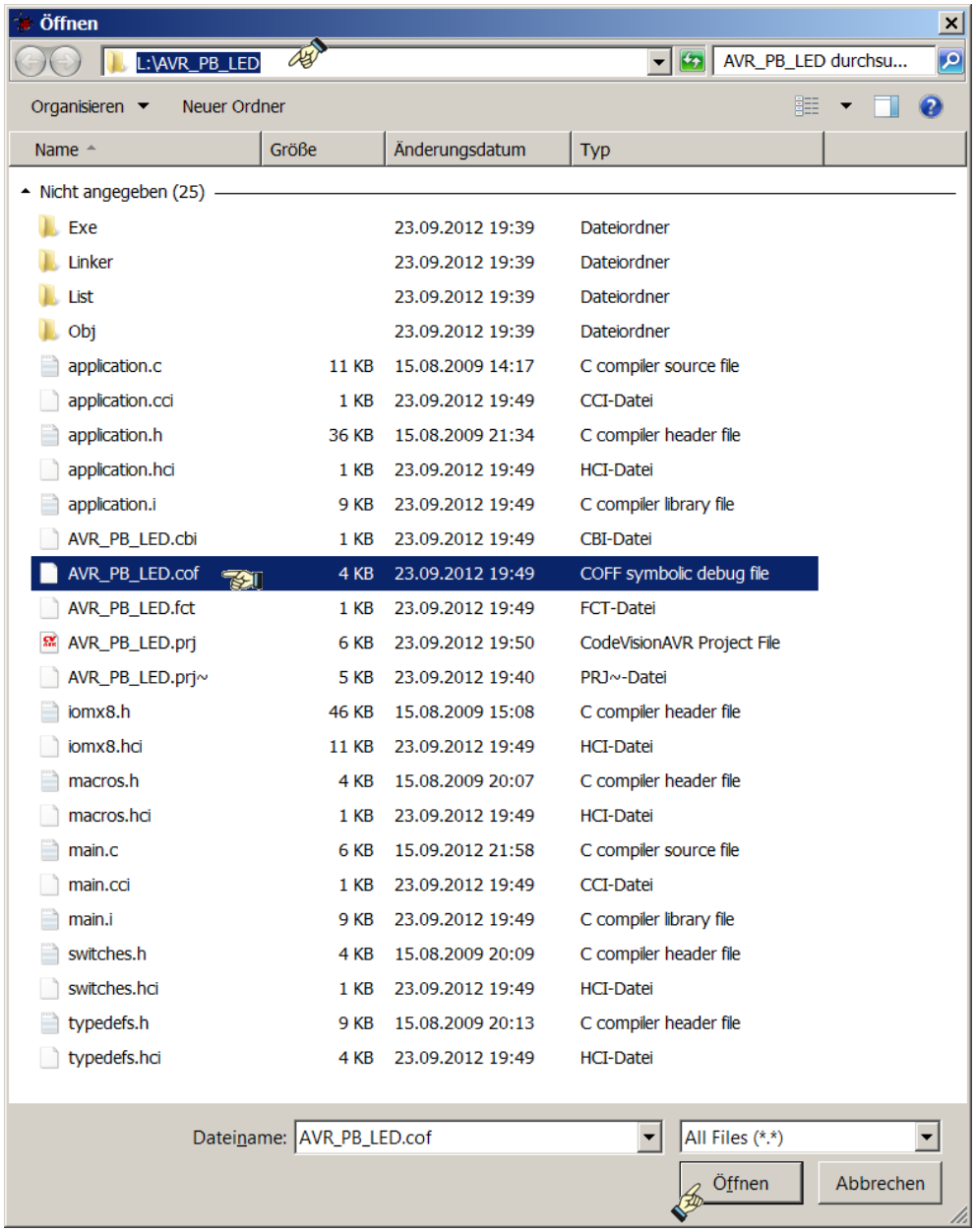


Bild 2.4-03: Suchen der COFF-Datei

Die benötigte **COFF**-Datei befindet sich im Verzeichnis **`L:\AVR_PB_LED`** des Projekt-Ordners vom Beispiel-Projekt **AVR_PB_LED** und hierher soll auch die Datei **`AVR_PB_LED.cof.aps`** abgespeichert werden:

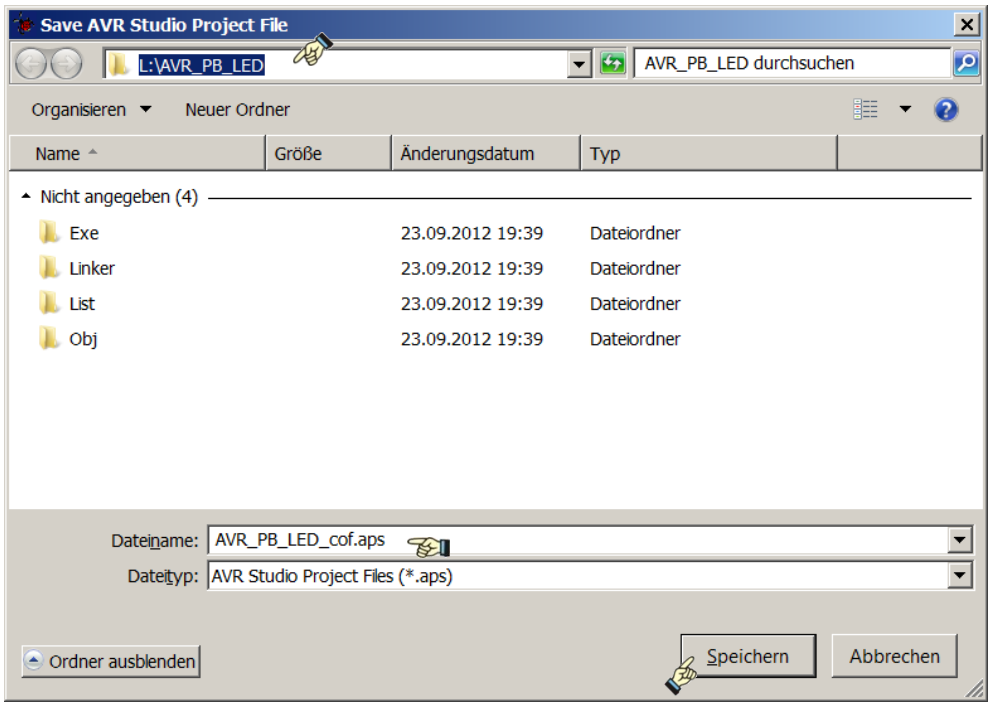


Bild 2.4-04: `AVR_PB_LED.cof.aps` speichern

Es dauert einen Moment, bis man die Debug-Plattform **AVR Simulator** für den Mikrocontroller **ATmega88** einstellen kann:

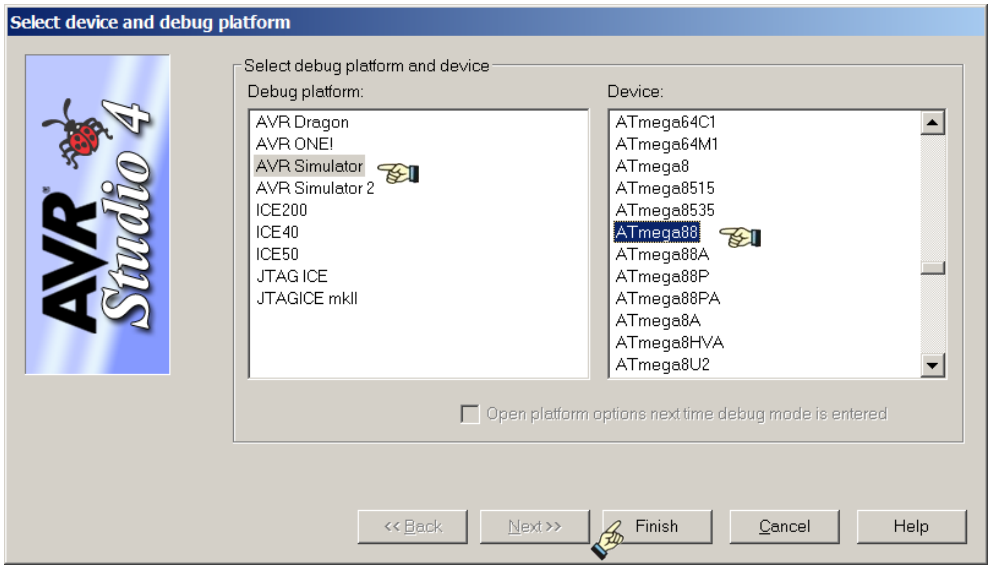


Bild 2.4-05: Debug-Plattform für den ATmega88 einstellen

Nach einer weiteren kurzen Verweildauer öffnet sich AVR Studio mit der für das C-Projekt **AVR_PB_LED** eingestellten Debug-Plattform mit zahlreichen **Toolbars**, von denen besonders hervorzuheben sind:

- Prozessor-Daten (Counter, Pointer, Register usw.)
- Die aktuelle Quell-Datei **`D:\AVR_PB_LED\main.c`** mit einem gelben Pfeil auf die erste ausführbare Anweisung
- Ein-/Ausgabe-Werte (I/O View; Hardware-Ressourcen)
- Nachrichten über die laufenden Aktivitäten (Message Output)

Jetzt "schiebt" man sich die Seiten der Felder so hin, dass eine vernünftige Übersicht erscheint:

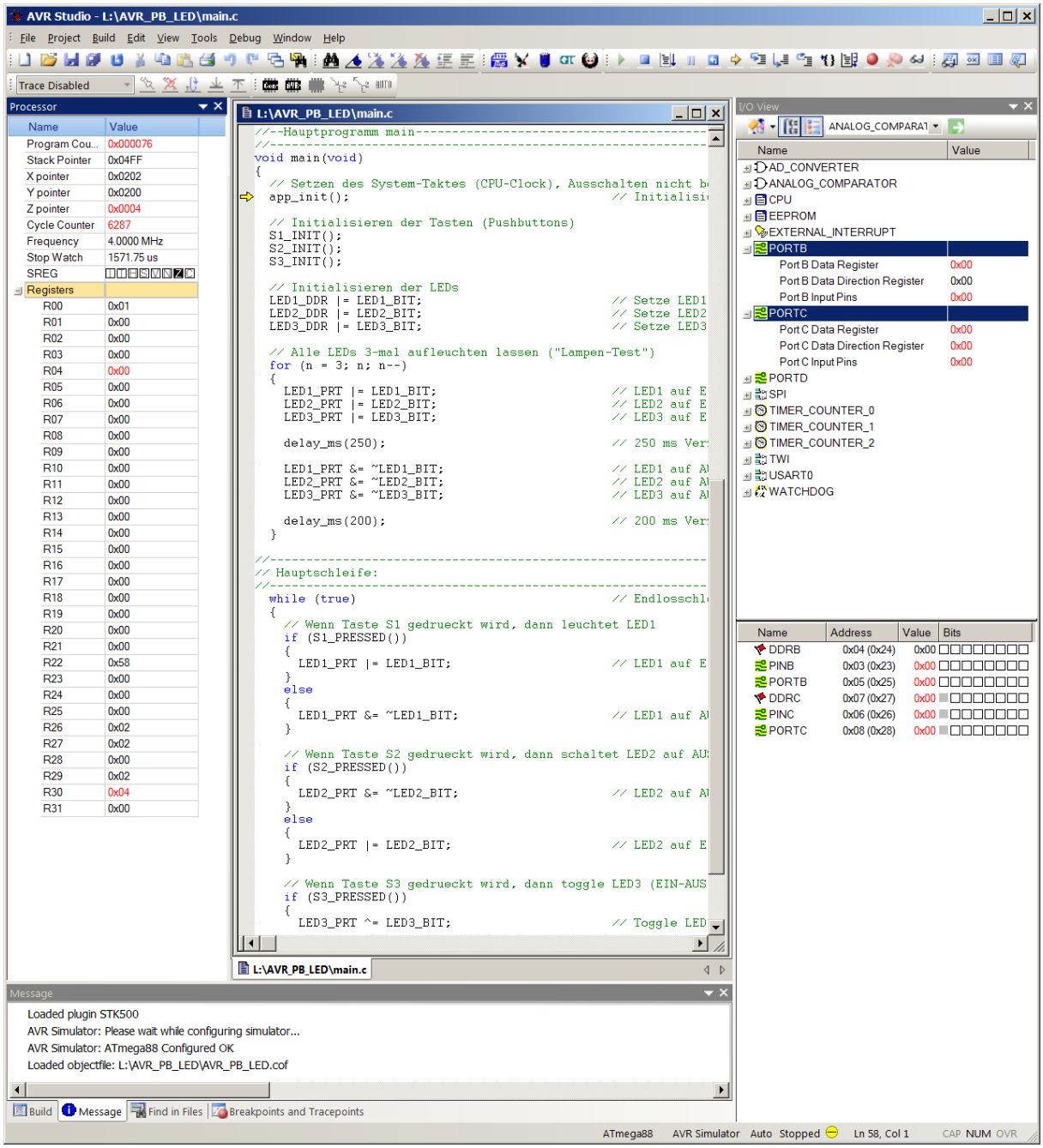


Bild 2.4-06: Debug-Plattform mit Quell-Datei `main.c` (Bildvergrößerung)

Mit der Funktionstaste **F10** bzw. **F11** vom **PC** kann man nun Schritt für Schritt den Programmablauf simulieren. Wenn man dabei den Cursor auf eine Variable positioniert, dann bekommt man den jeweiligen Wert angezeigt. Im Menüpunkt **Debug** kann man sehen was sonst noch so alles möglich ist. Hier nur ein paar Hinweise der zahlreichen Möglichkeiten des Debuggers:

Wenn man in CVAVR eine Änderung im Quell-Programm vornimmt, neu kompiliert und wieder nach AVR Studio wechselt, dann merkt AVR Studio, dass sich die ***.cof**-Datei geändert hat und fragt nach, ob diese Datei neu geladen werden soll. Wenn diese Anfrage mit **JA** beantwortet wird, dann startet der Debugger nach dem Neuladen wieder von vorne.

Wenn AVR Studio neu aus CVAVR heraus gestartet wird, dann kann ein vorhandenes Projekt unter dem Menüpunkt **Project => Recent Project *Projektname*** wieder unkompliziert geladen werden.

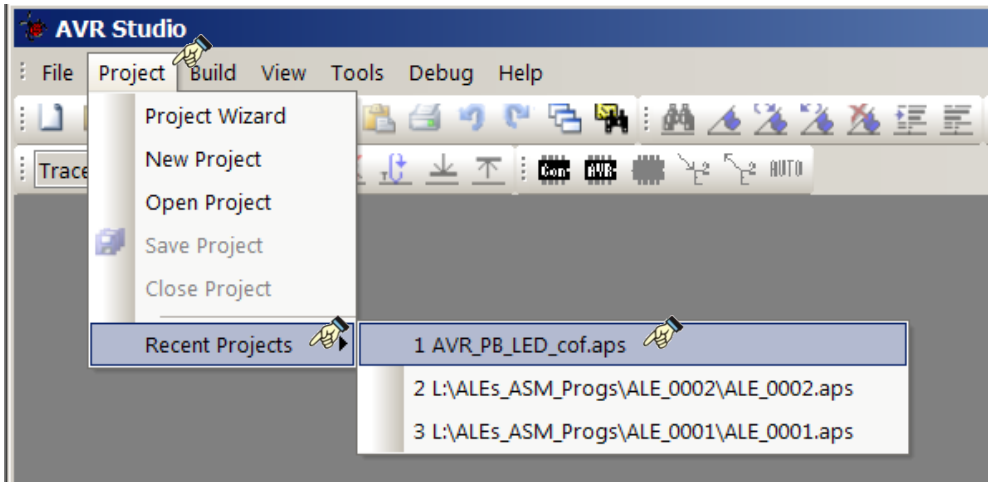


Bild 2.4-07: Fortsetzen des Debugging nach einem Neustart

Wenn man in diesem Projekt beispielsweise auf **PORTC** klickt, dann kann man während der schrittweisen Programmausführung (mit **F10**) sehen, wie sich **PC2**, **PC3**, **PC4** (entsprechend dem kurzen Aufleuchten der LED's) nach dem Reset verändern.

Es gibt jede Menge Möglichkeiten, Neues zu entdecken und auszuprobieren!

Nur Mut- es kann nichts Schlimmes passieren!

Der Mikrocontroller ist geduldig !!!

2.5 C-Compiler-Optionen (Arbeiten mit dem CVAVR)

Wenn das Projekt trivial ist (so wie hier) und beim Kompilieren keine Errors aufgetreten sind, kann man davon ausgehen, dass auch der Test-Lauf auf dem Testboard fehlerfrei verläuft.

Es sei denn: Es hat sich ein **LOGISCHER FEHLER** eingeschlichen, den der Compiler natürlich nicht bemerken kann! Dieser wird aber beim Test-Lauf bemerkt, weil die Funktion nicht so funktioniert wie sie soll.

Um das schnell herauszufinden, dafür ist das Testboard ja da!

Es muss also die Datei **AVR_PB_LED.hex** ist den Mikrocontroller übertragen - geflasht - werden. Der Flash-Vorgang sei hier als eine Kopie des Abschnitts **6.5 Flashen eines C-Programms in ein Mikrocontroller ATmega88** aus dem **Teil 206 C-Compiler und AVR Studio** wiederholt**:**

Zunächst wird das Testboard beschaltet so wie es auch im Abschnitt **1.2 Beschaltung** im **Teil 601 - PB_LED** beschrieben wird:

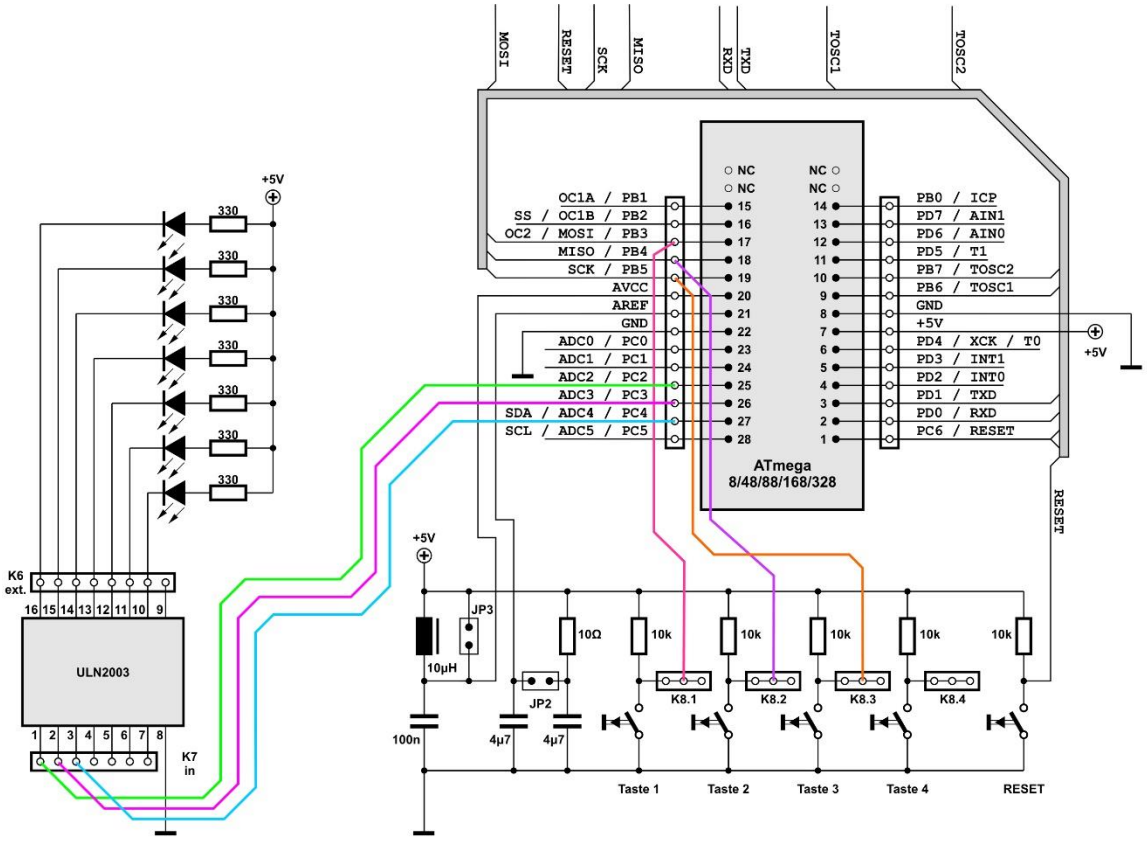


Bild 2.5-01: Beschaltung der LEDs und Taster (Bildvergrößerung)

Dann wird das Board mit seinem Netzteil mit Spannung versorgt.

Der ISP-Programmieradapter wird mit seinem USB-Stecker in einen freien USB-Steckplatz des PCs gesteckt, in dem das neue Programm abgespeichert wurde.

Überflüssige Schaltkreise und die Stromversorgung wurden in dem Beschaltungsbild weggelassen. AVR Studio wird nun erneut gestartet.

Dann wird die logische Verbindung zum **CC2-AVR Programmer** (ISP-Programmieradapter) hergestellt. Die Einstellungen wurden ja schon im Abschnitt **4.3.2 Mikrocontroller-Einstellungen im AVR Studio** von **Teil 204 AVR Studio** vorgenommen, so dass man sich damit nicht mehr aufhalten muss:

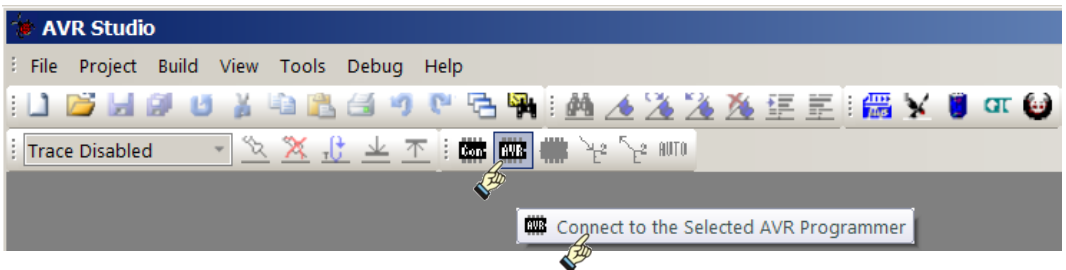


Bild 2.5-02: Mit dem CC2-AVR-Programmer eine Verbindung herstellen

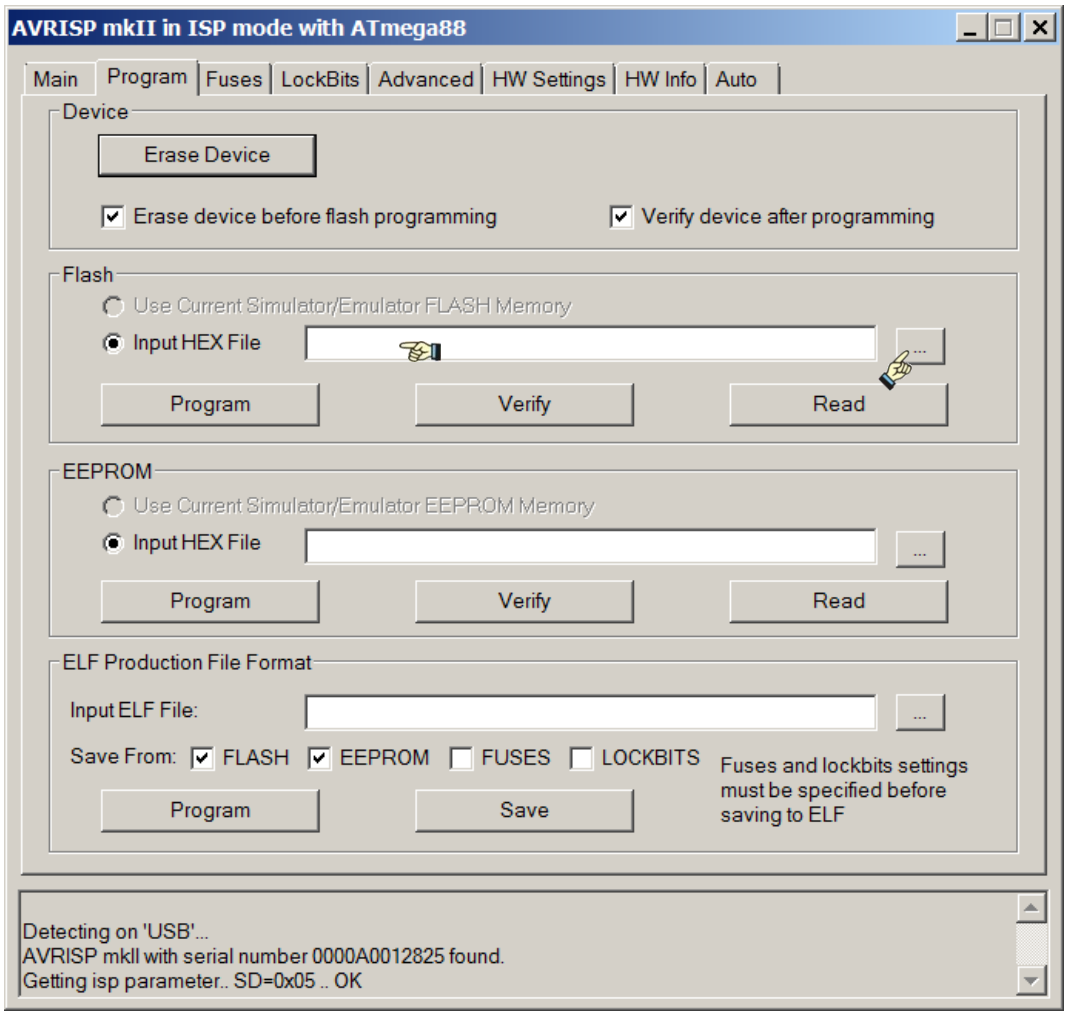


Bild 2.5-03: Objektdatei (*.hex-File) suchen

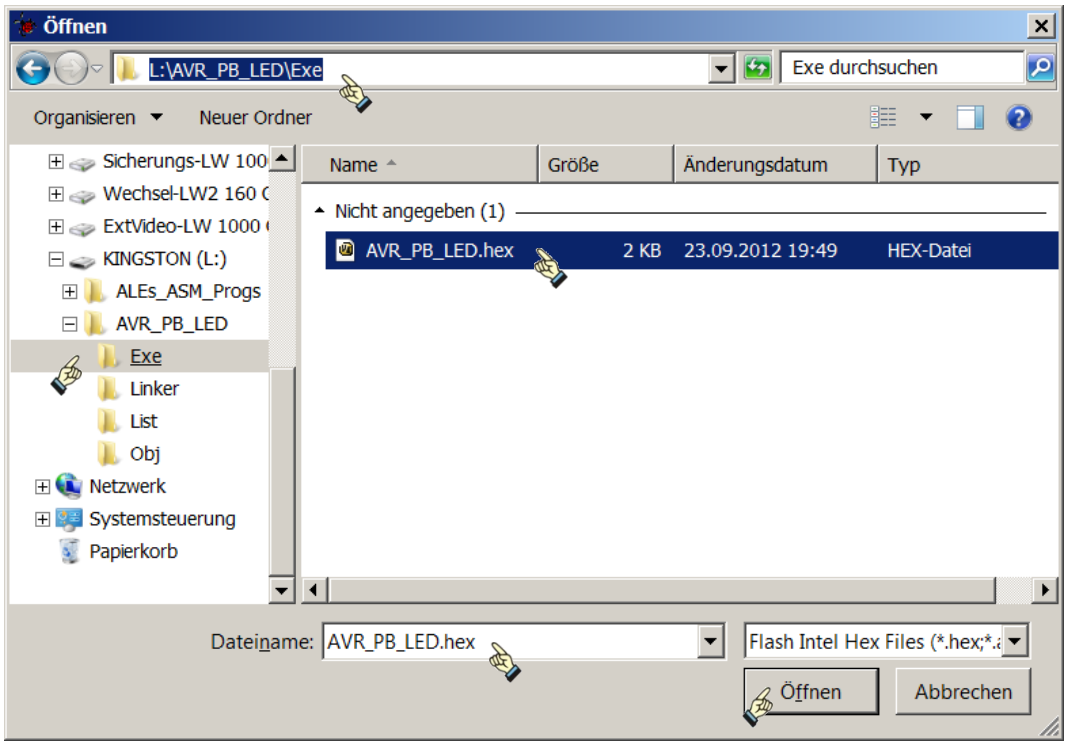


Bild 2.5-04: Objektdatei öffnen

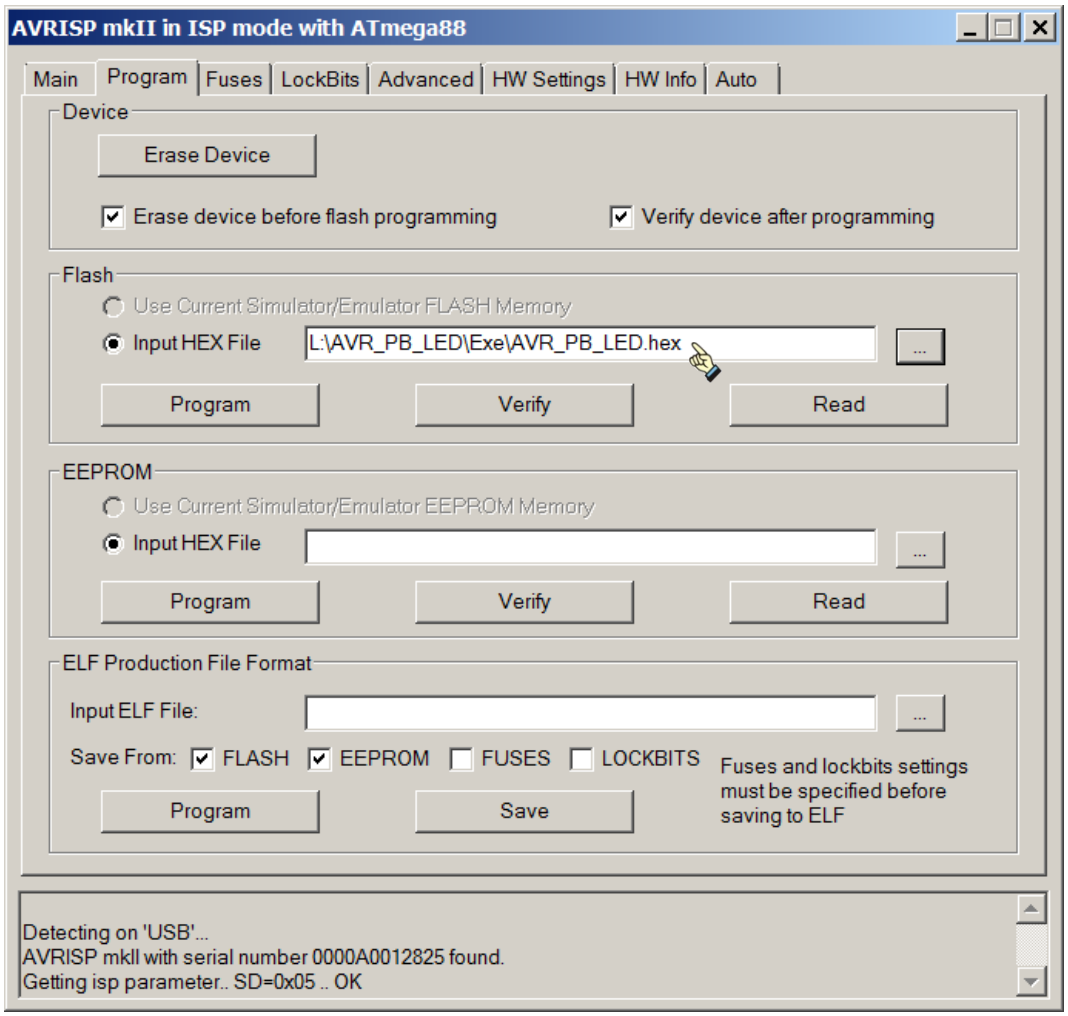


Bild 2.5-05: Programm flashen ==> und das Programm testen ==> alles O.K.

Und das Objekt-Programm ist immer wieder schnell geändert und schnell in den Controller geflasht, so dass dieser Test schnell und oft gestartet werden kann - bis die **LOGIK** stimmt!

CVAVR wird erneut aufgerufen. Der Compiler hat sich die "alte" Einstellung gemerkt, so dass gleich mit dem Projekt **AVR_PB_LED** weiter gearbeitet werden kann. Sollte inzwischen an einem anderen Projekt gearbeitet worden sein, so ist das Projekt über seine Projekt-Datei (hier: **`AVR_PB_LED.prj`**) schnell wieder mit seinen ganzen Dateien geladen.

Es wird angenommen, dass das Programm geändert wurde - **WIE**, das sei an dieser Stelle nicht von Bedeutung.

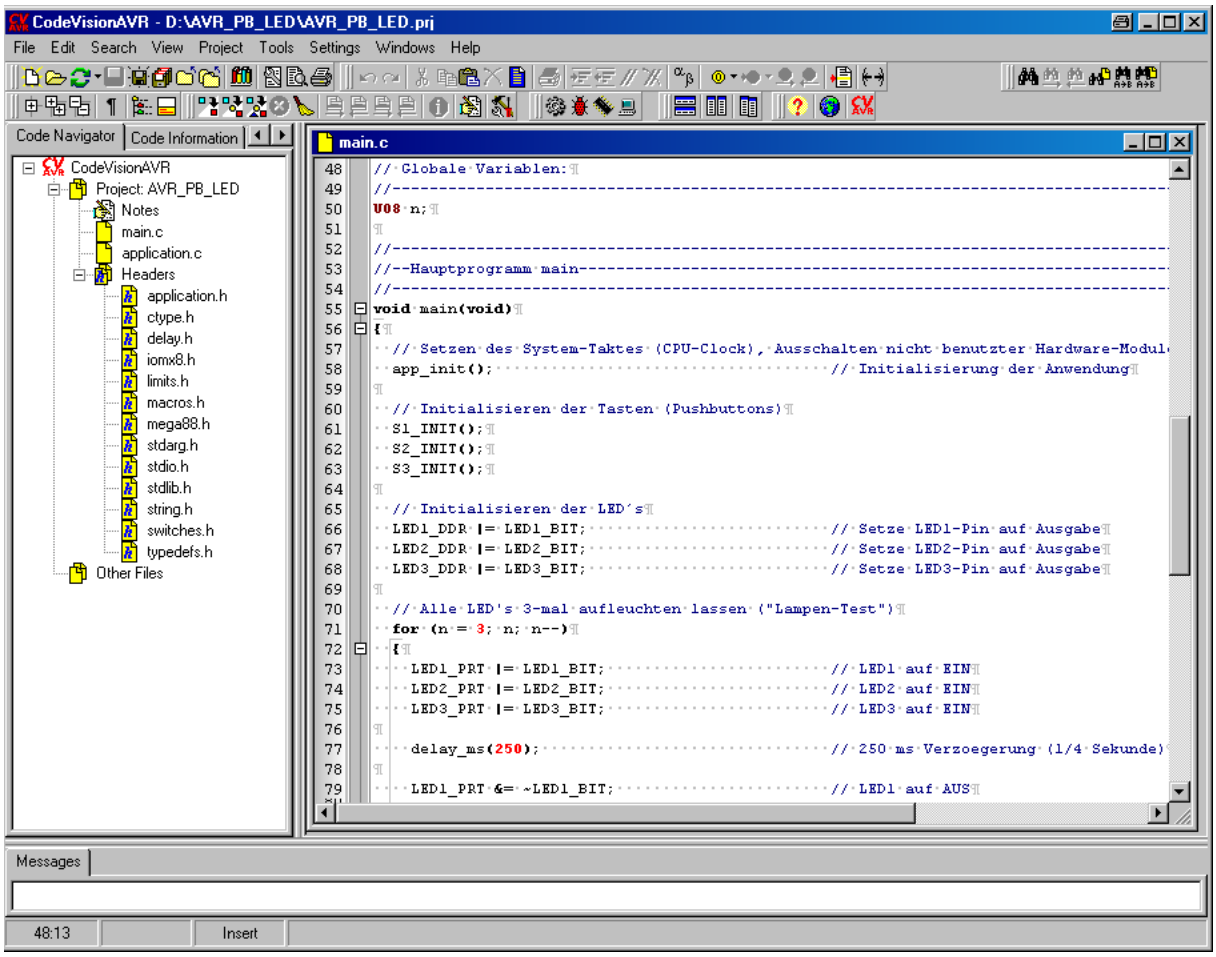


Bild 2.5-06: Aufruf des veränderten Projektes AVR_PB_LED

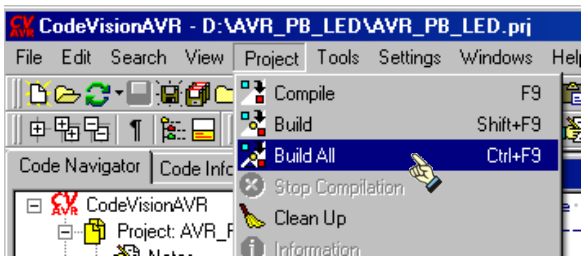


Bild 2.5-07: Neu-Kompilierung

Es wird ja angenommen, dass gerade eine Veränderung in der **`main.c`** vorgenommen wurde, so dass eine Neu-Kompilierung (mit allen Schritten) notwendig wird. Wenn keine Verschlimmbesserungen eingebaut wurden, dann werden wieder die gleichen Listen und Warnungen angezeigt, wie sie die Bilder **Bild 2.1.2-04, -05** und **-06** zeigen.

Bei der Neu-Kompilierung wurde auch eine neue **`AVR_PB_LED.hex`** erzeugt, die vom AVR Studio in den Mikrocontroller geflasht werden muss. Unter der Voraussetzung, dass das Testboard einschließlich der benötigten Ressourcen (dazu gehören auch die Pin-Verbindungen!) über den USBprog mit dem PC verbunden ist, wird AVR Studio gestartet und die logische Verknüpfung zwischen den Projekt-Programm **`AVR_PB_LED.hex`** auf dem PC und dem Mikrocontroller auf dem Testboard hergestellt:

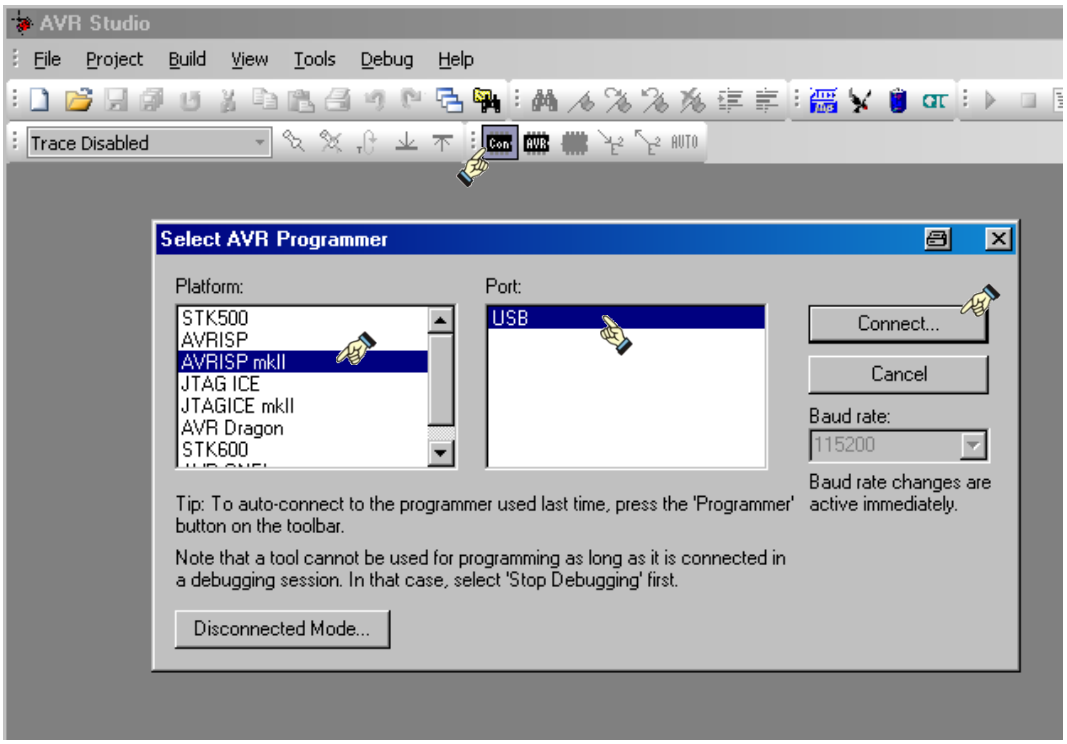


Bild 2.5-08: Logische Verbindung des USBprog mit dem AVR Studio herstellen

Con => AVRISPmkII => USB => Connect...

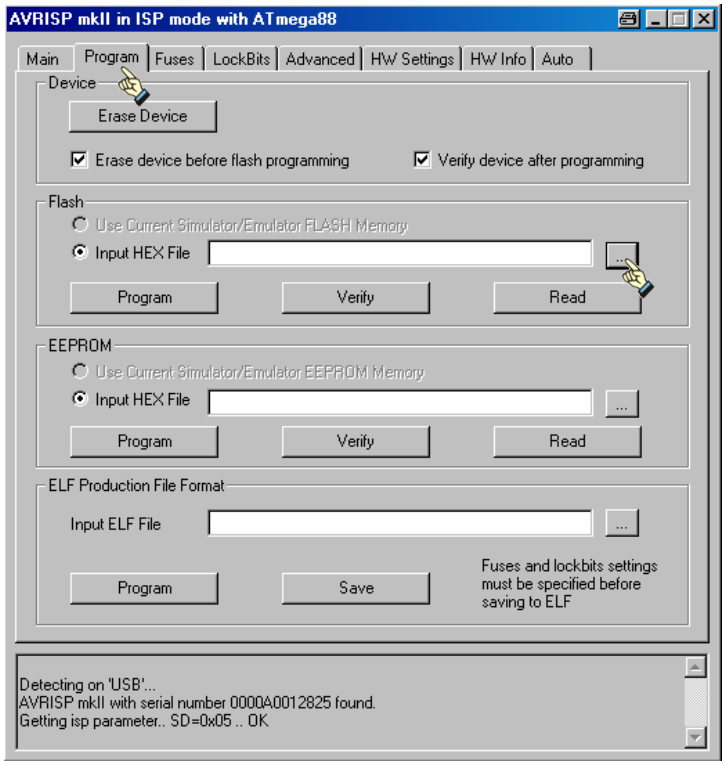


Bild 2.5-09: Suchen der exe-Datei `AVR_PB_LED.hex`

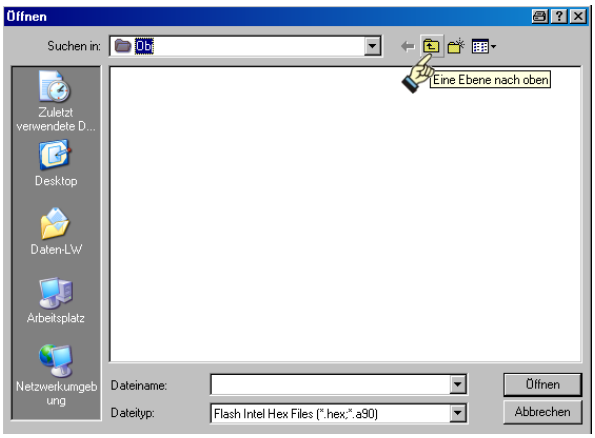


Bild 2.5-10: Unterverzeichnis Exe suchen

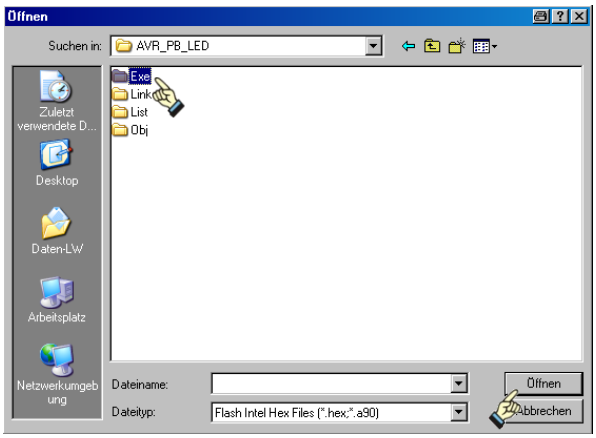


Bild 2.5-11: Unterverzeichnis Exe öffnen

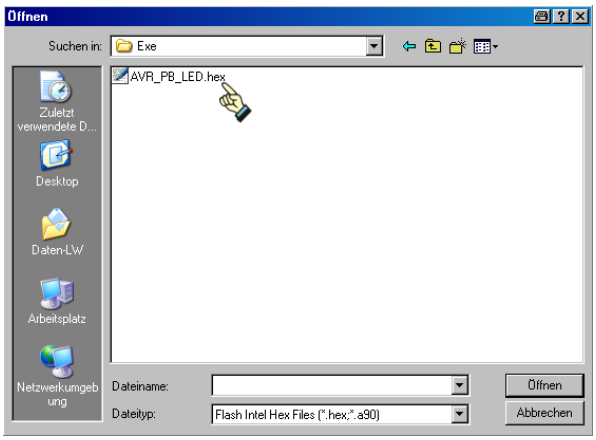


Bild 2.5-12: `AVR_PB_LED.hex` anklicken

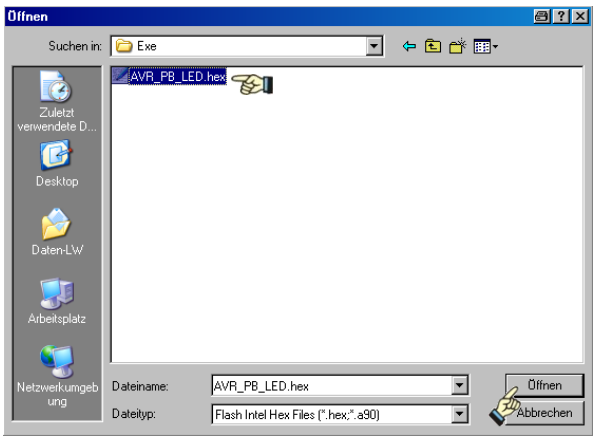


Bild 2.5-13: `AVR_PB_LED.hex` öffnen

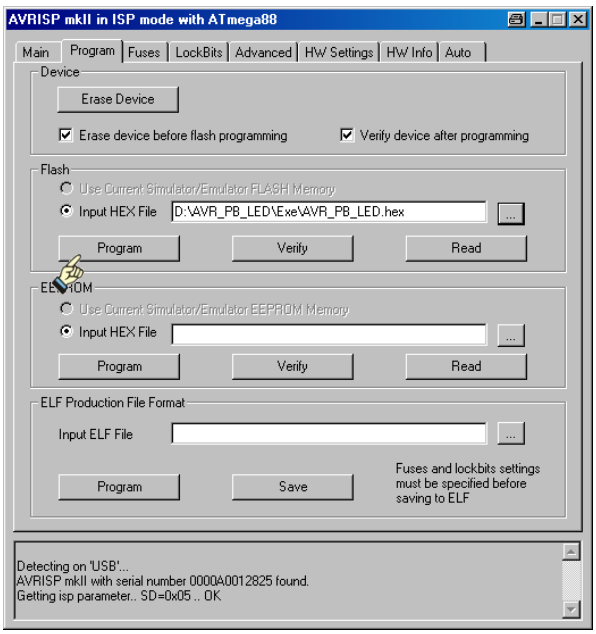


Bild 2.5-14: `AVR_PB_LED.hex` flashen

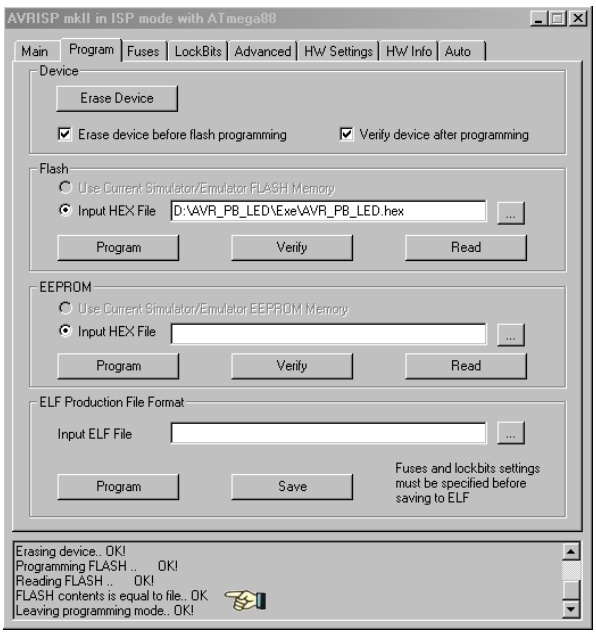


Bild 2.5-15: `AVR_PB_LED.hex` ist programmiert

HEUREKA! Die Tasten lassen die LED's wie gewollt AN- und AUSgehen!

Was ist zu unternehmen, wenn es nicht gleich so läuft - wenn man sich zum Beispiel vertippt hat??? Kehren wir zum alten immer wieder zitierten Beispiel-Projekt zurück und öffnen das vorhandene **C**-Projekt **AVR_PB_LED**. (Sollte das Projekt nicht mehr vorhanden sein, so ist es ratsam, die Phasen des Abschnittes **2.1 Erzeugen eines C-Projektes** noch einmal abzulaufen).

Im folgenden Beispiel wurde "versehentlich" statt eines Semikolons ein Doppelpunkt gesetzt. Es werden im **Code Navigator** (linke Bildhälfte) angezeigte **Errors** durch Anklicken im Quell-Programm auf der rechten Bildhälfte sofort gefunden und markiert. Sollte das entsprechende Quell-Programm in der rechten Bildhälfte noch nicht geöffnet sein, so wird es geöffnet und die fehlerhafte Anweisung markiert. Der Compiler hat das gemerkt und führt **keine Assemblierung** durch:

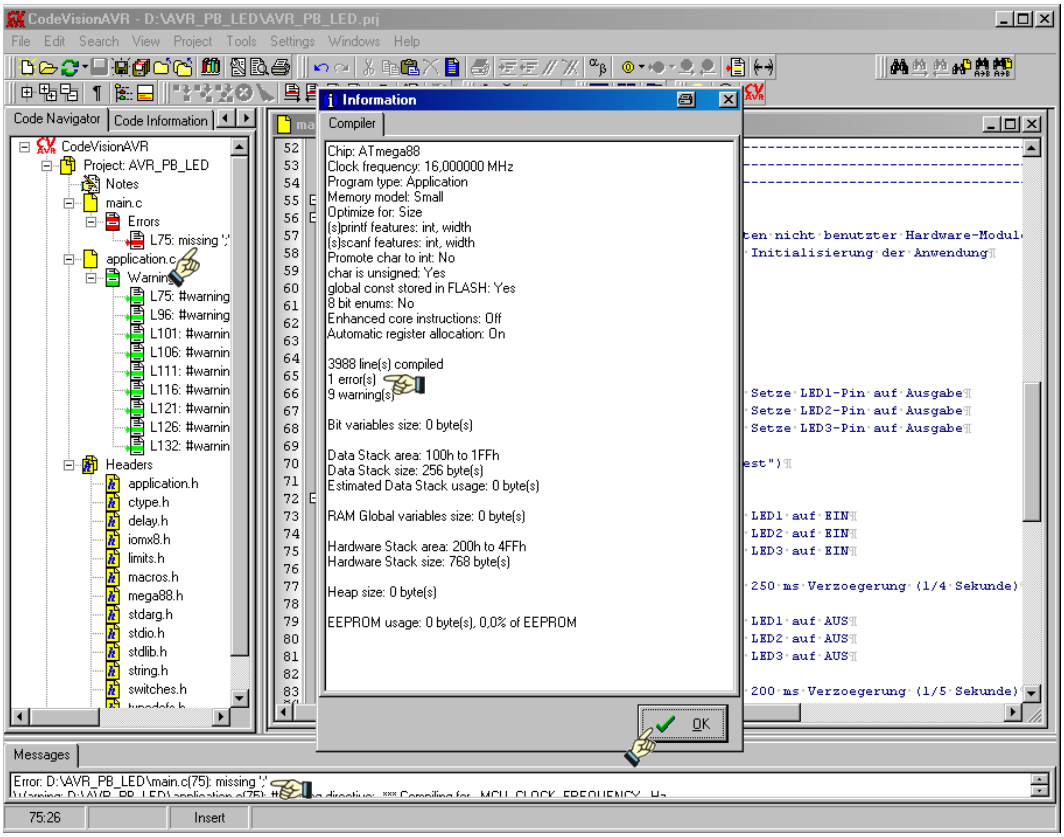


Bild 2.5-16: Fehlererkennung im AVR_PB_LED

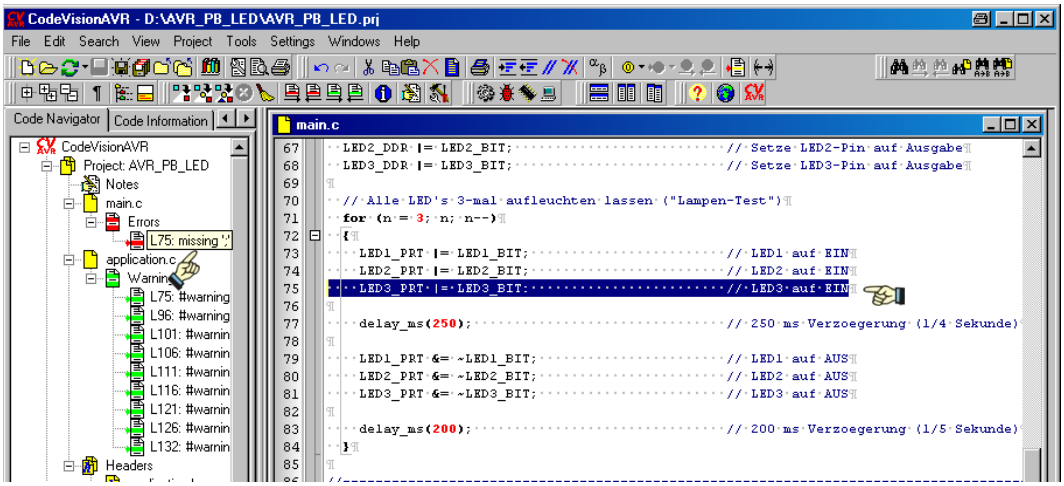


Bild 2.5-17: In Zeile 75 steht ein Doppelpunkt "`:`" statt eines Semikolons "`;`"

Der Fehler wird direkt im rechten Fenster behoben und das Projekt **AVR_PB_LED** neu kompiliert. Dabei wird die Datei **`main.c`** automatisch im Projekt-Ordner berichtigt.

Will man eine gerade nicht angezeigte oder verdeckte **`*.c`**- oder **`*.h`**-Datei zu Editier-Zwecken auf der rechten Bildhälfte öffnen, so klickt man auf die gewünschte Datei in der links angezeigten Ordnerstruktur und schon ist sie da, zum Beispiel die Datei **`application.h`**:

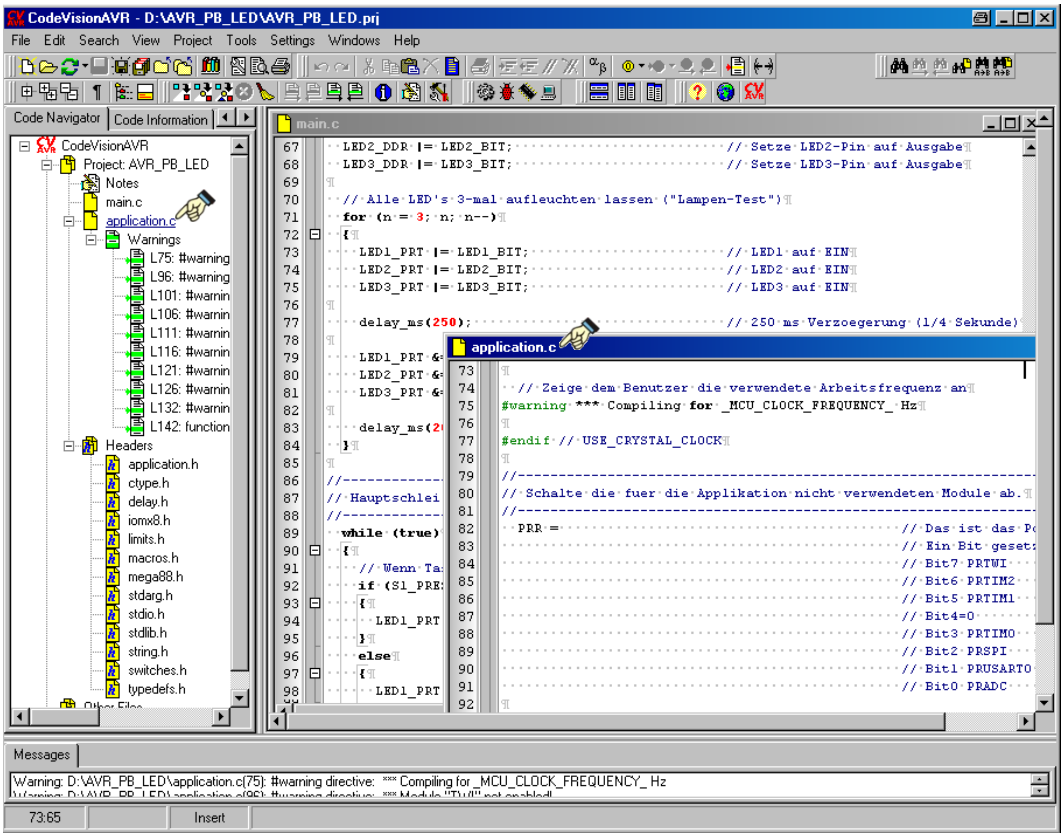


Bild 2.5-18: Öffnen einer Datei z.B. zu Editierzwecken; hier: `application.c`

Die geänderten Anweisungen werden automatisch unmittelbar bei der Neu-Kompilierung auf der Festplatte in der Datei festgehalten, ohne dass eine explizite Anweisung zur Speicherung angefordert wird.

Eine große Hilfe kann es auch bedeuten, sich die in einer Header-Datei aufgeführten Makros (z.B. der Header-Datei **`macros.h`**) auflisten zu lassen. Dazu geht man auf das Register **Code Information**:

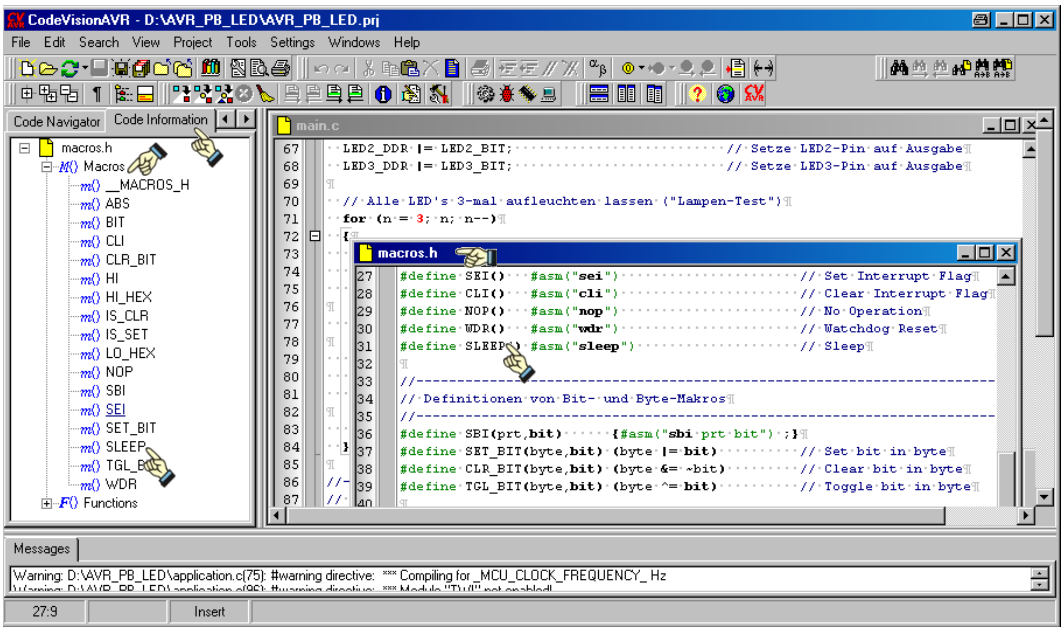


Bild 2.5-19: Auflisten der Makros in der Header-Datei `macros.h`

So gibt es in dem Compiler CVAVR noch eine ganze Menge zu entdecken. Einfach mal viele Knöpfe ausprobieren!

Im Abschnitt **2.1 Erzeugen eines C-Projektes** wurde anhand des praktischen Beispiels **AVR_PB_LED** nur die generelle Abfolge zur Erzeugung eines **C**-Projektes beschrieben, über eine Anordnung von zahlreichen Projekten auf der Festplatte "schwieg des Sängers Höflichkeit". Das Thema wird jedoch dadurch interessant, dass CVAVR zunächst für das **Create New Project** von seinem bestehenden Ordner zur Ablage ausgeht:

`C:\cvavr2\bin`

Es ist wohl wenig sinnvoll, zahlreiche AVR-Projekte gerade im Bereich der System-Platte abzulegen. Abgesehen davon, dass es viele "Ordnungen" gibt und jeder zu Recht auf seine als das "Alleinseligmachende" beharrt, sollte man sich an dieser Stelle darüber Gedanken machen, **WO** und **WIE** man seine und die kopierten Projekte ablegt, schließlich will man sie ja auch schnell wiederfinden. In dem Zusammenhang ist auch die "schwierigste Frage" zu stellen, die jeder Programmierer immer wieder von neuem beantworten muss: "Wie soll das Kind (besser: wie sollen **die** Kinder) heißen?!" Eine Möglichkeit ist es, unter einem "General"-Ordner alle AVR-Projekte nach dem Entstehungsdatum (zum Beispiel das Datum von der Datei **`main.c`**) abzulegen.

Der Autor hat sich für das Tutorial entschlossen, keinen "General-Ordner" **AVR_Projekt_Dateien** zu verwenden, sondern die AVR-Projekte in den **Teilen** zu belassen, wo sie angesprochen und erklärt werden. Dabei wird für kleine Beispiel-Programme die Form

xxx_Programm_yyyyy

und für ganze Projekte die Form

xxx_Projekt_yyyyy

gewählt. **xxx** steht für die Nummer des **Teils**, in dem das Programm bzw. das Projekt erscheint und **yyyyy** steht für die Programm- bzw. Projekt-Kurz-Bezeichnung.

Entsprechend werden die selbst erstellten globalen Header-Dateien und die Modul-Dateien in den folgenden Unterverzeichnissen abgelegt:

505_Globale_Header_Dateien

505_AVR_Modul_Dateien

Natürlich lassen sich noch etliche Kriterien für die Ordner-Festlegung heranziehen, z.B. nach technischer Realisierung, nach Themen-/Anwendungsbereichen usw. usw. Eins ist aber bei allen Projekten identisch: In allen Projekt- bzw. Programm-Ordnern stehen vor der Kompilierung

- alle **`*.c`**-Dateien, mindestens jedoch das Hauptprogramm **`main.c`**,
- die selbst erzeugten globalen Header-Dateien **`*.h`**,
- und die benötigten Modul-Dateien, bestehend aus **`*.c`**- und **`*.h`**-Dateien.

Später werden bei der Kompilierung noch weitere Unterverzeichnisse und zahlreiche weitere Zwischen-Dateien angelegt (vergl. Abschnitt **2.2 Dateistruktur eines C-Projektes**). Letztere braucht man nicht unbedingt aufzubewahren und können gelöscht werden. Sie werden ja bei jedem Kompilierungslauf erneut erzeugt und werden deshalb auch nicht in den Datei-Ordnern mitgeliefert.