

**Anleitung
zur Verwendung der
Eclipse IDE**

mit AVR Prozessoren

Prof. Dr. G. Rücklé
Hochschule Darmstadt
Fb. Elektro- und Informationstechnik
v3

Kapitel 1

Kurze Einführung in Eclipse IDE

1.1 Die Systemumgebung: Linux mit Eclipse

Das Ihnen hier vorliegende Skript sollten Sie vor dem ersten Labortermin durchgearbeitet haben, da für alle Versuche ein paar Grundlagenkenntnisse im Betriebssystem UNIX/Linux auf dem Entwicklungsrechner sowie in der Bedienung des Zielsystems notwendige Voraussetzung für einen schnellen und reibungslosen Verlauf sind.

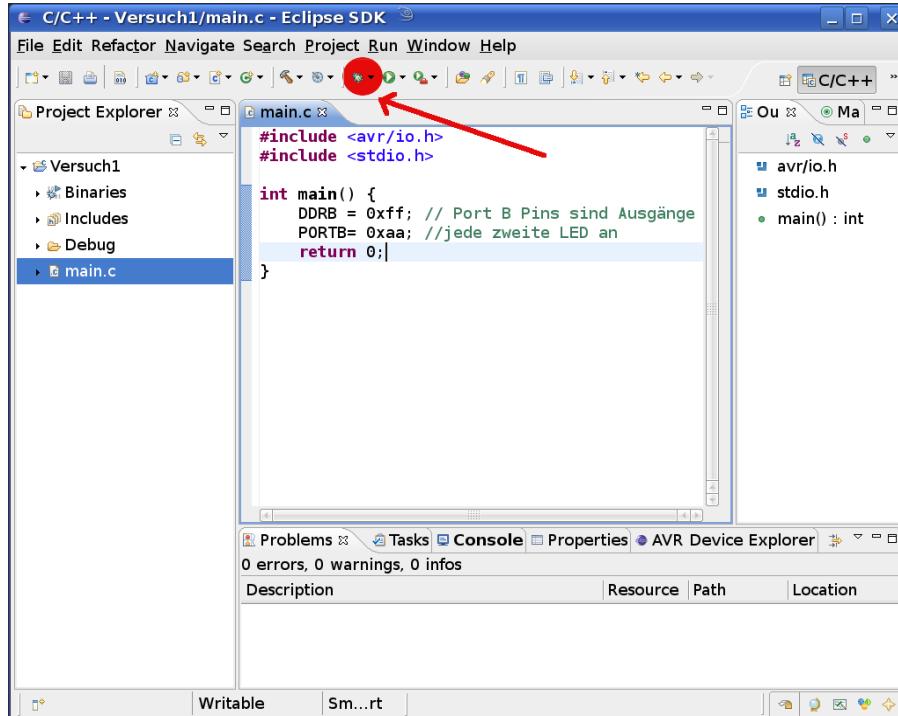
Die im Labor zur Verfügung stehende Umgebung setzt sich aus vernetzten Entwicklungsrechnern unter LINUX, der Entwicklungsumgebung ECLIPSE und dem eigentlichen Zielsystem mit dem μ C ATMEGA128 zusammen.

Nach dem Einschalten der Rechner wird automatisch das Linux Betriebssystem geladen und steht dann zur Verfügung. An den angeschlossenen Bildschirmen erscheint ein “login prompt” der zur Eingabe der Benutzerkennung und des Passwortes auffordert. Danach werden verschiedene Windows angezeigt. In den xterm-Windows bzw., Konsole-Windows können UNIX-Kommandos zur Bearbeitung der Aufgaben erfolgen. **Die Zugangsdaten (Benutzerkennung/Passwort Kombination) sind die gleichen, wie sie bei der Web-Anmeldung vergeben wurden!** Jeder Benutzer ist für diese Zugangsdaten auch im Sinne der Laborordnung verantwortlich und darf sie nicht weitergeben.

1.1.1 Workspace Einstellungen

Zur Sicherheit ist es sinnvoll, ein automatisches Abspeichern vor dem Übersetzen (Build) Ihres Projektes zu konfigurieren. Die entsprechende Stelle finden Sie unter

`Windows > Preferences > General >Workspace: Save automatically before build`



Ebenso ist es wichtig, die korrekte Einstellung für den Prozessortyp und die Taktfrequenz zu für Ihr Projekt finden. Gehen Sie mit der Maus im Projekt-Explorer über den Projektnamen und drücken die rechte Maustaste:

Properties > AVR Target Hardware

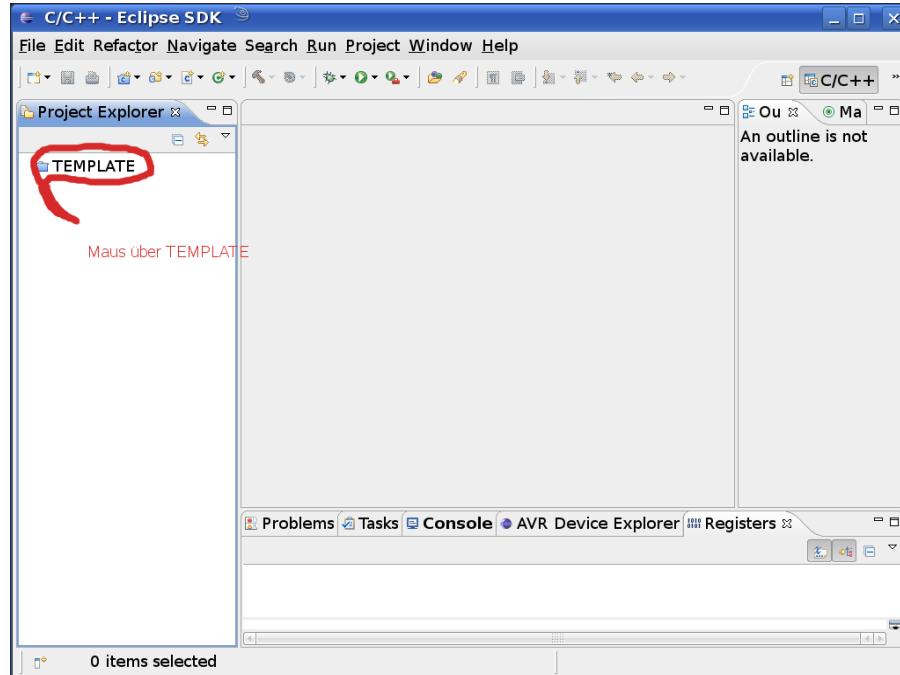
Sollten Sie den 'project explorer' nicht sehen, so können Sie diesen Über 'Window->Show View->Project Explorer' aktivieren.

1.1.2 Programmeingabe und Build-Prozess

Auf dem Entwicklungsrechner finden Sie nach dem Anmelden i.A. die Entwicklungsumgebung 'eclipse' vor. Wenn dort ein Projekt mit dem Namen 'TEMPLATE' existiert, so können Sie dieses als Vorlage nehmen und kopieren. Sollten Sie den 'project explorer' nicht sehen, so können Sie diesen Über 'Window->Show View->Project Explorer' aktivieren. Sind schon alle TEMPLATES für die Versuche (z.B. avr,av2,...) angelegt, so überspringen Sie die folgenden Bilder bis hin zum Anlegen einer neuen Quelldatei mit 'NEW'.

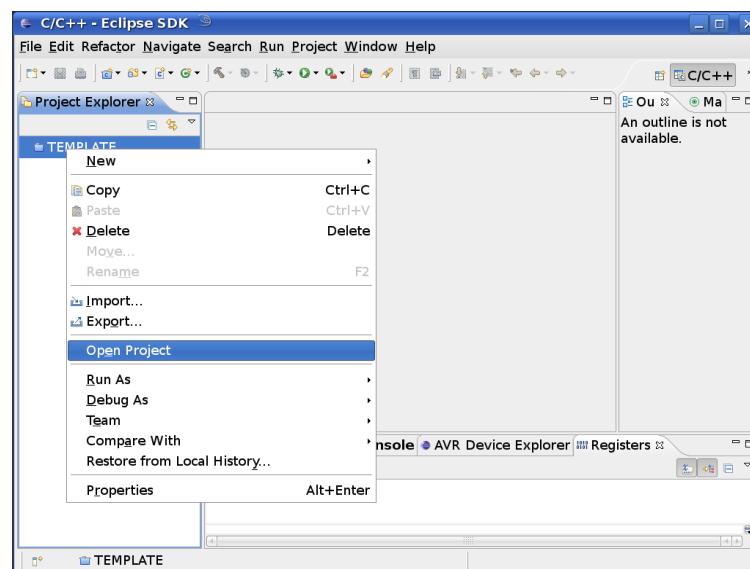
1.1.2.1 Projekt kopieren

Ausgehend von einem bereits existierenden Projekt (hier mit dem Namen 'TEMPLATE') wird beispielhaft der Kopierprozess dargestellt. Beachten Sie dabei, dass die DEBUG-Einstellungen **nicht** mit kopiert werden und jeweils immer wieder getrennt erfolgen müssen.

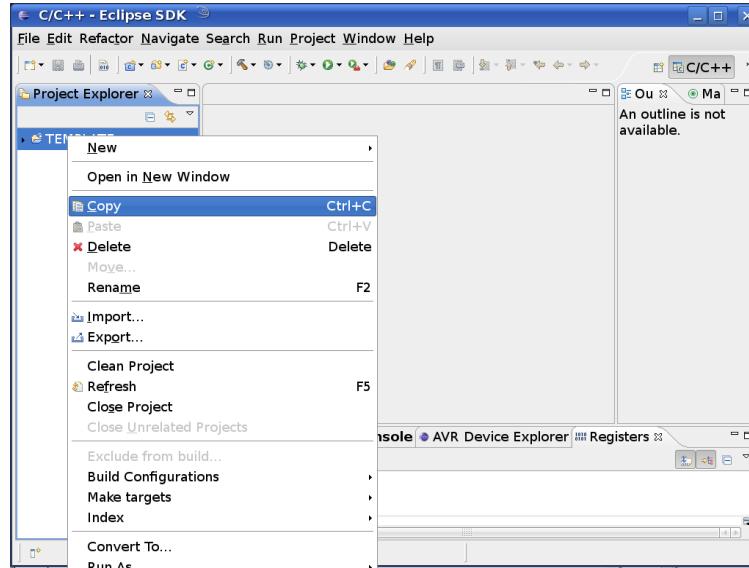


Markieren Sie dieses (TEMPLATE) Projekt und kopieren Sie es

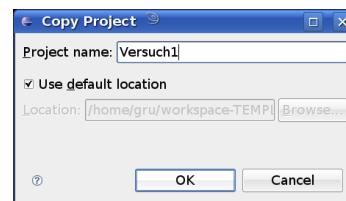
- mit Project->open project'



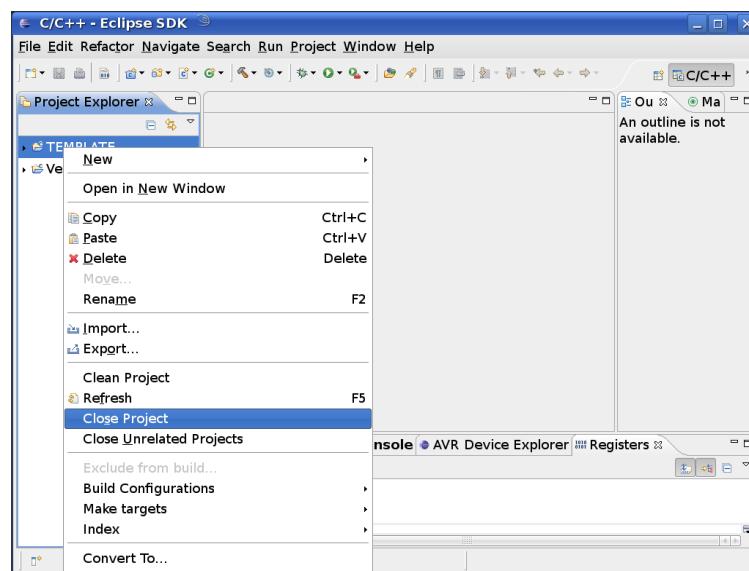
- rechte Maustaste → copy



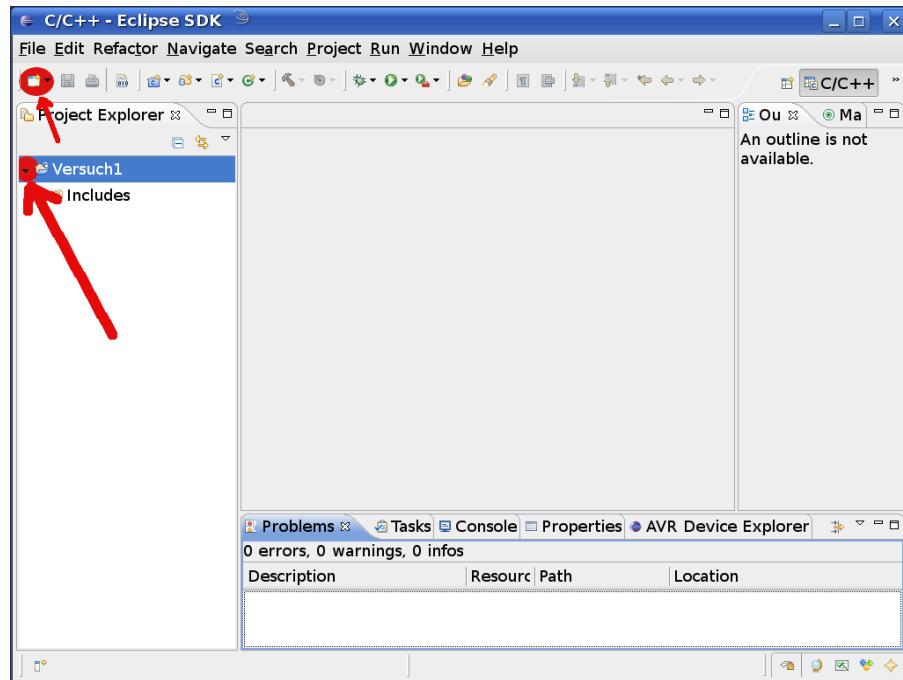
- rechte Maustaste → paste, legen Sie Ihren neuen Projektnamen fest:



- rechte Maustaste → close project um das alte (TEMPLATE) Projekt zu schliessen:



Durch anklicken des kleinen schwarzen Dreiecks vor dem neu kopierten Projekt oder einem schon existierenden wird der neu angelegte Projektinhalt angezeigt:

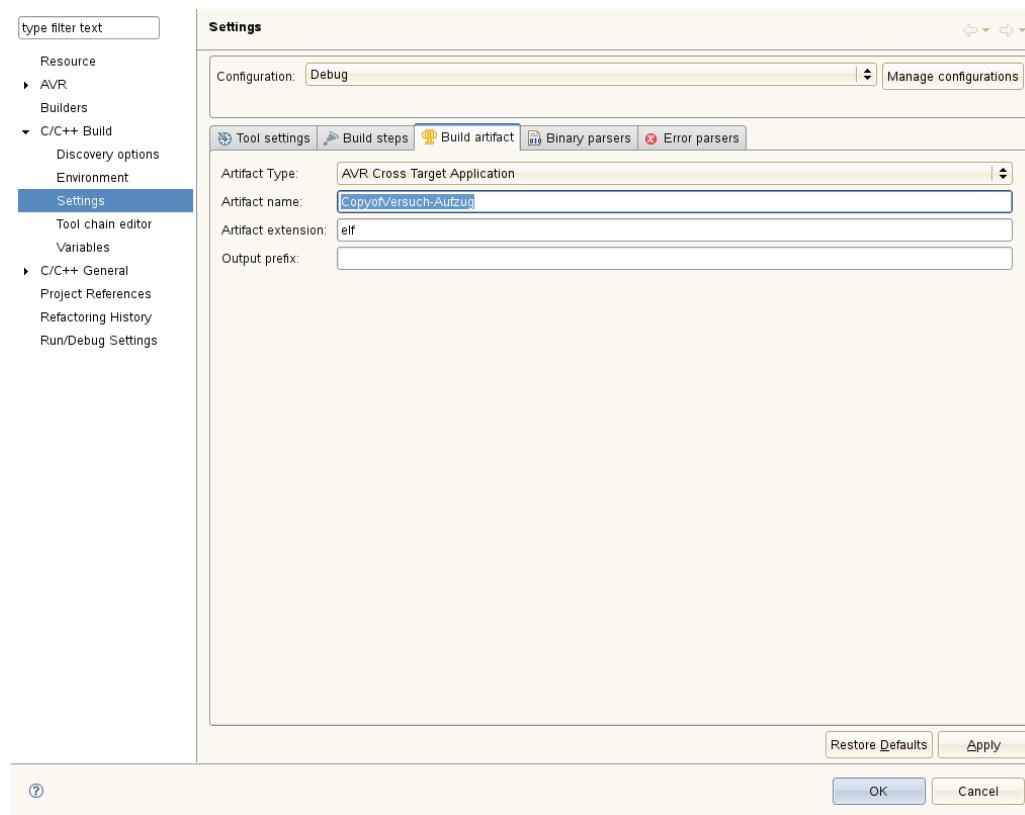


1.1.2.2 Programmname ändern

Falls Sie den Namen des ausführbaren Programmes, das durch den Eclipse Build erzeugt wird, ändern wollen, so können Sie dies unter

'Project > Properties: Settings : Build Artifact : Artifact Name'

tun. Beim Kopieren bzw. Umbenennen wird dieser nicht wieder nicht so angepasst, wie es Ihnen recht wäre. Es ist auch zweckmäßig in Dateinamen und auch Projektnamen **keine** Leerzeichen (Blanks) zu haben. Verschiedene Skripte, die im Hintergrund ablaufen, können damit ggf. nicht umgehen.

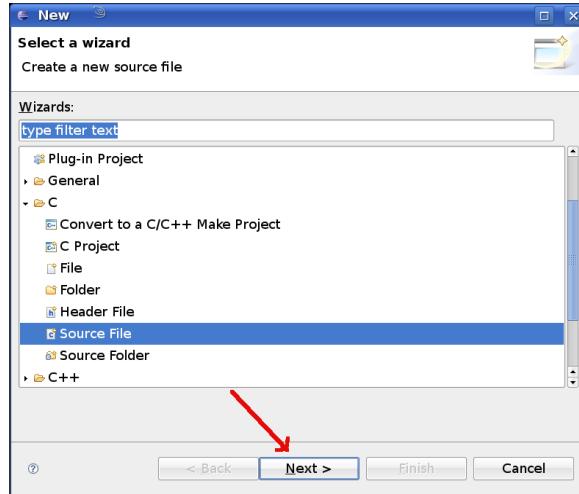


1.1.2.3 Neue Quelldatei anlegen

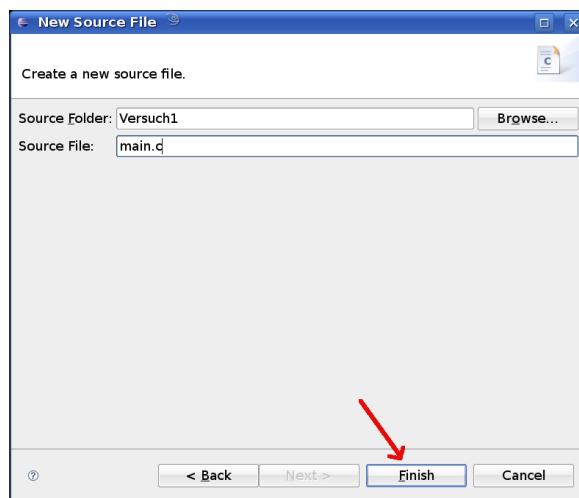
Legen Sie nun mit dem NEW-Wizard (der Knopf mit dem Icon 'New' in der zweiten Reihe links)

- NEW → Source File

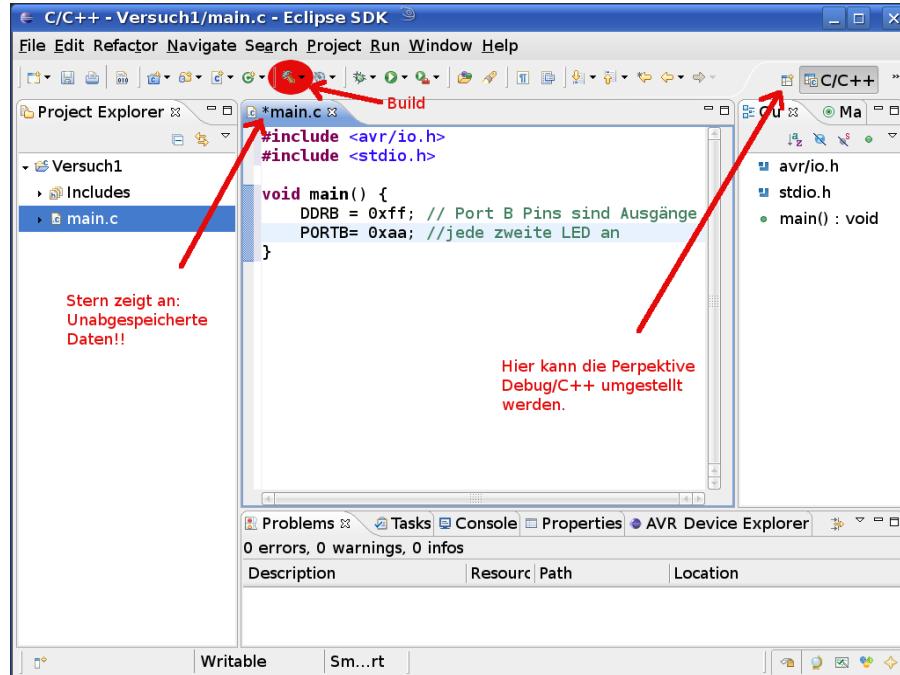
eine Neue Datei (z.B. 'main.c' an:



Geben Sie den Namen der neuen Datei an (Endung muss '.c' sein!):

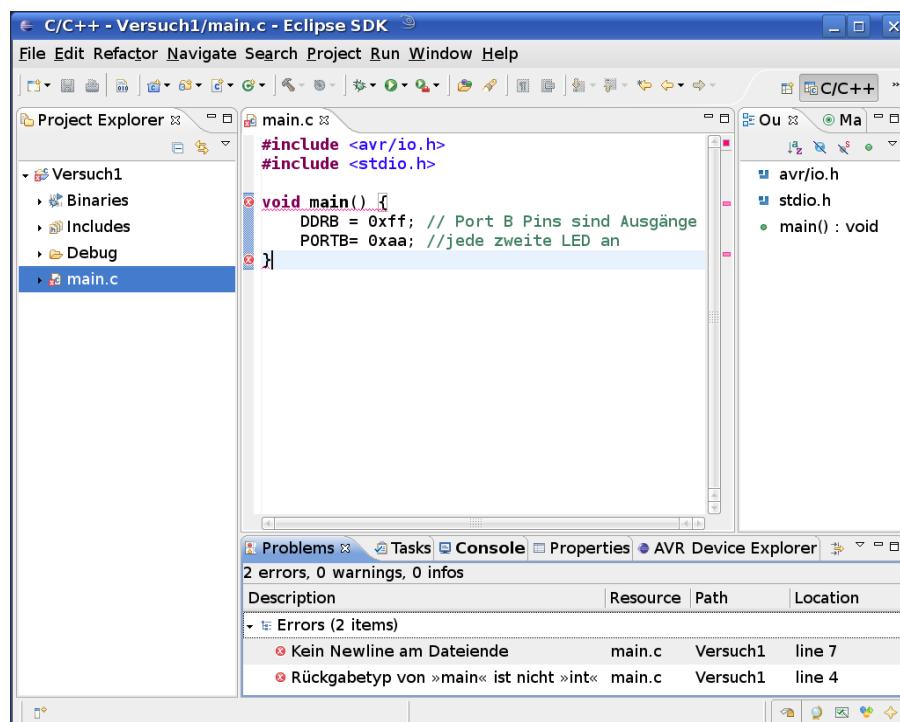


Damit öffnet sich ein Editor-Fenster für die Eingabe Ihres Programmes. Falls Sie mehrere Quell-dateien haben möchten, so können Sie diesen Schritt wiederholen.

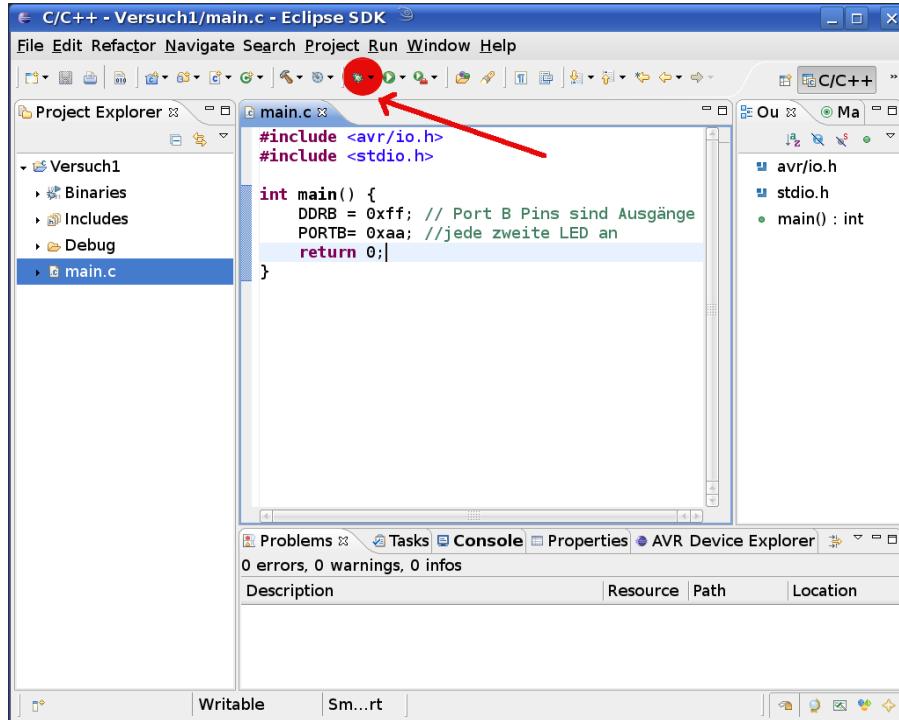


WICHTIG: Vergessen Sie nicht das Abspeichern Ihrer Änderungen, da sonst mit dem alten Quellcode getestet wird!

Falls Ihr eingegebenes Programm Fehler hat, so wird dies nach dem Build-Prozess durch ein kleines rotes Kreuz angezeigt:

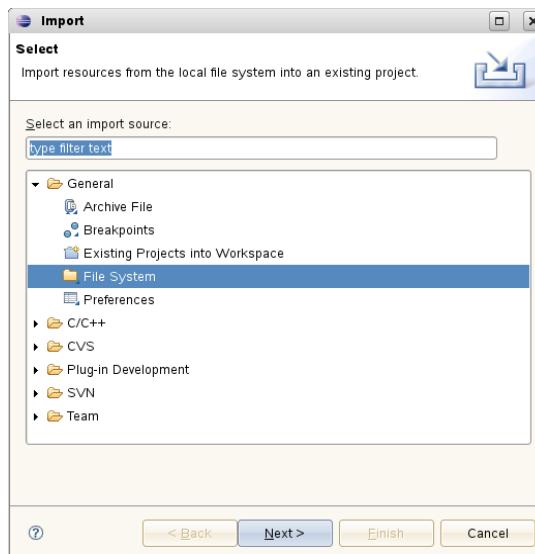


und auch nochmals im Fenster 'Problems' angezeigt. Nach erfolgter Korrektur und erneutem 'Build' ist das Programm ohne Fehler. Mit dem Knopf 'Debug Versuch1' kann dann der Testvorgang beginnen.

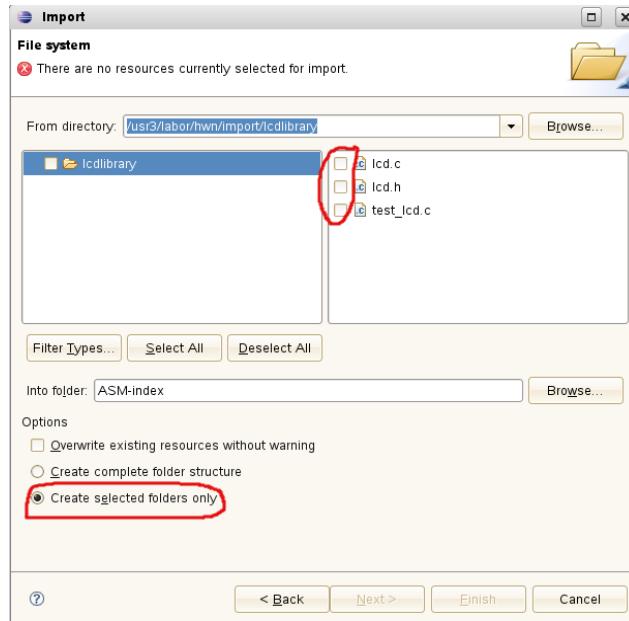


1.1.2.4 Dateien importieren

Über 'File>Import' kann (neben anderen Varianten) auch eine Liste von Dateien importiert werden. From-Folder auswählen:

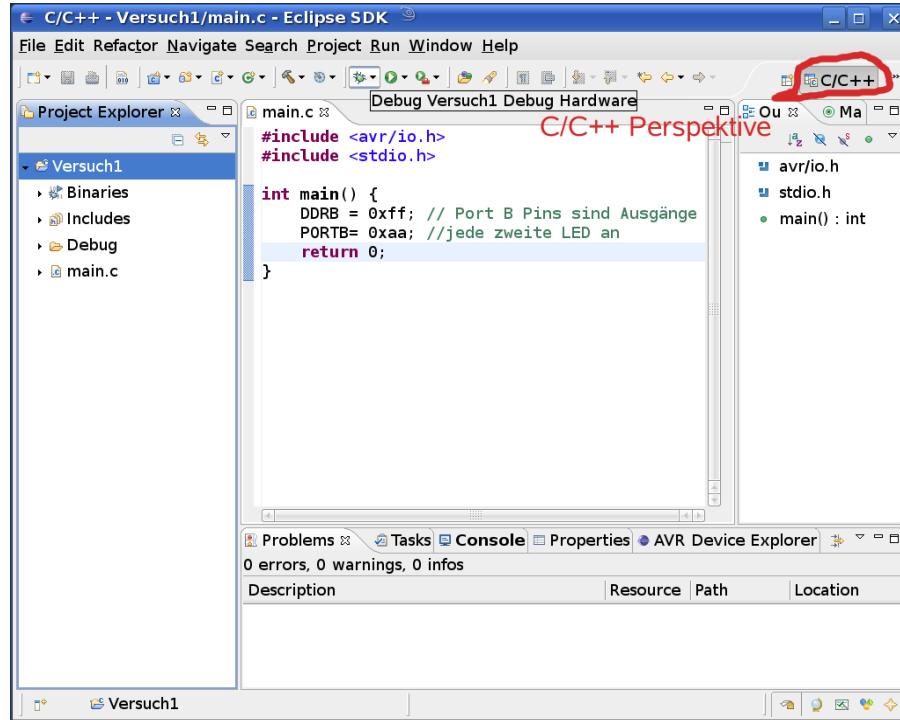


Wichtig: 'Create selected Folders only' setzen! Die gewünschten Dateien im gewählten From-Directory müssen dann entsprechend angehakt werden. Into-Folder entsprechend Ihres Projektes auswählen:

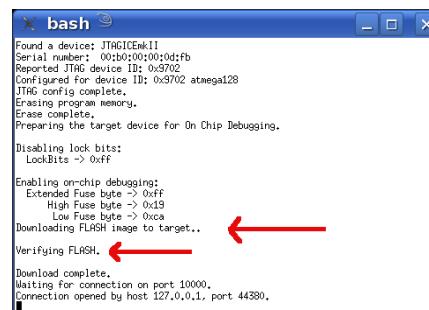


1.1.3 Testvorgang mit Eclipse

Mittels der Funktionstaste F11 kann auch der Debug Ihres Programmes gestartet werden. Für viele Menüpunkte gibt es auch Tastenkürzel. Starten wir also die Debug Sitzung mit dem Debug-Knopf:

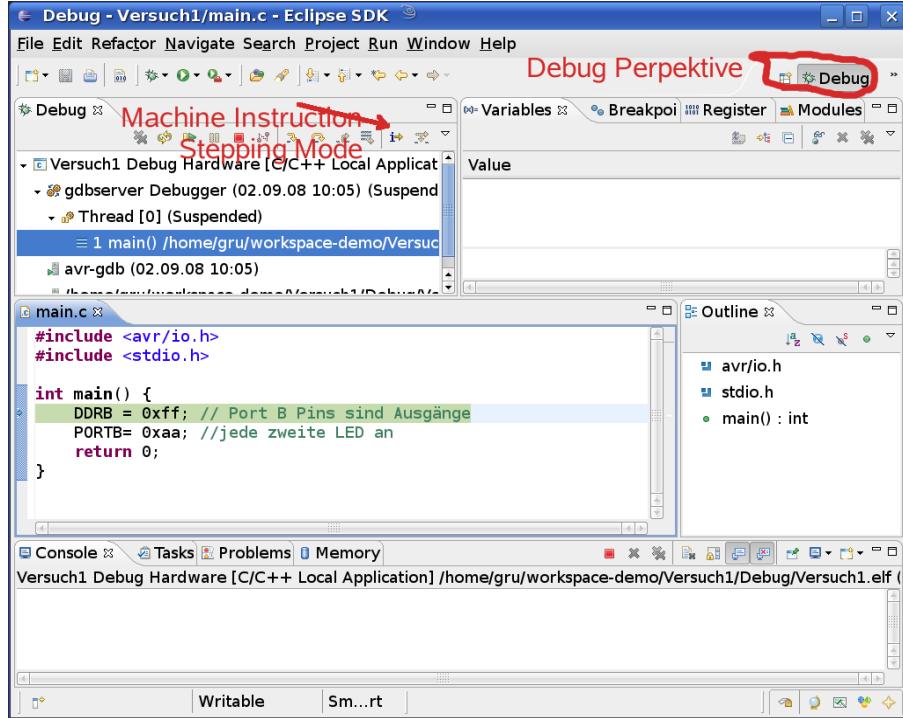


Zusätzlich zu dem Eclipse-Fenster wird jetzt noch ein weiteres Fenster sich öffnen, welches die Ausführung des Programmes 'avarice' anzeigt, dass den Kontakt zwischen AVR-Hardware und Eclipse herstellt:

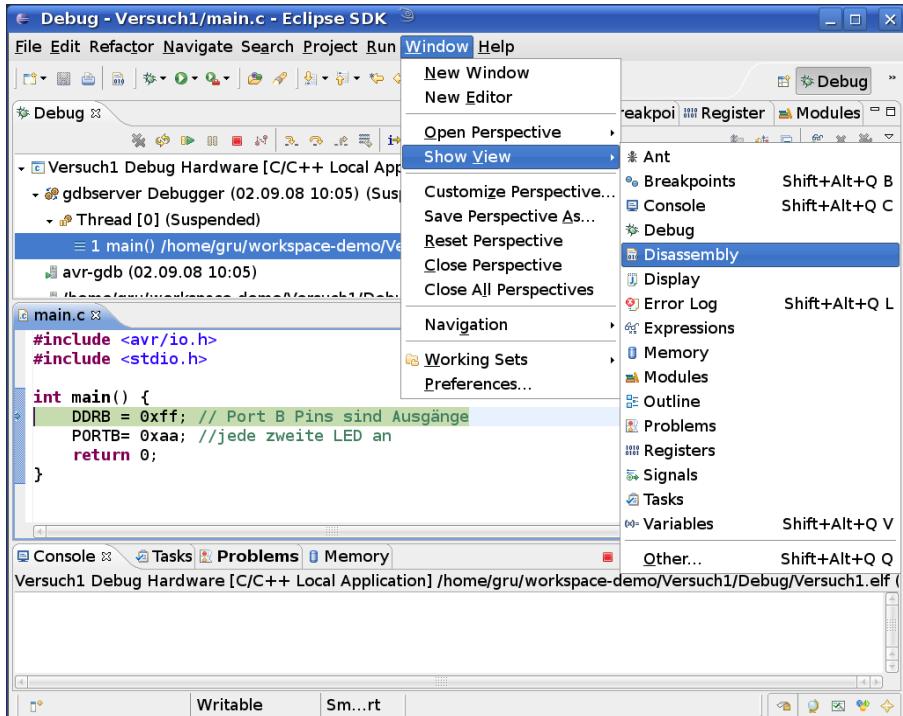


Wichtig dabei ist zu überprüfen, dass Ihr Programm korrekt im AVR-Entwicklungsboard angekommen ist. Dazu wird automatisch der Inhalt des Entwicklungsboardspeichers (Flash) gelöscht und mit dem neuen Programm überschrieben.

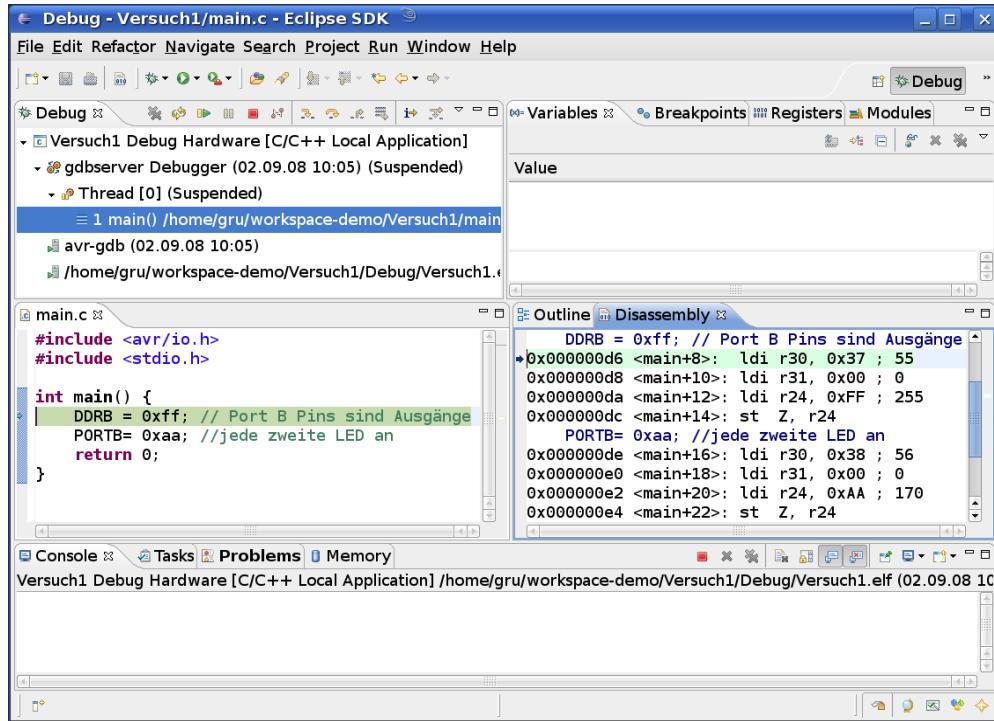
Ihr gestartetes Programm wird gleich zu Beginn am Anfang von Main angehalten:



Die Maschinenbefehle Ihres Programmes können Sie sich durch Aktivierung des 'Dissassembly'-Fensters ansehen:



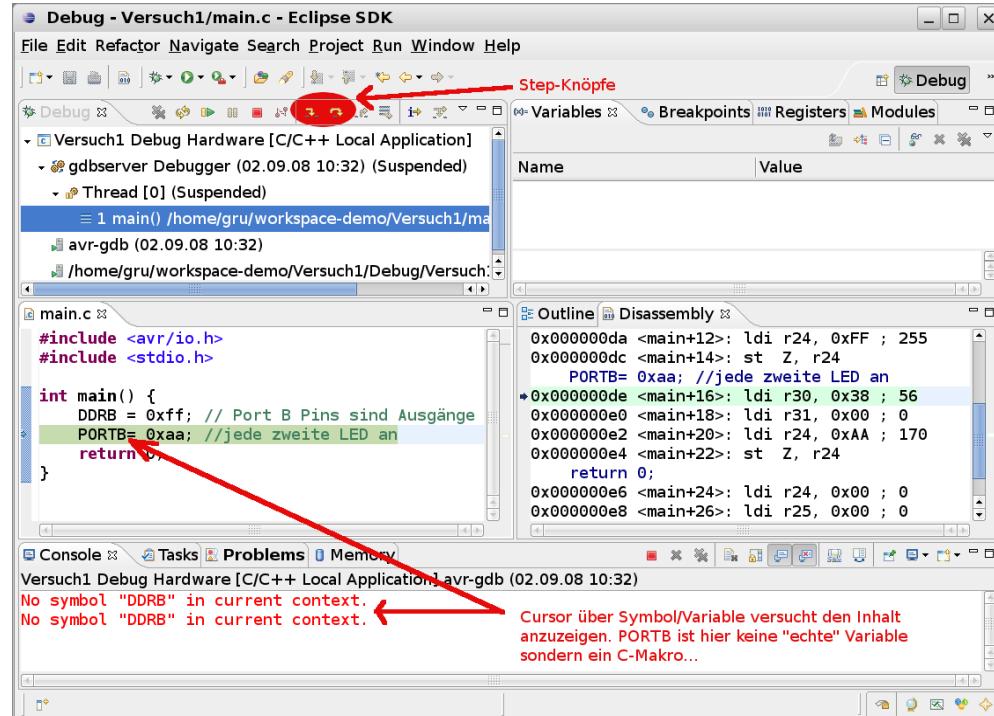
Auch hier zeigt ein kleiner Pfeil am Rand des Fensters an, bei welcher Programmzeile Sie sich gerade befinden.



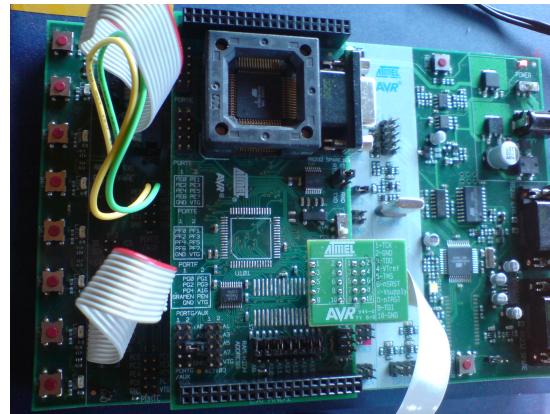
Über die im 'Debug'-Fenster befindlichen Knöpfe kann die Ausführung gesteuert werden:

- Resume (F8): ohne anzuhalten weiter
- Terminate (^F2): Programmlauf beenden
- Step Into(F5): Falls es ein UP Aufruf ist, wird in dieses UP verzweigt.
- Step Over (F6): Falls es ein UP Aufruf ist, wird erst nach dem Aufruf wieder angehalten.
- Step return (F7): Bis zum Ende des UP laufen.
- Instruction Stepping Mode: Auf Maschinenbefehlebene schrittweise ausführen.

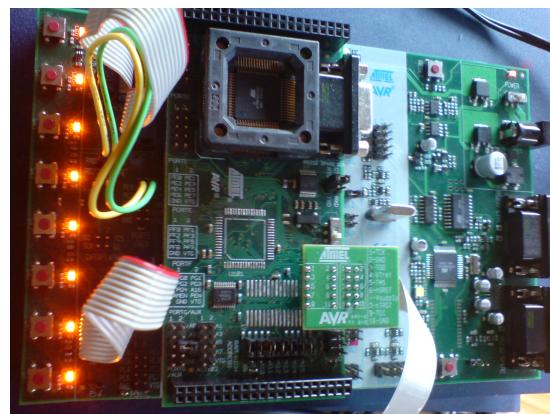
Anfänglich werden die Leuchtdioden an PORTB aus sein, da PORTB nach dem RESET im hochohmigen Zustand ist und die Leuchtdioden durch Pullups am Treiber (1 = aus, 0 = an) auf 1 gehalten werden.



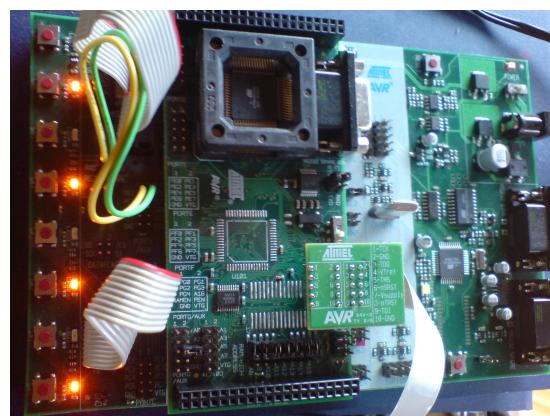
AVR-Board mit LED's aus:

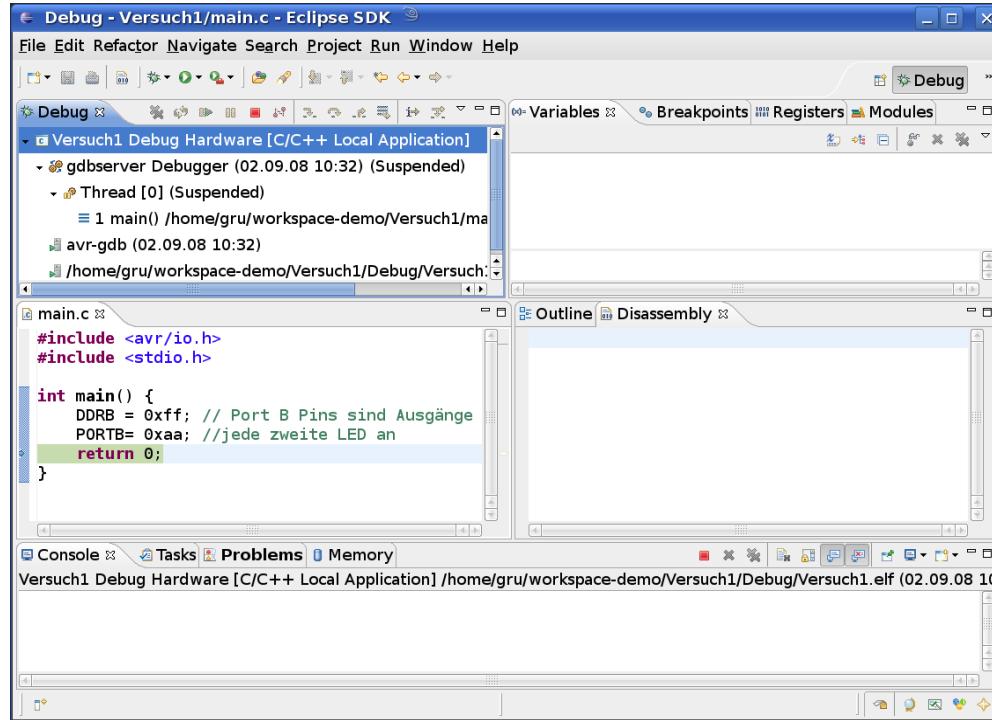


Nach der Zuweisung von 0xff and DDRB werden alle Pins von PORTB als Ausgänge konfiguriert.
Da nach dem RESET eine Null im PORTB-Register stand, sind jetzt alle LED's an:



Nach der Zuweisung von 0xaa an das PORTB Register ist dann nur noch jede zweite LED an:



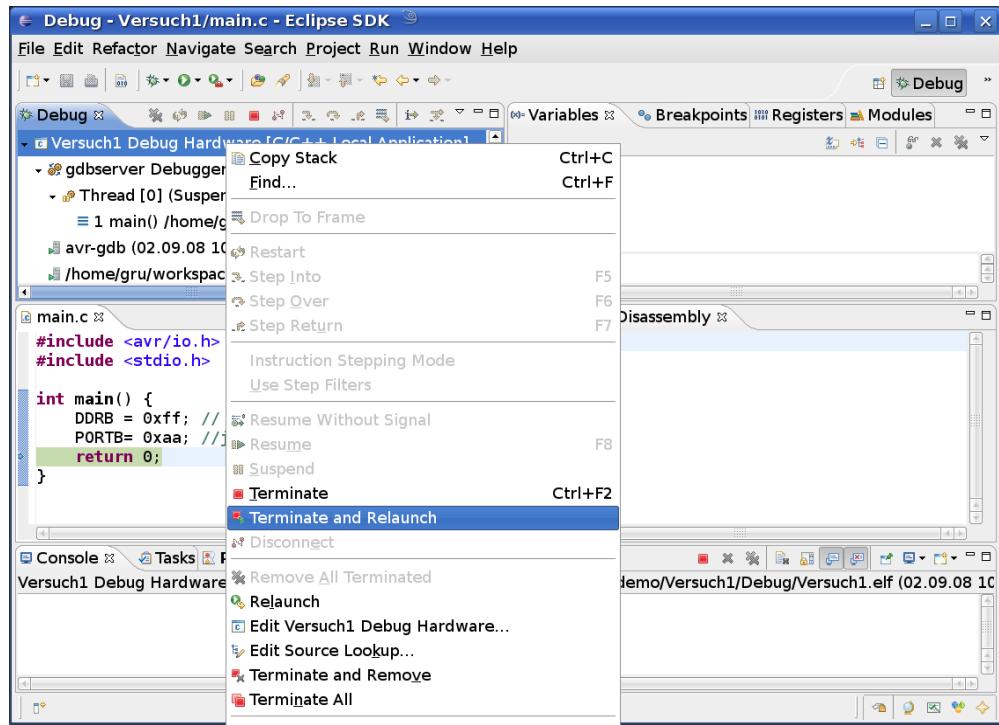


und das Programm ist beim 'return' in 'main' angelangt.

Nach Änderungen in Ihrem Programm sollte ein Neustart jetzt über das Kontextmenu (rechte Maustaste) im Debugger-Fenster erfolgen über

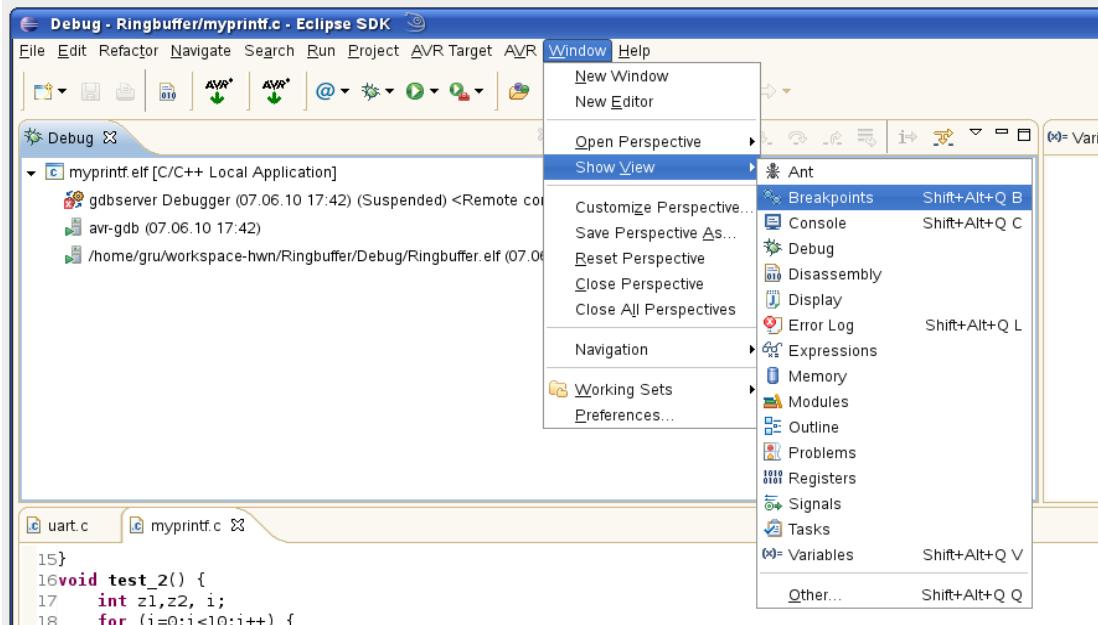
- Terminate and Relaunch

da ansonsten die alten Testläufe weiter in dem Debugger Fenster bleiben und wenigstens am Anfang für Vewrwirrung sorgen.



1.1.4 Weitere Debug Fenster

Es gibt in Eclipse noch eine Reihe von Fenstern, die je nach Situation von Interesse sein können:



Insbesondere die Fenster

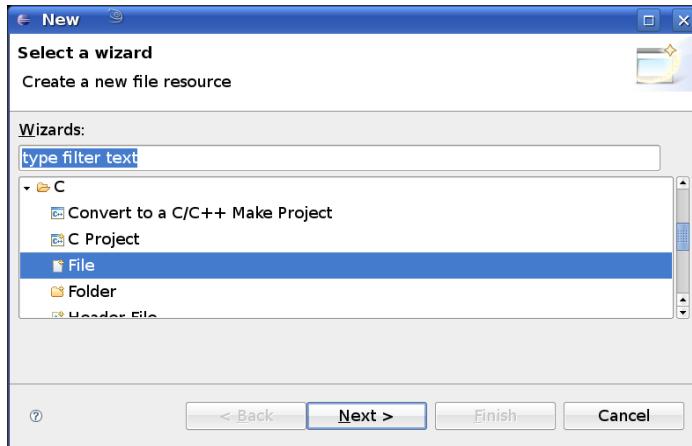
- Breakpoints

- Register
- Variables
- Disassembly

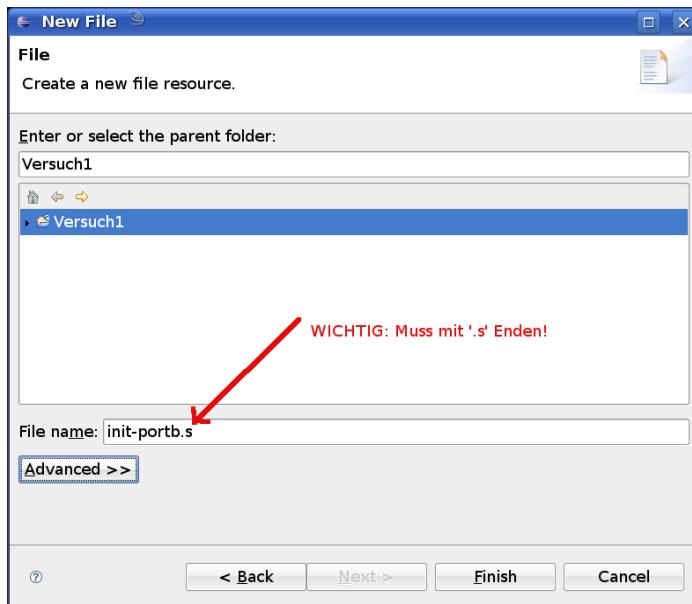
sind dabei von Interesse. Rechte Maustaste liefert die Kontextinformation in dem Fenster selbst dann zu diesem Fenster.

1.1.5 Einbinden von Assembler UP's

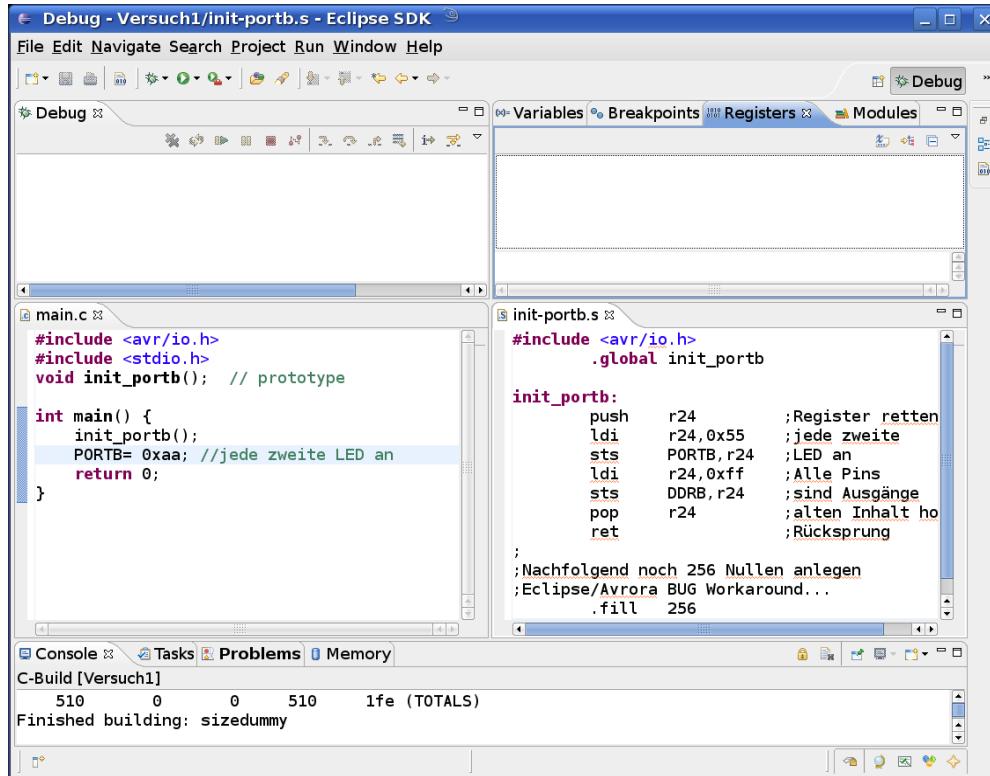
Das Einbinden von Assemblerunterprogrammen erfolgt genauso wie die Erzeugung des ersten oder einer weiteren C-Quelldatei. Es wird via 'new' ein entsprechender 'wizard' gestartet, der die benötigten Daten abfragt:



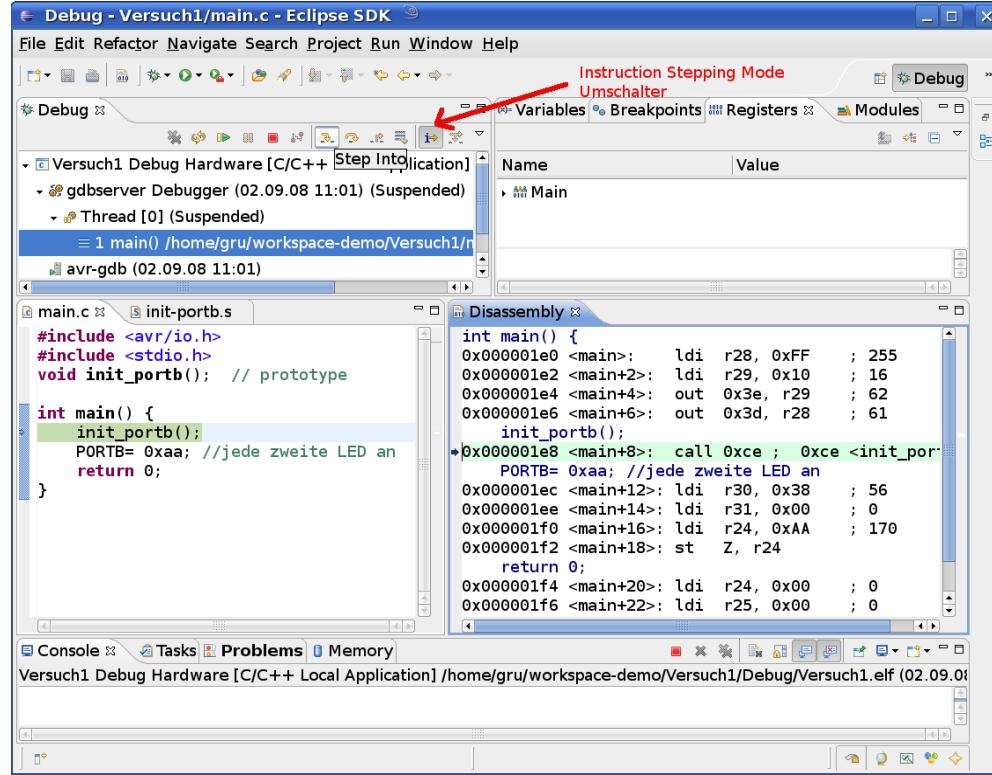
Wichtig ist, dass eine Assemblerquelldatei die Endung '.s' bekommt, da sie sonst nicht korrekt in den Build-Prozess mit einbezogen wird:



Auch bei Assemblerprogrammen wird über `#include <avr/io.h>` das Einbinden der Symbole für die Geräteregister des AVR-Controllers veranlasst:



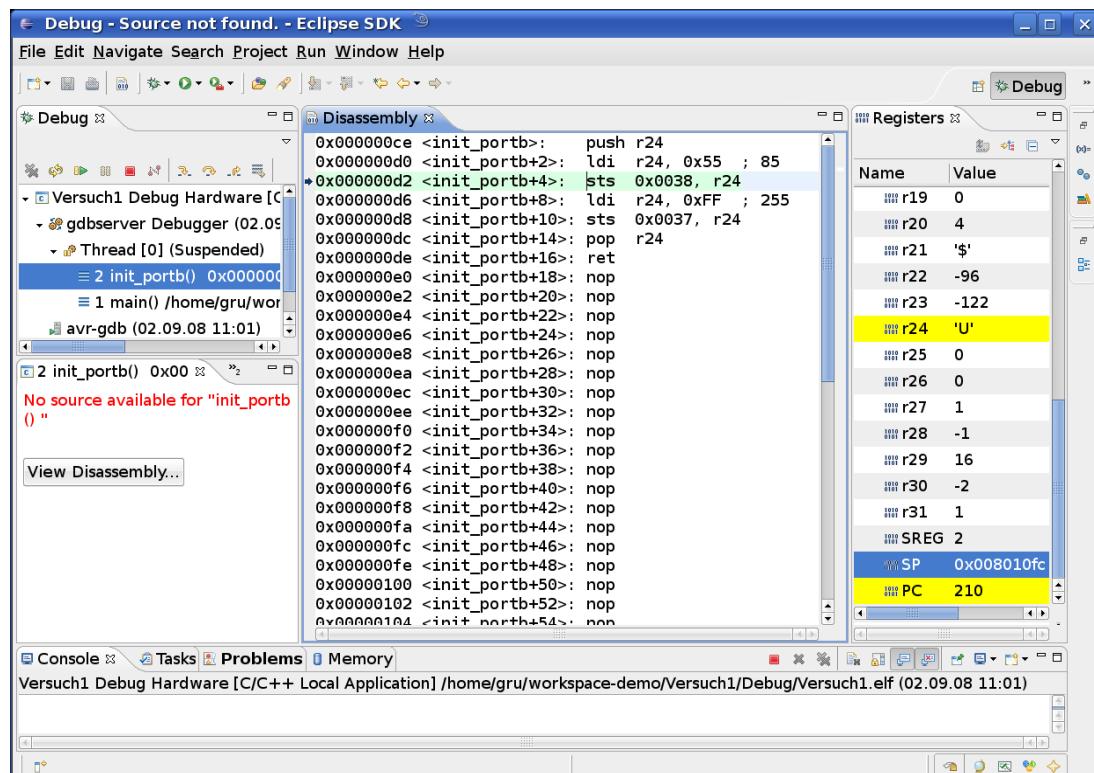
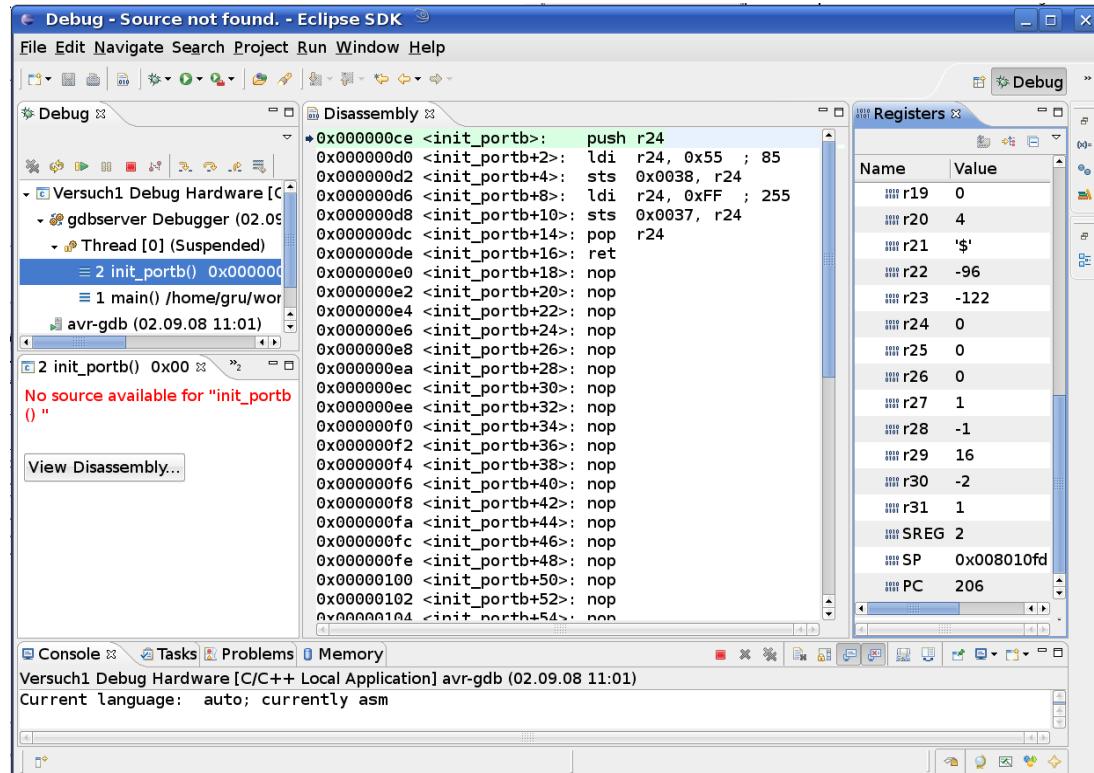
Wenn Sie nun von dem C-Programm aus in das Assemblerunterprogramm hinein wollen, ist es wichtig, dass Sie auf 'Instruction Stepping Mode' umstellen. Nur so gelangen Sie in das Assembler Unterprogramm:



Dann öffnet sich das Disassemblerfenster und Sie können das Registerfenster über

Window > Show View > Registers

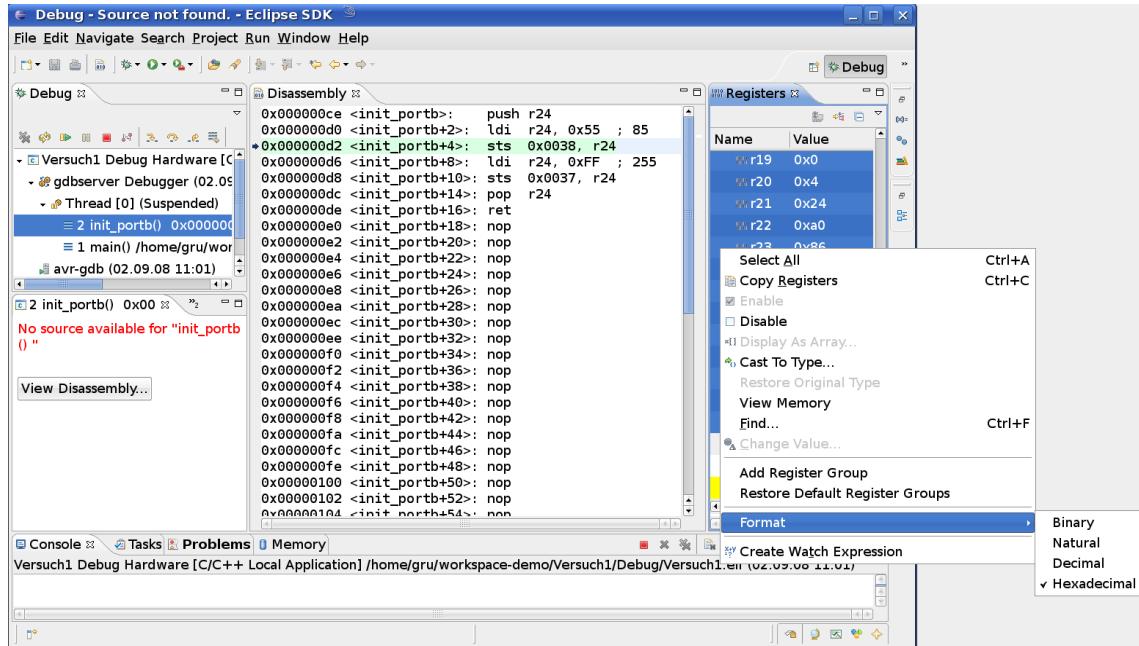
aktivieren. Schrittweise Ausführung markiert dabei auch die geänderten Register farbig.



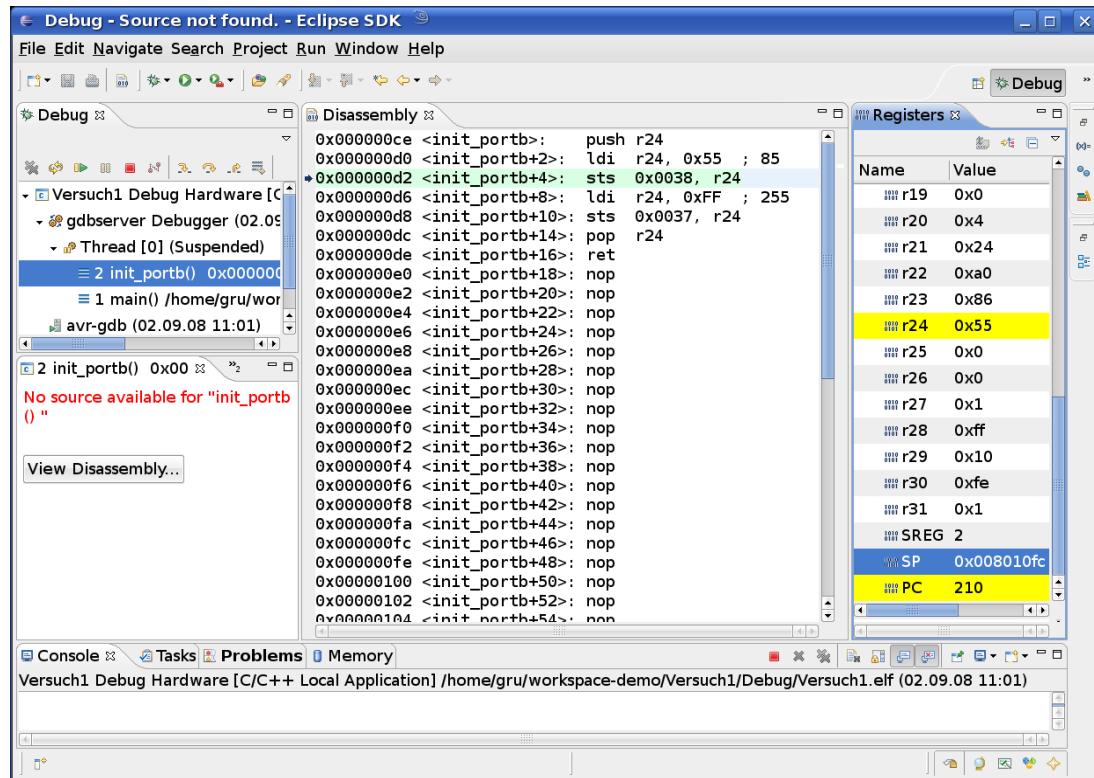
Falls Ihnen der Registerinhalt in hexadezimal Anzeige lieber ist, markieren Sie das erste Register und danach das letzte zu ändernde Register mit

- Shift-linke-Maustaste
- . Über das Kontextmenü
- rechte-Maustaste > Format

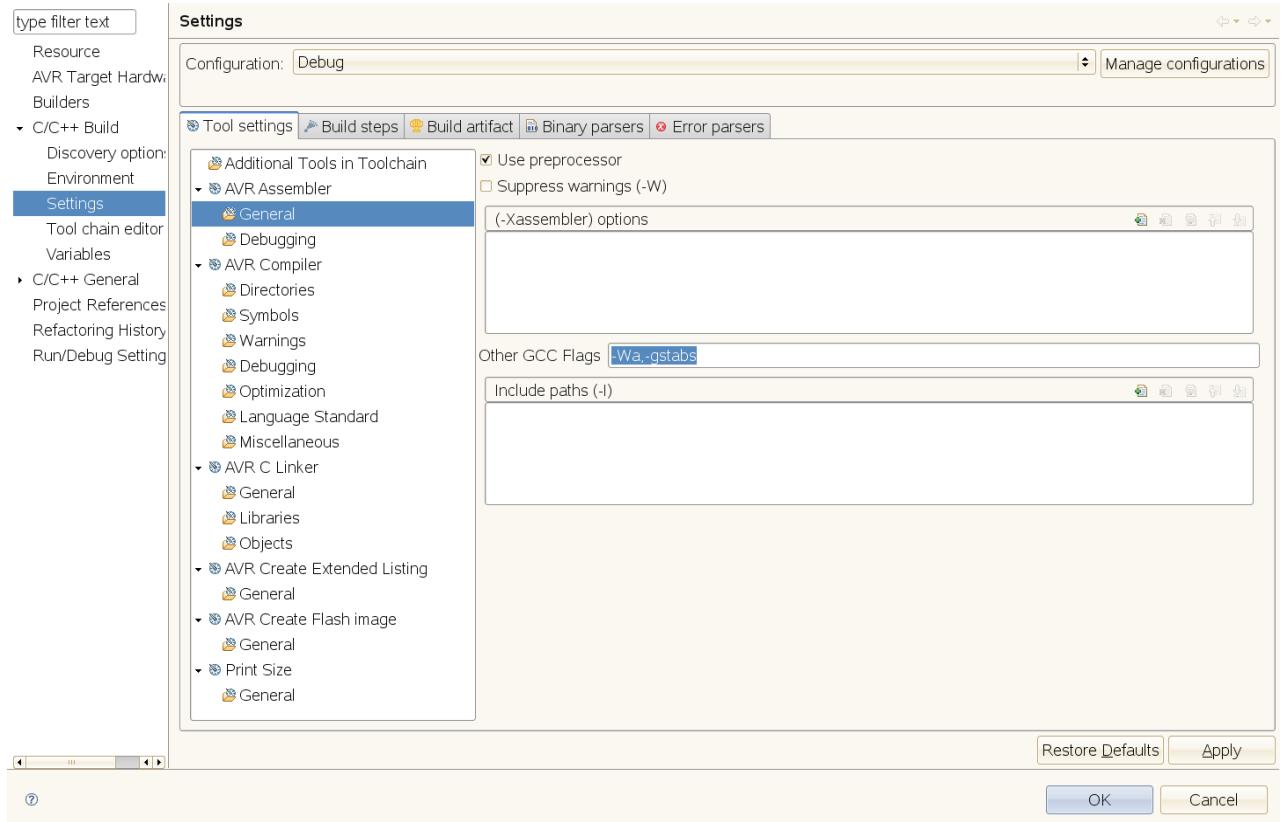
können Sie alle gewählten Register auf einmal anpassen:



und erhalten dann z.B. wie gewünscht die hexadezimale Ausgabe der Registerinhalte.



Um den Quelltext mit Zeilennummern in Assemblerprogrammen für den Debugprozess zur Verfügung zu haben, muss `-Wa,-gstabs` als Option für den avr-gcc eingetragen werden:



Dabei steht `-Wa` für eine Option, die nur der Assembler bekommen soll (gcc ruft den Assembler automatisch auf) und mit `-gstabs` wird festgelegt, dass die Symbole auch tatsächlich erzeugt und an den debugger weitergegeben werden.

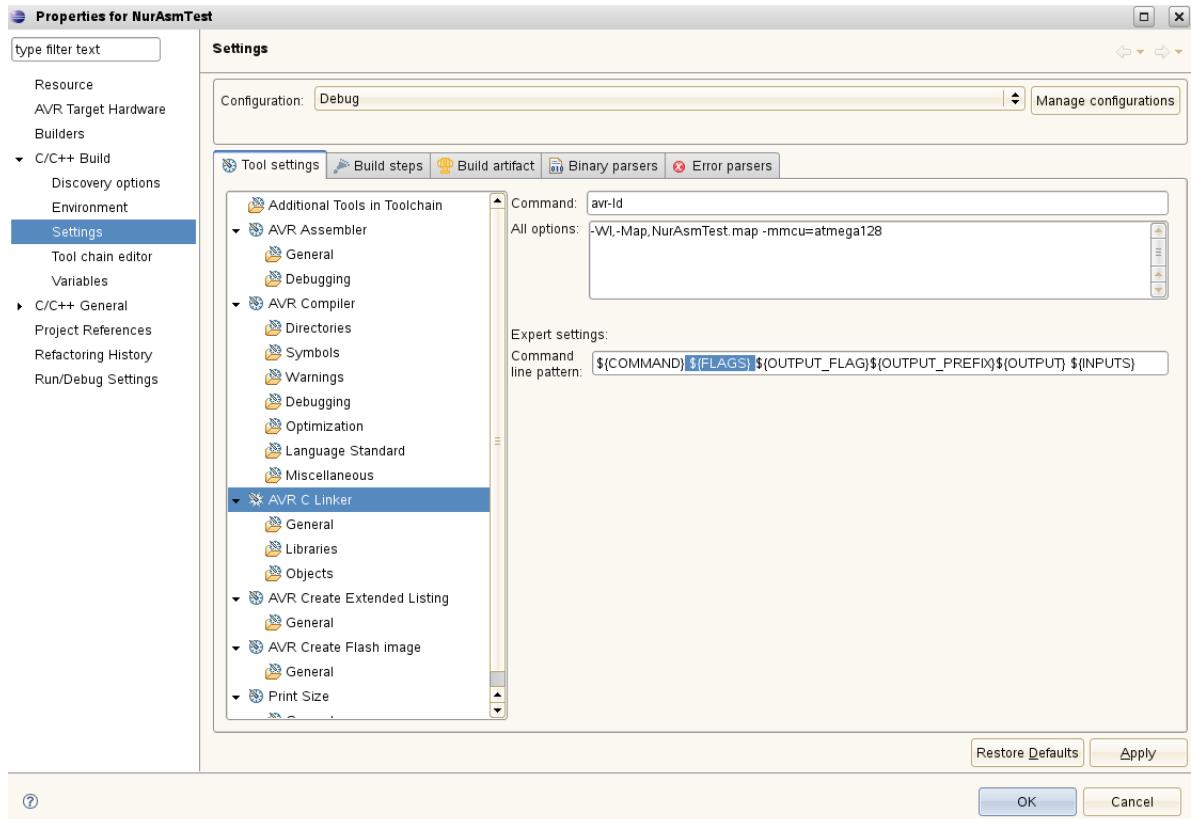
1.1.6 'Standalone' Assembler Programme

1.1.6.1 Ohne C Laufzeitsystem binden

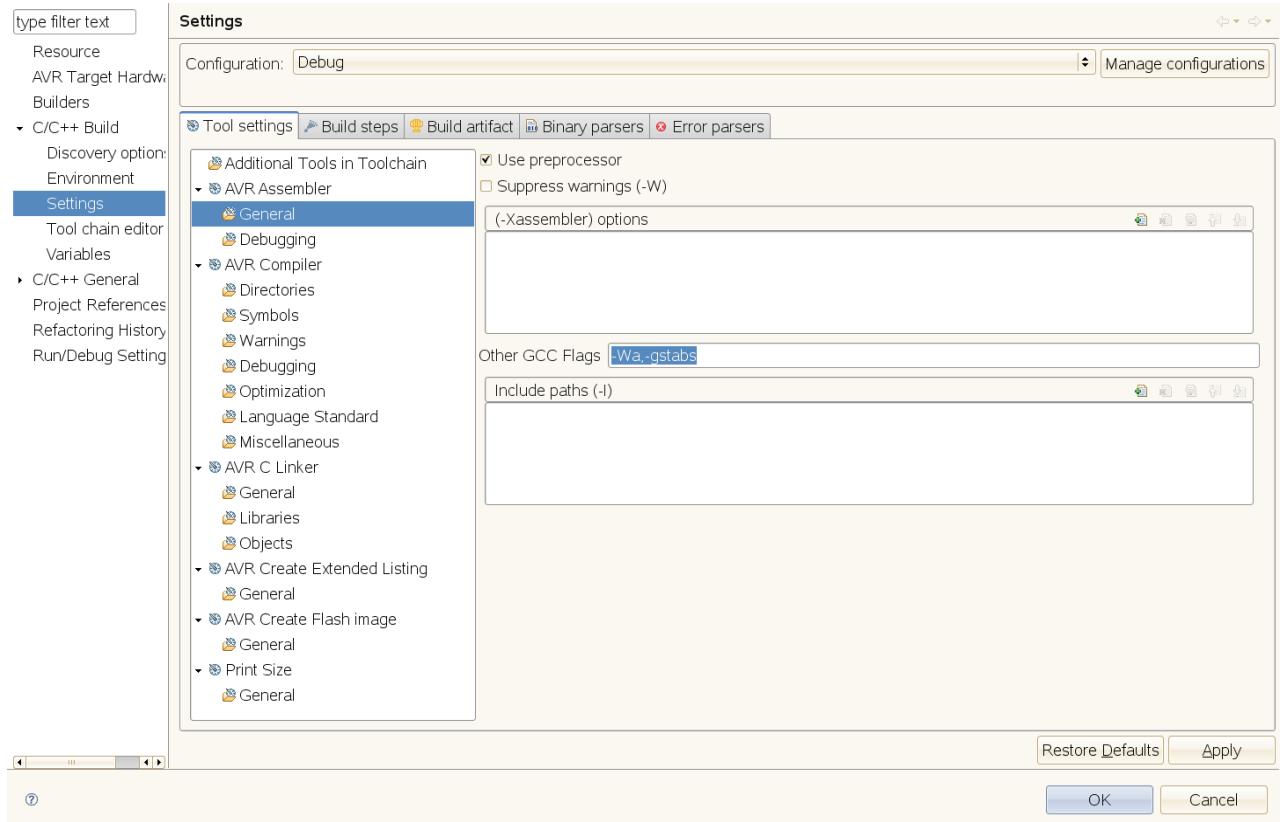
Die normale Vorgehensweise ist es, zusätzlich zu Ihrem Programm automatisch noch das C-Laufzeitsystem dazu zu binden (macht der avr-gcc). Das Laufzeitsystem wird als erstes und noch vor Ihrem `main()` die Kontrolle bekommen und div. Initialisierungen vornimmen. Auch die Interruptvektoren (bei der Adresse 0x0000 beginnend) werden dadurch gesetzt. Dies muss nun bei einem 'Nur-Assembler-Projekt' verhindert werden. Dazu muss folgende Einstellung in den Projektigenschaften vorgenommen werden:

- Unter 'C+-Build->Settings'+ als 'Command' den AVR-Linker 'avr-ld' eintragen und
- unter 'Expert Settings' den Eintrag von '`#{FLAGS}`' entfernen.

Damit wird also das C-Laufzeitsystem (inkl. der Vektortabelle bei Adresse 0x0000) nicht mehr zu dem Programm dazugebunden und Ihre Sektion .TEXT (für den Programmcode) beginnt bei Adresse 0x0000.



Um die Zeilenummern in Assemblerprogrammen erzeugen zu lassen, muss **-Wa, -gstabs** als Option für den avr-gcc eingetragen werden:



1.1.6.2 Codesektion mit fester Ladeadresse

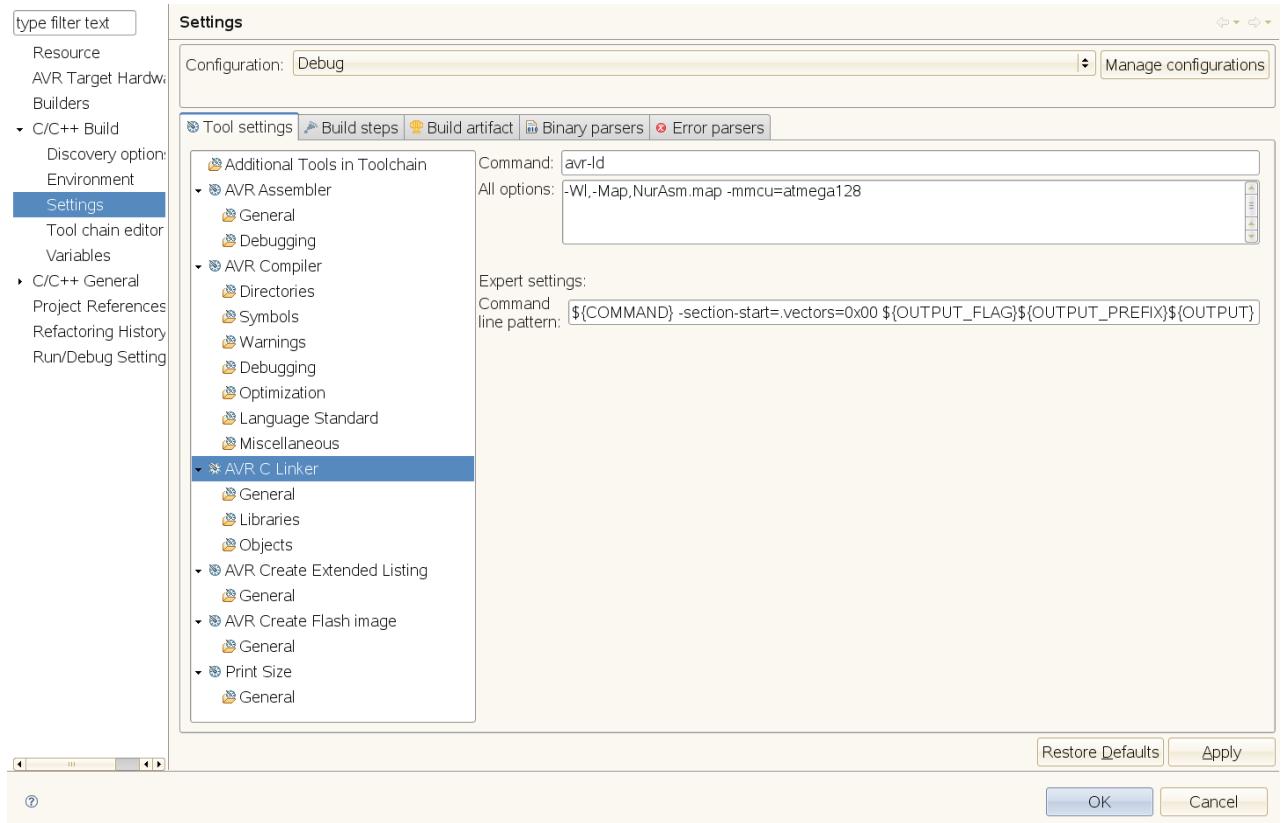
Normalerweise werden verschiedene Sektionen Ihres Programmes vom Linker (Binder) eigenständig an bestimmte Ladeadressen gelegt. Standardmäßig haben wir zunächst nur die Programmcode-sektion (.TEXT) und die Datensektion im RAM (.DATA). Falls wir nun eine eigene Ladeadresse haben wollen, um z.B. die Interruptvektoren selbst zu definieren, ist es notwendig, die Interruptvektoren sicher an die Flash-Adresse 0x0000 zu platzieren. Dies ist am einfachsten, indem man in dem Assembler Quellcode eine neue 'section' erzeugt, z.B. mit dem Namen '.vectors':

```
.section .vectors
    jmp main      ; reset vector
    jmp EXT_INT0 ; IRQ0 Handler
    jmp EXT_INT1 ; IRQ1 Handler
    jmp EXT_INT2 ; IRQ2 Handler
    jmp EXT_INT3 ; IRQ3 Handler
    ...
    jmp USART1_DRE; USART1,UDR Empty Handler
    jmp USART1_TXC; USART1 TX Complete Handler
    jmp TWI        ; Two-wire Serial Interface Interrupt
    jmp SPM_RDY   ; SPM Ready Handler
```

Wichtig ist, dass auch alle Vektoren vorhanden sind und damit die section '.vectors' auch die richtige Größe hat. Denn unmittelbar hinter dieser Sektion wird die '.text' section für den

Assemblercode beginnen. Dazu muss allerdings auch dem Linker gesagt werden, daß die reale Ladeadresse für '.vectors' die Adresse 0x0000 ist. Dies geschieht über die Option

`-section-start=.vectors=0x0000`



1.1.7 GCC: Spezielle Optionen

Zum initialen Setzen des Stackpointers wird folgende Angabe benötigt:

`-mstack=0x800400`

Dies setzt den Stackpointer beim AVR Prozessor beispielsweise auf 0x0400, was dem Ende einer 1 Kilobyte RAM entspricht. I.A. wird dies vom Compiler schon korrekt vorgenommen, wenn dieser den korrekten ATMEGAXXX Typ angegeben bekommt.

Achtung: Diese Option wird bei dem Eclipse AVR Plugin je nach Prozessorwahl auch schonmal explizit unter dem Punkt 'Miscellaneous' bei den Compiler-Settings angegeben, aber bei der Änderung des Prozessortype nicht wieder entfernt, so dass es hier zu falschen Einstellungen kommen kann. Erkennbar an entweder zu kleinem Stack oder aber an unkontrollierten Abstürzen des Programmes (typisch bei ersten 'RET' Befehl).

1.2 Bedienung des Zielsystems AVR

1.2.1 Starten des Entwicklungssystems

1.2.2 Starten des Zielsystems

Um das Zielsystem in Betrieb zu nehmen, muß

- die AVR-Entwicklungsplatine mit Strom (9V Netzteil) versorgt werden (Power LED, rot an?) und
- das Atmel JTAG-ICE per USB an den Arbeitsplatzrechner gekoppelt werden,
- per Netzteil mit Strom versorgt werden und zusätzlich
- der Anschluss 'RS232C Spare' per seriell Kabel mit dem Arbeitsplatzrechner verbunden sein.

1.2.3 Ausschalten des Entwicklungssystems

Um seine Sitzung zu beenden kann über einen Menueintrag "Exit" die Entwicklungsunggebung beendet werden. Die gesamte Sitzung kann mit Hilfe des Window-Managers (meist ein Button auf dem Bildschirm oder aber ein Kontext-Menu mit der Maustaste im Hintergrundbild erreichbar) ein Abmelden (leave) erfolgen.

1.3 UNIX Kurzeinführung

1.3.1 Kommando-Shell

Um mit einem Computer zu kommunizieren, ist eine definierte "Schnittstelle" zwischen Mensch und Maschine notwendig. Das Betriebssystem stellt (neben vielen anderen Funktionen) diese Schnittstelle in Form einer sogenannten Kommando-Shell ("Benutzeroberfläche") zur Verfügung. Dort werden Ihnen verschiedene Kommandos (z.B. Datei editieren, löschen, etc.) zur Verfügung gestellt mit denen Sie Ihre Daten (Dateien, Programme) verwalten können. Auch die Übertragung des assemblierten Programmes vom Entwicklungssystem zum Zielsystem wird mit Hilfe der Shell bzw. deren Kommandos und Hilfsprogrammen durchgeführt. Auch mit Hilfe des Programmes "konqueror" (ähnlich Windows Explorer) kann gearbeitet werden.

1.3.2 Die Verzeichnisstruktur

Das Betriebssystem UNIX verwaltet einen hierarchisch geordneten Verzeichnisbaum (Directory). Dieser beginnt mit dem Wurzelknoten (Stammverzeichnis, Root-Directory) und hat das Pfad-Trennzeichen '/' (forward slash). Windows verwendet z.B. den anderen Schrägstrich (backward slash).

In jedem Verzeichnis können Dateien erstellt werden (sofern man der Besitzer ist) und auch wiederum neue Verzeichnisse erzeugt werden. Anfanglich nach dem einloggen in das System sind sie im sog. Home-Directory (z.B. /usr/labor/mit/mit1). Dabei bezeichnet der erste Slash ("/") das Root-Directory und jeder durch Slash getrennte Namen ein Unterverzeichnis (Subdirectory). Einen solchen Weg im Verzeichnisbaum bezeichnet man auch als Pfad. Je nach Nomenklatur beinhaltet dieser Begriff Pfad gelegentlich auch den Dateinamen selbst und nicht nur die Liste der Directories.

1.3.3 Namenskonventionen

Wie vorher bereits erwähnt, bekommen Dateien (und Directories) einen Namen, der allerdings auch Sonderzeichen (z.B. einen Punkt) enthalten kann. Gross- und Kleinschreibung wird generell unterschieden!

1.4 Die einfachen Kommandos

1.4.1 Syntax

Jedes Kommando setzt sich aus mehreren Teilen zusammen:

1. Kommandoname (z.B. "ls"),
2. Flags angezeigt durch einen Bindestrich (z.B. -l),
3. Liste von Parametern (z.B. Dateinamen)
4. Ein-/Ausgabeumlenkung (z.B. >ausgabe.dat)

Beispielsweise besteht

`ls -l a*`

aus dem Kommandonamen ("ls"), der die Shell veranlaßt, ein Programm namens "/bin/ls" zu suchen, aus dem Flag "-l", das das Kommando zu einer langen Ausgabe veranlasst und dem Parameter "a*" der sämtlich Dateien beschreibt, die mit einem "a" anfangen. Als Ergebnis sehen Sie eine Liste aller Dateien mit ihren Attributen (alias dazu ist 'dir').

1.4.2 Kommandobearbeitung

Die Shell bietet Ihnen an mit Cursor-Tasten die Kommandoeingabe zu bearbeiten. Für die Beeinflussung von Programmen gibt es folgende Steuerzeichen:

- ^ C: Laufendes Programm unterbrechen
- ^ D: Dateiende EOF auf der Tastatur simulieren.

1.4.3 Umleitung

Es gibt folgende Möglichkeiten zur Ein- Ausgabeumlenkung:

<	Umleitung Standard-Eingabe
>	Umleitung Standard-Ausgabe
2>	Umleitung Fehler-Ausgabe
	Ausgabe 1. Kdo. als Eingabe für 2. Kdo. (Pipe)

Jedes gestartete Programm bekommt direkt drei verschiedene Kommunikationsströme zugewiesen, die das Programm direkt über Betriebssystemfunktionen verwenden kann. Ohne Angabe des entsprechenden Modifiers ist die Standardeingabe die Bildschirmtastatur und die beiden anderen Ströme sind auf die Terminalausgabe gelegt. Wenn dies geändert werden soll, so reicht es, auf der Shell-Ebene beim Aufruf des entsprechenden Programmes (Kommandos) die Modifier mit dem Umlenkungsziel bzw. der Umlenkungsquelle anzugeben. Z.B. wird das Kommando

`ls -l > liste`

eine Liste des Directory-Inhaltes erstellen und diese nicht wie sonst auf das Terminal schicken sondern es wird eine Datei "liste" im aktuellen Work-Directory angelegt und die Liste dort hineingeschrieben. Das Kommando

```
ls -l | lpr
```

wird die Liste des Directory-Inhaltes durch "ls -l" erstellt und die Ausgabe als Eingabe des Kommandos "lp" verwendet, das dann diese Liste in die Druckerwarteschlange zum Ausdrucken einreicht. Mit

```
lpq  
lprm 123
```

kann dann der Zustand der Druckerwarteschlange angezeigt werden bzw. der Druckauftrag 123 in der Warteschlange geloescht werden. Falls allerdings der Printout schon im Drucker ist, kann nur noch über die Druckerfunktion 'Cancel' oder 'Job abbrechen' Ä o.ä. ein Abbruch versucht werden.

1.5 Einschränkungen beim Programmieren

Es wird empfohlen für jeden Versuch ein eigenes Directory anzulegen. Die Quelldateien sollten mit ".s" (Assembler), ".c" (C Quellen) bzw. ".cpp" (C++), ".h" oder ".H" enden, denn **nur diese** werden gegen **Plattenausfälle** gesichert!

1.5.1 Das Zielsystem

Beachten Sie, daß durch das/die TEMPLATE(s) bestimmte Einstellungen vorgenommen wurden, die Sie selbst vornehmen müssen, falls Sie ein Projekt nicht kopieren sondern ganz neu anlegen (s.a. Anhang).

Den Aufbau des Zielsystems können Sie den im Web befindlichen Dokumenten entnehmen.

1.6 Bearbeitung von Programmen

1.6.0.1 Workspace von eclipse, Directory

Für jeden Laborversuch sollten Sie sich ein eigenes Projekt anlegen bzw. die schon angelegten TEMPLATES (Versuch1, Versuch2, etc.) verwenden und dort Ihre einzelnen Programme (als Quelltext) ablegen.

1.6.1 Nur ein TEMPLATE vorhanden

Kopieren Sie sich also das (einige) TEMPLATE in Ihren aktuellen Arbeitsbereich. Dieser befindet sich unter Ihrem Home-Directory mit dem Namen Workspace. Alle Projekte sind dort in weiteren Unterverzeichnissen angelegt und könnten auch ausserhalb von eclipse eingesehen und bearbeitet werden

1.6.2 Die C Compilierung mit eclipse

Ein editiertes C Programm muß nun von C Code in maschinenlesebares Format umgesetzt werden. Dazu rufen Sie in Ihrem Projekt den Punkt 'Build' oder 'Clean' auf. Damit wird von eclipse mittels des automatisch generierten Makefiles und des Programmes 'make' das Maschinenprogramm erzeugt. Der Makefile als Basis von 'make' enthält die einzelnen Schritte zur Erzeugung des Maschinenprogrammes und die Abhängigkeiten der Quelldateien untereinander. Dieses können Sie auch unabhängig von eclipse im entsprechenden Projektunterverzeichniss aufrufen.

1.6.3 Die Assemblierung

Selbst geschriebene Assemblerprogramme werden ebenfalls von make korrekt erkannt (Dateiendung '.s') und werden direkt von dem AVR Assembler und nicht vom C-Compiler bearbeitet und werden automatisch zu Ihrem restlichen Programm dazu gebunden (Linker). Auch hier könnten Sie das make verwenden oder aber auch mit dem C-Compiler direkt assemblieren, denn dieser erkennt an der Endung '.s', dass es sich um eine Assemblerprogramm handelt.

An Pseudobefehlen und Direktiven stehen bei dem hier verwendeten Assembler folgende zur Verfügung:

- Numerische Konstanten und Ausdrücke folgen der C Syntax (Prefix 0x für hexadezimal z.B.)
- .byte: allokiert ein einzelnes bzw. eine Liste Bytes
- .ascii: allokiert (unterminiert!) einen ASCII-String
- .asciz: allokiert Null-terminiert einen ASCII-String (C-String)
- .fill *n*: erzeugt *n* Bytes Nullen
- .data: wechselt in die Daten-Sektion
- .text: wechselt in die Programm-Sektion (Maschinencode und ROM Konstante)
- .set/.equ: (oder '=') erzeugt ein Symbol mit einem Konstanten Wert.
- .global (.globl): deklariert ein über die Modulgrenze hinaus sichtbares Symbol für den Linker (z.B. externes UP)
- .extern: deklariert ein Symbol als ausserhalb des Moduls befindlich.
- lo8(): nimmt die unteren 8 Bit einer 16 Bit Zahl
- hi8(): nimmt die oberen 8 Bit einer 16 Bit Zahl
- pm(): nimmt eine Maschinencodeadresse und konvertiert diese in eine RAM address. (div 2, da ROM 16 Bit organisiert)

Beispielrahmen für ein Assemblerprogramm:

```
#INCLUDE <avr/io.h>
.global mysub
mysub:
    ret
```

Anfänglich ist die Sektion 'text' aktiviert, wo die Programmdaten (Flash) landen sollen.

1.6.4 Arbeiten mit 'Releases'

Unter eclipse kann man verschiedene 'Releases' anlegen, d.h. verschiedene Kombinationen von Quelldateien in den Generierungsprozess (build) einbeziehen. Dazu ist der Menüpunkt 'Build Configurations' im Projektmenü (rechte Maustaste über dem Projektnamen im Baum) gedacht. Über 'Manage' lässt sich eine neue Konfiguration anlegen. Verwenden Sie dazu die Kopieroption, die eine vorhandene Konfiguration dupliziert.

Jede Quelldatei lässt sich über die Menüoption 'Exclude from Build' in ihrer Zugehörigkeit zu den einzelnen Konfigurationen einstellen. Der Menüname 'Exclude from build' ist schlecht gewählt, denn man kann mit diesem Menüpunkt auch zu einem 'build' eine Datei hinzufügen! Besser wäre 'Manage build configuration for file'.

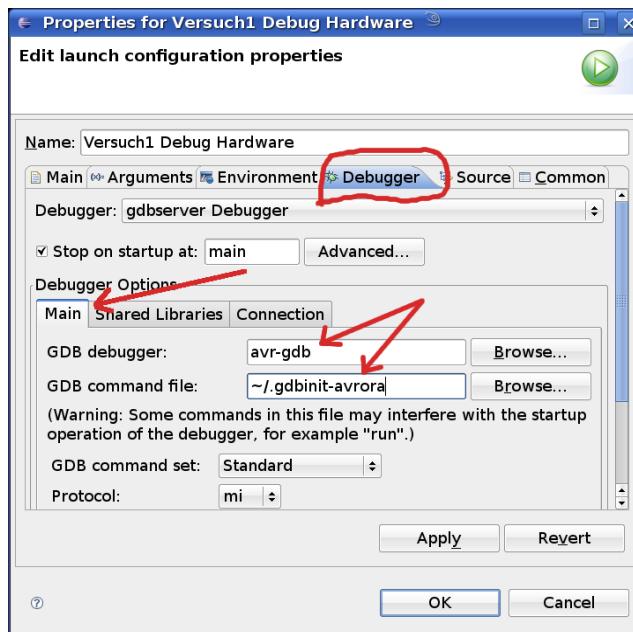
1.7 Eclipse mit AVR-Simulator

Das Zusammenspiel zwischen dem Simulator 'avrora' und eclipse ist ebenfalls auf dem gdbserver Protokoll basierend. Es gibt folgendes zu beachten:

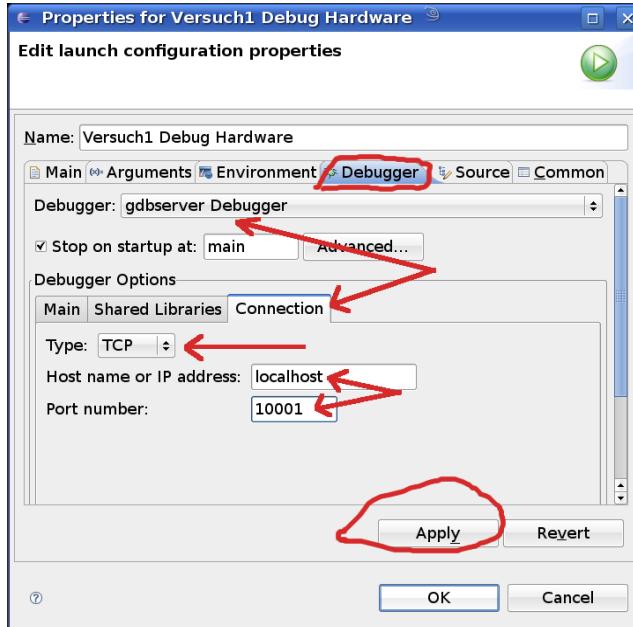
- Wir verwenden eine modifizierte Version des avrora Simulators, die von anderer im Netz erhältlichen abgeleitet wurde,
- Es einen Hardwaremonitor, der Änderungen der Geräteregister in hex anzeigt und auch mit einem Zeitstempel versieht,
- Es gibt einen seriell-Monitor, der Ein-/Ausgaben auf der seriellen Schnittstelle über TCP/IP anschliesst bzw. simuliert (telnet). Die Baudrate wird hierbei nicht berücksichtigt,
- PORTA als Input und PORTB als Output sowie der ADC (Kanal 0) werden unterstützt.
- Der Pause-Knopf im Debugger ist nicht implementiert.
- In JAVA geschrieben, also auch auf anderen Plattformen lauffähig.

Ein typischer manueller Aufruf von avrora ist weiter hinten beschrieben.

An Änderungen sind für die Simulationsumgebung bei Eclipse notwendig, die Sie mit 'Run->Open Debug Dialog', nach der Anwahl des Reiters 'Debugger' bekommen:



Der avrora-Simulator verwendet als Port 10001, so dass es hier nicht zu Konflikten mit dem anderen Tool 'avarice' kommen kann.



Es sollte bei Debug dann neben dem 'avrora'-Fenster ein 'telnet'-Fenster auftauchen, welches die serielle Schnittstelle anbindet.

1.7.1 scanf/printf and serielle Schnittstelle binden

Die formatierte Ausgabe von Daten kann in C einfach mittels der Routinen

- `scanf()` u. `printf()` bzw. auf beliebigen Dateien mit
- `fscanf()` u. `fprintf()`

erfolgen (s.a. Manualseiten auf dem Server). Jedes C-Programm bekommt automatisch drei offene Kommunikationskanäle (0,1, und 2) beim Start von 'main()' geliefert:

- 0: `stdin` (**genutzt von `scanf()`**)
- 1: `stdout` **genutzt von `printf()`**
- 2: `stderr` **nutzbar durch `fprintf(stderr,...)` oder `perror()`**

Die Namen `stdin`, `stdout` und `stderr` sind auch die Namen globaler Variablen (aus `stdio.h` zu importieren). Die Routinen `scanf/printf/fprintf` bedienen sich weiterer Routinen zur Formatierung und Umwandlung sowie auf unterster Ebene Routinen, die nur genau ein Zeichen ('`getchar()`') lesen bzw. genau ein Zeichen ausgeben ('`putchar()`'). Diese Routinen sind für jeden Kanal frei konfigurier- bzw. ersetzbar. In der Umgebung eines Betriebssystems wären diese Kanäle typischerweise auf die Tastatur (0) und die Konsolausgabe (1 und 2) bzw. einen Textwindow gelegt. Im Falle von Embedded Systems ist dies aber nicht so einfach festzulegen, weil es unklar ist, welches Gerät für eine solche Kommunikation zuständig ist. Falls Sie also mit `scanf/printf` arbeiten wollen (diese verwenden Kanal 0/`stdin` bzw. 1/`stdout`), kann hierzu der untenstehende Programmausschnitt verwendet werden, der die ursprünglichen bei uns nicht tauglichen (leeren) Default-Routinen '`getchar`' und '`putchar`' (vom Laufzeitsystem eingetragen) durch eigene, anwendungsspezifische Routinen ersetzt. Dies erfolgt durch den Aufruf von '`FDEV_SETUP_STREAM()`'. Die Kanäle 0,1 und 2 sind dabei mit globalen

Variablen vom Typ 'FILE' (stdin, stdout, stderr; jeweils ein Zeiger auf eine Struktur '`* FILE`') assoziiert, die Details zur Pufferung, Statusinformationen etc. enthalten und die zum Lesen und Schreiben zu verwendenden 'Low-Level-Routinen' in Form von Funktionszeigern beinhalten.

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

//BAUD muss vor setbaud.h vorhanden sein
#define BAUD 9600
#include <util/setbaud.h>

void uart_init(void);
int uart_putchar(char c, FILE *stream);
int uart_getchar(FILE *stream);

static FILE mystdout = FDEV_SETUP_STREAM( uart_putchar, NULL, _FDEV_SETUP_WRITE );
static FILE mystdin = FDEV_SETUP_STREAM( NULL, uart_getchar, _FDEV_SETUP_READ );

void uart_init(void)
{
    UCSROB = (1<<TXENO) | (1<<RXENO);      // UART TX und RX einschalten
    UCSROC = (1<<USBS0)|(3<<UCSZ00);        // 2 stop bits, 8Bit Data
    UBRROH = UBRRH_VALUE;
    UBRROL = UBRLL_VALUE;
    stdout = &mystdout;
    stdin = &mystdin;
}

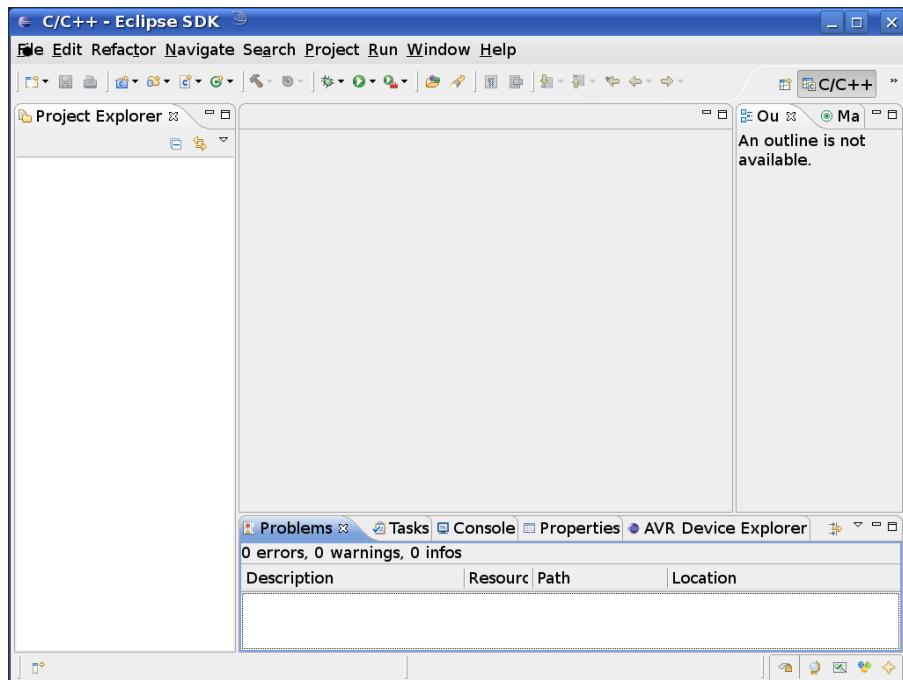
int uart_putchar( char c, FILE *stream )
{
    if( c == '\n' ) uart_putchar( '\r', stream );
    loop_until_bit_is_set( UCSROA, UDRE0 );
    UDRO = c;
    return 0; // 0: Falls ohne Fehler gelesen
}

int uart_getchar(FILE *stream)
{
    char c;
    loop_until_bit_is_set(UCSROA, RXC);
    c = UDRO;
    uart_putchar(c, stdout);
    if (c == '\r') return '\n';
    return(c);
}

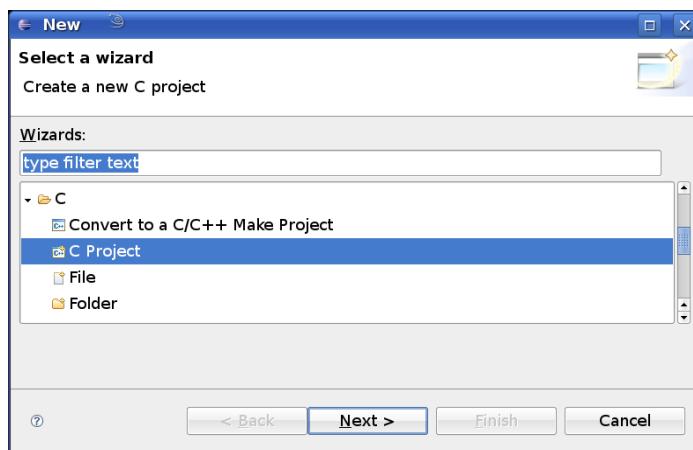
int main(void)
{
    char c;
    int i;
    uart_init();
    printf( "Hello, world!\n" );
    scanf("%d",&i);
    printf("i*i=%08d\n",i*i);
    loop_until_bit_is_set( UCSROA, TXC );
    return 0;
}
```

Hinweis: Bei den Prozessoren ATMega32 und ATMega168 beispielsweise gibt es nur eine Schnittstelle. Damit entfällt der Index 0 bzw. 1. Darüber hinaus gibt es die Besonderheit, dass beim Schreiben auf UCSRC das MSBit gesetzt sein muss (s.a. z.B. Datenblatt ATMega 162 Seite 184 bzw. 189!)

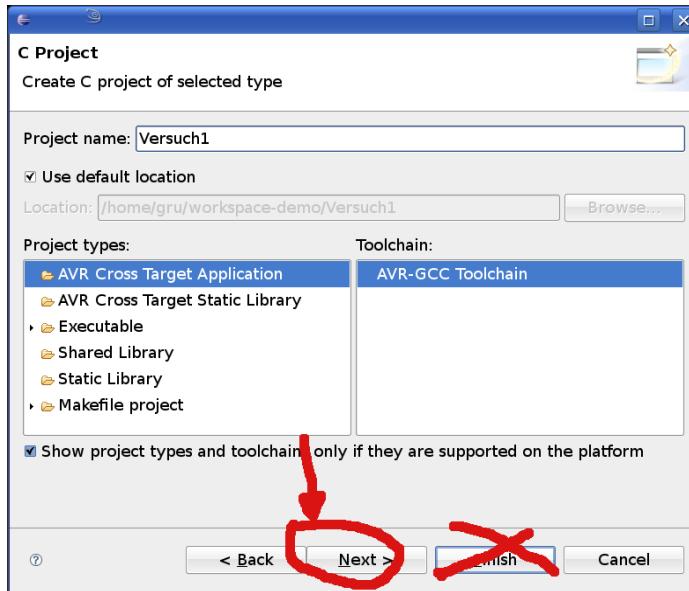
1.8 Eclipse ohne TEMPLATE



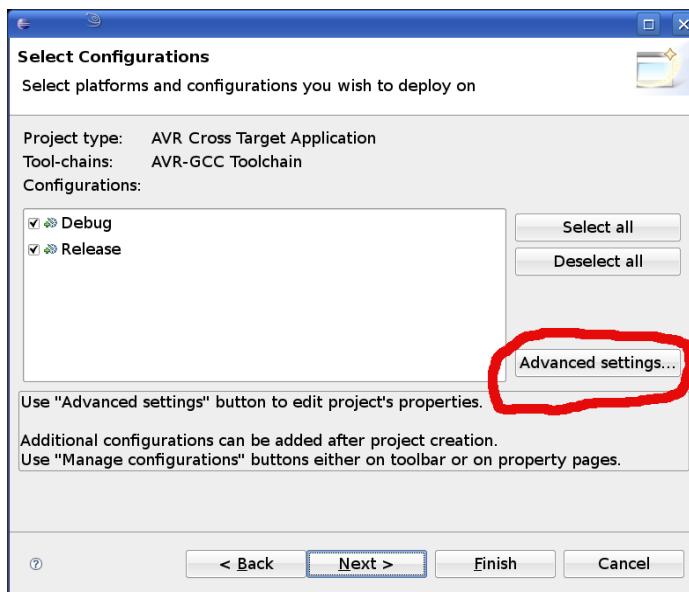
Drücken Sie 'File->new->Project' und gehen mit 'Next' weiter:



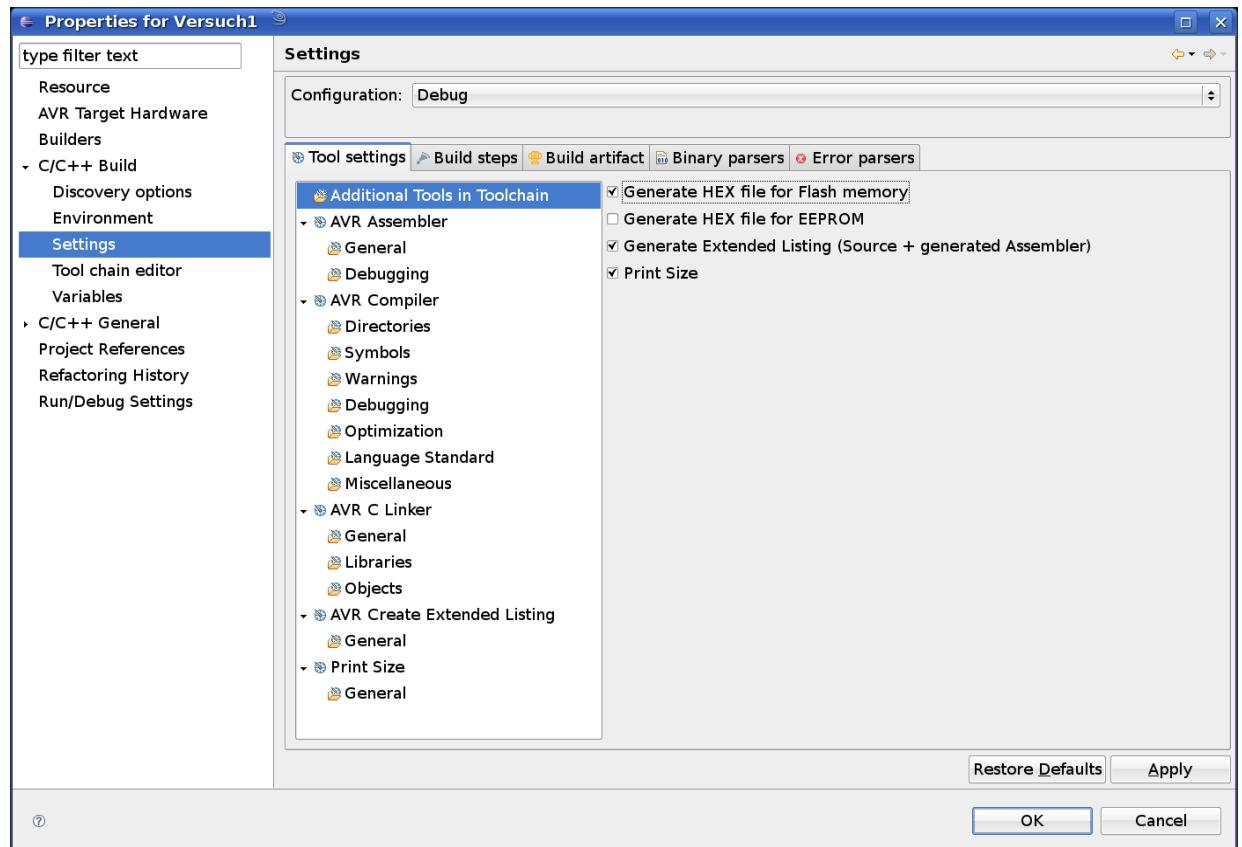
Wählen Sie 'C Project' aus und gehen mit 'Next' weiter:



Geben Sie dem Projekt einen Namen und wählen Sie den Projekt Typ 'AVR Cross Target Application' und gehen Sie mit 'NEXT' weiter:

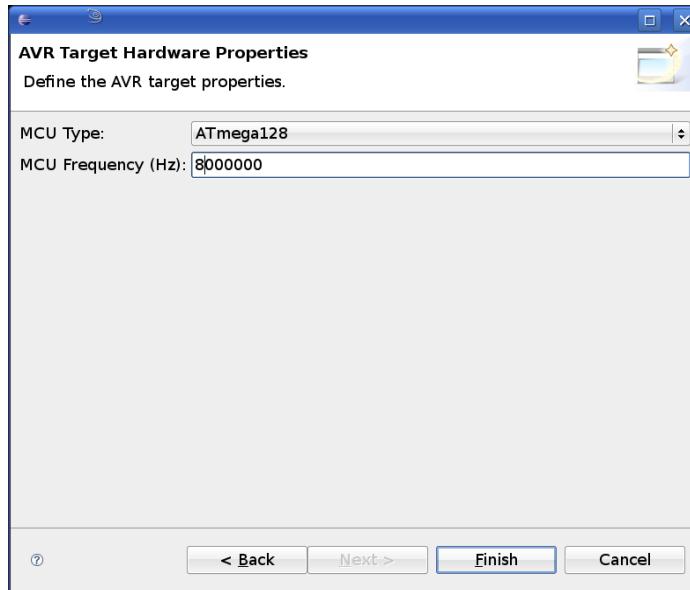


Mittels 'Advanced Settings' und über 'C/C++ Settings' stellen Sie unter 'Additional Tools' die Option 'Generate HEX file...' an, die das Flash-Speicherabbild erzeugt. Mit 'NEXT' geht es weiter:

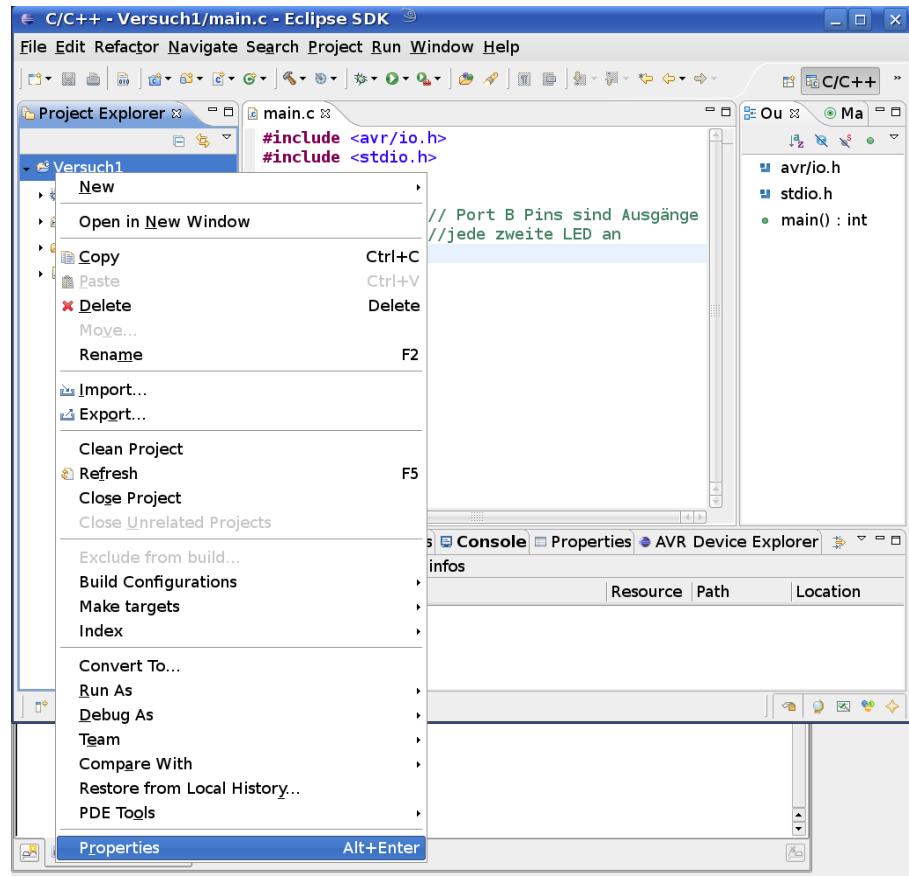


Mit 'AVR Target Hardware' stellen Sie den Prozessortyp ein. 'FINISH'.

Jetzt muss die DEBUG Einstellung angepasst werden (Über Project->Properties'):



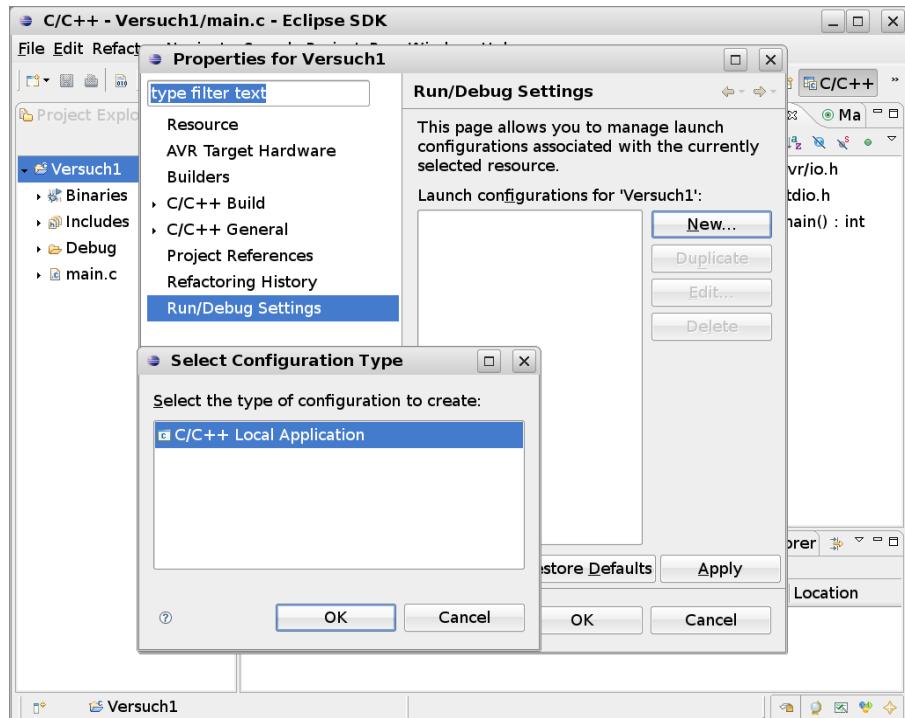
Drücken Sie 'NEW' und erzeugen eine neue DEBUG-Konfiguration für 'C/C++':



An Einstellungen ist folgendes zu ändern (ohne die Hochkommata):

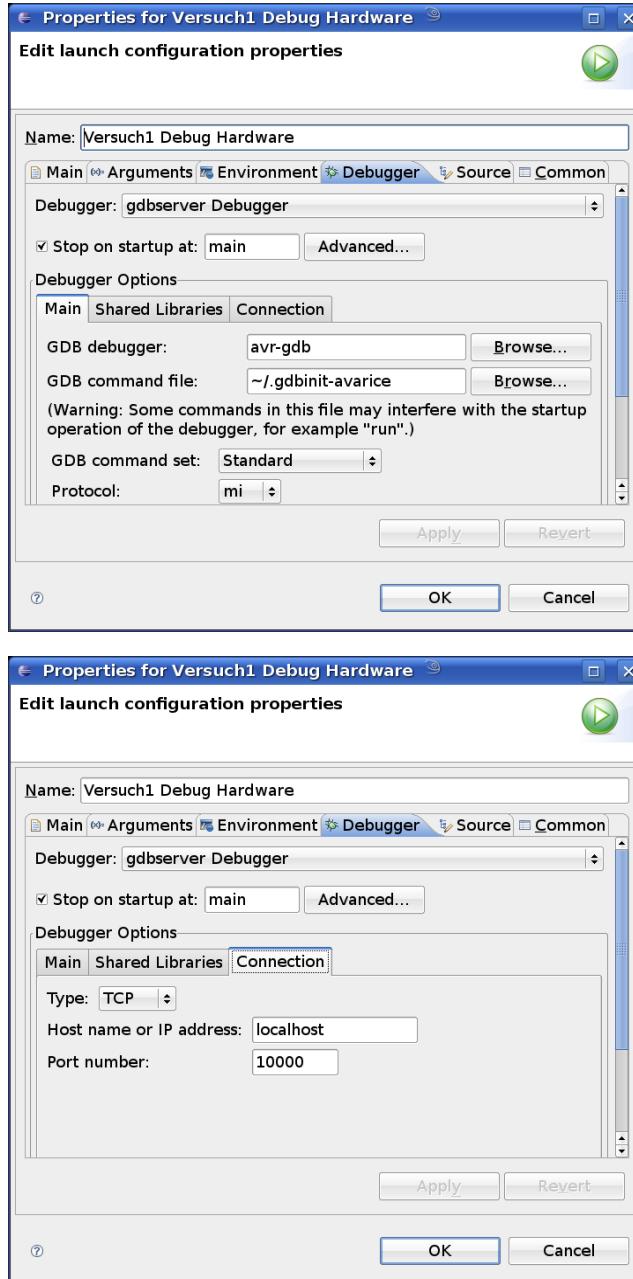
1. Auswahlmenü 'Debugger': 'gdbserver Debugger'
2. Feld 'GDB debugger': Eintragen von 'avr-gdb'
3. Feld GDB command file: Eintragen von '~/gdbinit-avarice'

Dann ist der Reiter 'Connection' anzuwählen um die Verbindung zum Debugger-Server zu konfigurieren:



1. Das Auswahlmenü 'Type': TCP'
2. Das Feld 'host name...': localhost
3. Das Feld 'Port number': 10000

Mit 'OK' beenden dieses Dialoges.



Interessant ist auch das Setting Window->Preferences...->General->Workspace->Save automatically before welches vor dem build-Prozess alle geänderten Files automatisch abspeichert. Falls man das Abspeichern nämlich vergisst, wird leider das alte File kompiliert!

1.9 Manuell: gdb Debugger und avarice

Eclipse erwartet zum Debuggen einen sogenannten gdbserver-Prozess (macht avarice), der über

- USB auf der einen Seite zum AVR-Prozessor hält und auf der anderen Seite

- die Netzwerk- bzw. TCP/IP-Schnittstelle bereithält.
- Diese wird dann vom Debugger 'avr-gdb' angesprochen, der wiederum auf der anderen Seite Fernbedienungskommandos akzeptiert.
- Darüber wird 'avr-gdb' von Eclipse fernbedient.

Das ProgrammierTool avarice liest eine Datei im Intel-Hex-Format (meist Endung .hex), der avr-gdb und Eclipse brauchen eine Datei im ELF-Format, (meist Endung .elf oder keine Endung). Von Eclipse werden beide Formate erzeugt (.hex s. a. zusätzliche Konfigurationseinstellungen von Eclipse).

Sowohl das Tool 'avarice' wie auch der Simulator 'avrora' bedienen diese gdbserver-Schnittstelle.

1.9.1 Start von avarice

Zum Test sollte zunächst von Hand das tool 'avarice' gestartet werden, um die USB-Schnittstelle und damit die Verbindung zum ATMELprogrammer und zur Prozessorplatine zu testen und auch das Flash zu löschen:

```
avarice --erase -j usb --mkII :10000
```

Ist dies gelungen, kann man beispielsweise das Programm 'testmich' laden:

```
avarice --file testmich.hex --erase --program --verify -2 -j usb --mkII :10000
```

Der Aufruf macht folgendes:

1. Er über gibt dem Tool den Namen einer Datei, die im Intel-Hex-Format die Daten für das Flash enthält.
2. Das Flash wird gelöscht (--erase).
3. Das Flash wird mit dem Inhalt der Datei programmiert (--program).
4. Der Inhalt des Flash wird ausgelesen und mit dem Dateiinhalt verglichen (--verify).
5. Übertragungsprotokoll Version 2 (-2).
6. Benutzt werden soll die usb Schnittstelle. Der Programmer wird automatisch gesucht und erkannt.
7. Der Programmer ist vom Typ mkII.
8. Der gdbserver-Teil von avarice wartet auf dem lokalen Rechner, Port 10000 auf eine Verbindung (von z.B. eclipse)

Dies kann man auch automatisieren, in dem man die Initialisierungsdatei für den Eclipse-gdb Aufruf anpasst und dort etwa ein Kommando zum Starten einer Kommando-Batch-Datei gibt:

```
shell /usr3/avr/bin/do-avarice ${PWD}
```

Diese muss dann aus dem Wordirekotory \${PWD}, welches als Parameter an die Kommandodatei übergeben wird, die richtigen Dateien (erkennbar an den Endungen) verarbeitet.

1.9.2 Start von avrora

Ein Start von avrora könnte also so aussehen:

```
java -jar avrora-beta-1.7.105.jar -platform=mica2 -nodecount=1  
-monitors=gdb,leds,serial -verbose=loader,atmel.flash -status testmich.elf
```

Kapitel 2

Anhang zum ATMEGA128

2.0.3 Symbolische Namen der Interruptvektoren

```
#define INTO_vect           _VECTOR(1)
#define INT1_vect            _VECTOR(2)
#define INT2_vect            _VECTOR(3)
#define INT3_vect            _VECTOR(4)
#define INT4_vect            _VECTOR(5)
#define INT5_vect            _VECTOR(6)
#define INT6_vect            _VECTOR(7)
#define INT7_vect            _VECTOR(8)
#define TIMER2_COMP_vect     _VECTOR(9)
#define TIMER2_OVF_vect      _VECTOR(10)
#define TIMER1_CAPT_vect     _VECTOR(11)
#define TIMER1_COMPA_vect    _VECTOR(12)
#define TIMER1_COMPB_vect    _VECTOR(13)
#define TIMER1_OVF_vect      _VECTOR(14)
#define TIMERO_COMP_vect     _VECTOR(15)
#define TIMERO_OVF_vect      _VECTOR(16)
#define SPI_STC_vect         _VECTOR(17)
#define USART0_RX_vect       _VECTOR(18)
#define USART0_UDRE_vect     _VECTOR(19)
#define USART0_TX_vect       _VECTOR(20)
#define ADC_vect              _VECTOR(21)
#define EE_READY_vect         _VECTOR(22)
#define ANALOG_COMP_vect     _VECTOR(23)
#define TIMER1_COMPC_vect    _VECTOR(24)
#define TIMER3_CAPT_vect     _VECTOR(25)
#define TIMER3_COMPA_vect    _VECTOR(26)
#define TIMER3_COMPB_vect    _VECTOR(27)
#define TIMER3_COMPC_vect    _VECTOR(28)
#define TIMER3_OVF_vect      _VECTOR(29)
#define USART1_RX_vect       _VECTOR(30)
#define USART1_UDRE_vect     _VECTOR(31)
#define USART1_TX_vect       _VECTOR(32)
#define TWI_vect              _VECTOR(33)
#define SPM_READY_vect        _VECTOR(34)
```

2.0.4 Symbolische Namen Timer0 Register Bits

```
/* Timer/Counter Interrupt MaSK register - TIMSK */
#define OCIE2      7
#define TOIE2      6
#define TICIE1      5
#define OCIE1A     4
#define OCIE1B     3
#define TOIE1      2
#define OCIE0      1
#define TOIE0      0

/* Timer/Counter Interrupt Flag Register - TIFR */
#define OCF2       7
#define TOV2       6
#define ICF1       5
#define OCF1A     4
#define OCF1B     3
#define TOV1       2
#define OCF0       1
#define TOVO       0

/* Timer/Counter 0 Control Register - TCCRO */
#define FOC0       7
#define WGM00      6
#define COM01      5
#define COM00      4
#define WGM01      3
#define CS02       2
#define CS01       1
#define CS00       0

/* Timer/Counter 0 Asynchronous Control & Status Register - ASSR */
#define AS0        3
#define TCNOUB     2
#define OCROUB     1
#define TCROUB     0

/* Watchdog Timer Control Register - WDTCR */
#define WDCE       4
#define WDE        3
#define WDP2       2
#define WDP1       1
#define WDPO       0
```

2.0.5 Symbolische Namen Watchdog Register Bits

```
/* Watchdog Timer Control Register - WDTCR */
#define WDCE       4
#define WDE        3
#define WDP2       2
#define WDP1       1
#define WDPO       0
```

2.0.6 Symbolische Namen der USART0 Register Bits

```
/* USART0 Register C - UCSROC */
#define UMSEL0      6
#define UPM01       5
#define UPM00       4
#define USBS0       3
#define UCSZ01      2
#define UCSZ00      1
#define UCPOLO     0

/* USART0 Status Register A - UCSROA */
#define RXCO        7
#define TXCO        6
#define UDREO        5
#define FEO          4
#define DOR0          3
#define UPEO          2
#define U2X0          1
#define MPCMO         0

/* USART0 Control Register B - UCSROB */
#define RXCIE0       7
#define TXCIE0       6
#define UDRIE0       5
#define RXENO        4
#define TXENO        3
#define UCSZ02       2
#define RXB80         1
#define TXB80         0
```

2.0.7 Symbolische Namen der ADC Register Bits

```
/* Analog Comparator Control and Status Register - ACSR */
#define ACD          7
#define ACBG         6
#define ACO          5
#define ACI          4
#define ACIE         3
#define ACIC         2
#define ACIS1        1
#define ACISO        0

/* ADC Control and status register - ADCSRA */
#define ADEN         7
#define ADSC         6
#define ADFR         5
#define ADIF         4
#define ADIE         3
#define ADPS2        2
#define ADPS1        1
#define ADPS0        0
```

```
/* ADC Multiplexer select - ADMUX */  
#define    REFS1      7  
#define    REFS0      6  
#define    ADLAR      5  
#define    MUX4       4  
#define    MUX3       3  
#define    MUX2       2  
#define    MUX1       1  
#define    MUX0       0
```