

Tutorial

Setup einer freien ARM Toolchain unter Windows XP pro Sp3

Julian Friedrich BSc
A-1180 Wien, Erndtgasse 3

Wien, 03.02.2010



Erstellt im Rahmen des Projekts CanCar

Studiengang MES

EmbSys Projekt 2009/10

Kurzfassung

Dieses Tutorial soll bei der Installation einer freien Entwicklungsumgebung für ARM Prozessoren und deren Konfiguration für die Verwendung des CodeSourcery G++ Lite EABI ARM C/C++ Compiler behilflich sein. Des Weiteren soll die Installation des openOCD GNU Debuggers beschrieben werden, sowie dessen Integration in die Eclipse Umgebung. Das besondere an dieser Kombination ist die Tatsache, dass alle hierbei verwendeten Tools frei erhältlich sind, was den Einsatz im universitären Betrieb erheblich erleichtert. Zudem kann die so entstehende Entwicklungsumgebung für die Benutzung eines Echtzeitbetriebssystems (freeRTOS) vorbereitet werden indem ein zusätzliches Plug-in installiert wird, welches beim Debuggen von Applikationen in der Umgebung eines Echtzeitbetriebssystems behilflich ist. Diese Vorbereitung ist jedoch optional und für die Erstellung einfacher Beispiele ohne RTOS nicht notwendig.

Inhaltsverzeichnis

1	Installation	3
1.1	Verwendete Programmversionen.....	3
1.2	Verwendete Hardware	4
1.3	CodeSourcery G++ EABI Setup.....	4
1.4	openOCD 0.3.1 Setup	6
1.4.1	openOCD Treiber Installation (Amontec JTAG-Key Tiny)	7
1.5	Eclipse.....	9
1.6	Installation der nötigen Eclipse Plug-ins	11
1.6.1	GNU ARM Eclipse sowie Zylind CDT Plug-in.....	11
1.6.2	FreeRTOS Stateviewer Plug-in (optional)	14
2	Konfiguration	17
3	Beispiel Anwendung	21
3.1	Ein neues Projekt anlegen.....	21
3.2	Anpassen der openOCD Skripte	22
3.3	Das neue Projekt Konfigurieren	23
3.4	Kompilieren des Projekts.....	26
3.5	Download in den Flash	27
3.6	Debuggen der Applikation.....	28
3.6.1	Anlegen einer Debug Konfiguration	28
3.6.2	Starten einer Debug Session.....	30
3.6.3	Beenden einer Debug Session	32
4	How To	34
4.1	Code Editor	34
4.1.1	Code Completion bzw Content Assist.....	34
4.1.2	Folding Aktivieren.....	34
4.1.3	Automatisch Speichern vor einem Build.....	34
4.1.4	Suchfunktion.....	34
4.1.5	Zum Funktionsaufruf gehörende Funktion öffnen	35
4.1.6	Sprung zwischen C und H Dateien.....	35
4.2	Debuggen	35
4.2.1	Einen Breakpoint hinzufügen bzw. entfernen	35
4.2.2	Einen Ausdruck zum Expressions Fenster hinzufügen	35
4.2.3	Darstellungsart einer Variablen ändern	35

Literaturverzeichnis	36
Abbildungsverzeichnis	38
Buildprotokoll des Projekt Blinky	40
openOCD Output während des Flashen	43

1 Installation

Der erste Teil dieses Tutorials befasst sich mit der Installation der verschiedenen benötigten Tools, sowie deren Integration in die Eclipse Umgebung. Nach der erfolgreichen Installation wird die Verwendung der in diesem Tutorial vorgestellten Toolchain anhand eines Beispiels demonstriert (siehe Kapitel 3).

1.1 Verwendete Programmversionen

Eines der größten Probleme bei der Verwendung einer derartigen Toolchain ist die hohe Geschwindigkeit mit der die benötigten Tools aktualisiert bzw. weiterentwickelt werden. Daher ist es des Öfteren nicht möglich die jeweils aktuellsten Tools einzusetzen, da diese von Plug-Ins oder anderen Tools oft (noch) nicht unterstützt werden. Aus diesem Grund ist es auch ratsam die Zusammenstellung der Toolchain während eines laufenden Projekts nur dann zu ändern, wenn es unbedingt notwendig ist. In diesem Tutorial werden die folgenden Tools verwendet, welche abgesehen von der Eclipse IDE alle dem aktuellsten Stand entsprechen:

Programm	Version	Link
CodeSourcery G++ Lite EABI	2009q3-68	Siehe [CoSo10a] im Literaturverzeichnis
openOCD	0.3.1	Siehe [Chop10] im Literaturverzeichnis
Eclipse C/C++ IDE	3.4.2 (Ganymede SR2)	Siehe [Ecl109] im Literaturverzeichnis
GNU ARM Eclipse Plug-in	0.5.3.201001261103	Siehe [GAEP10] im Literaturverzeichnis
Zylin Embedded CDT	4.10.1	Siehe [Zyl10] im Literaturverzeichnis
Stateviewer Plug-in	1.07	Siehe [Wit10] im Literaturverzeichnis

Tabelle 1 (Verwendete Software Versionen)

Die etwas ältere Eclipse Version wird verwendet da das GNU ARM Eclipse Plug-In zum jetzigen Zeitpunkt noch keine Unterstützung für Eclipse Galileo bietet, was aber laut den Entwicklern des GNU ARM Eclipse Plug-Ins bald der Fall sein sollte.

Anmerkung zu openOCD

Während der Projektlaufzeit wurde in der openOCD Community diskutiert ob das Integrieren der proprietären FTD2XX Treiber in das Installer Paket von openOCD gegen die GPL Lizenz verstößt unter der openOCD veröffentlicht wird. Die Antwort auf diese Frage wurde leider mit ja beantwortet, was zur Folge hatte, dass die „offiziellen“ openOCD Installer Binaries nicht mehr länger zum Download angeboten werden dürfen. Zwar werden nachwievor die openOCD Quelldateien zum Download angeboten, aber diese müssen vom Benutzer erst für das jeweilige Hostbetriebssystem kompiliert werden, was leider alles andere als trivial ist. Allerdings wurde openOCD von Freddie Chopin zusammen mit den libUSB Treibern (welche ebenfalls ein Open Source Projekt sind und daher unter der GPL Lizenz stehen) für 32Bit Windows Systeme kompiliert und auf seiner Homepage (siehe [Chop10]) in Form eines Installationspakets zum freien Download zur Verfügung gestellt. Ein 64Bit Support dieses Pakets ist im Moment nicht gegeben, es wird aber an einer entsprechenden Version gearbeitet

1.2 Verwendete Hardware

Bei der in diesem Tutorial verwendeten Hardware handelt es sich zum einen um den Amontec JTAG-Key Tiny der Firma Amontec (siehe Abbildung 1 sowie [Amo10]), sowie zum anderen um ein ARM Cortex-M3 Evalboard der Firma Micro4you mit einem STM32F103RB Mikrocontroller (siehe Abbildung 2 sowie [Mic10]).



Abbildung 1 (Amontec JTAG-Key Tiny)

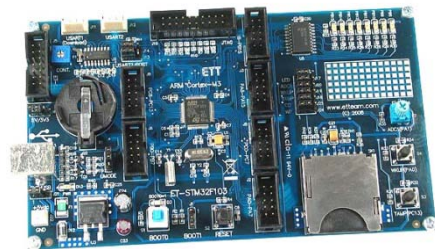


Abbildung 2 (STM32 Evalboard)

1.3 CodeSourcery G++ EABI Setup

Die Installation des CodeSourcery G++ EABI Compilers wird durch Ausführen der Datei „arm-2009q3-68-arm-none-eabi.exe“ gestartet, welche von der CodeSourcery Homepage (Link siehe [CoSo10a]) geladen werden kann. Nachdem der Willkommen Dialog durch Klicken auf die Schaltfläche „Next“ bestätigt wurde, und die Lizenzbestimmungen akzeptiert wurden, kann der nachfolgende Dialog mit „Next“ bestätigt werden. Im folgenden Dialog (siehe Abbildung 3) kann der Installationsumfang ausgewählt werden, wobei die Standard-Einstellung „Typical“ beibehalten wird und der Dialog mit „Next“ bestätigt wird.

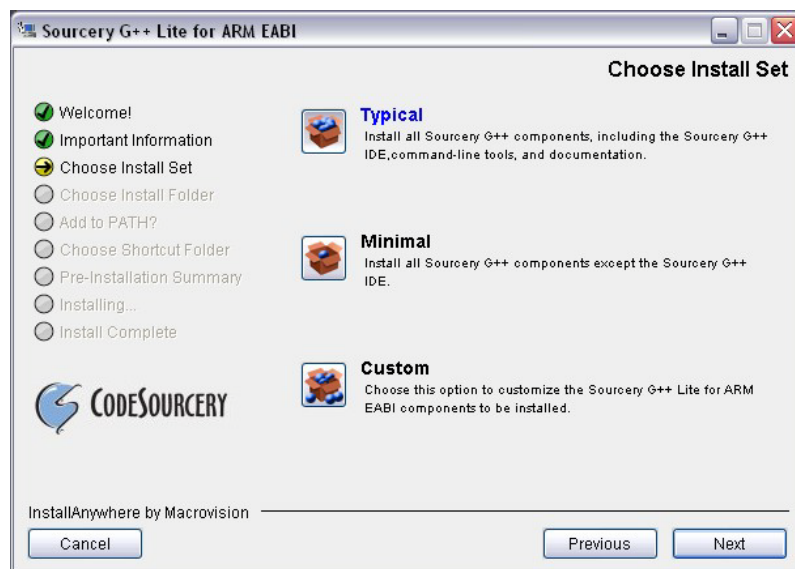


Abbildung 3 (Codesourcery G++ Lite EABI Setup, Schritt 4)

Selbiges gilt auch für den nächsten Schritt bei dem der Installationspfad gewählt werden kann. Im folgenden Schritt kann der Benutzer wählen ob die Umgebungsvariable „PATH“

vom Setup automatisch angepasst werden soll, was durch auswählen der Option „*Modify PATH for current user*“ und anklicken von „*Next*“ bestätigt wird.

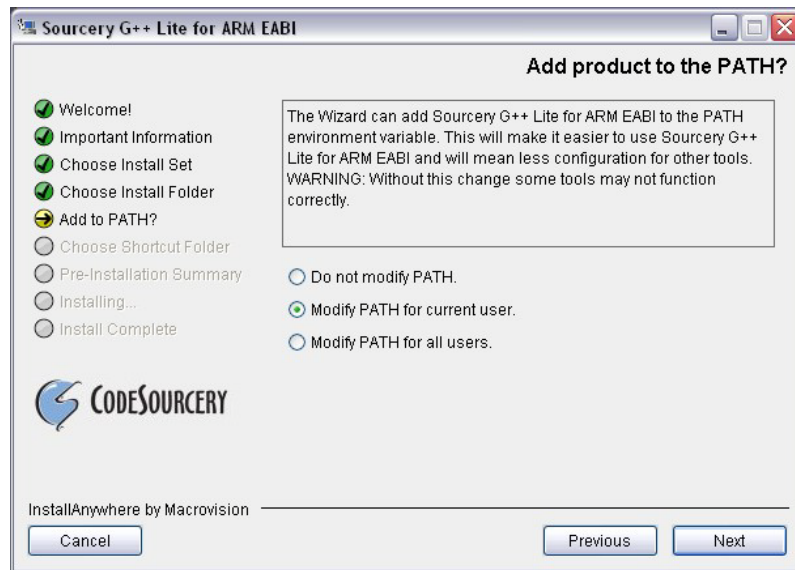


Abbildung 4 (Codesourcery G++ Lite EABI Setup, Schritt 6)

Nachdem der folgende Schritt durch Auswahl des Startmenü Ordners und Bestätigung durch klicken auf „*Next*“ abgeschlossen ist, kann die Installation durch einen Klick auf die Schaltfläche „*Instal*“ gestartet werden. Nach Abschluss der Installation kann ausgewählt werden ob ein „*Getting Started Guide*“ angezeigt werden soll (was auch später mittels einer Verknüpfung im Startmenüordner der Installation nachgeholt werden kann). Danach wird die Installation durch klicken auf „*Done*“ abgeschlossen. Um zu Überprüfen ob das Setup fehlerfrei ausgeführt wurde, wird die Windows XP Eingabeaufforderung geöffnet und die beiden folgenden Befehle eingegeben:

```
arm-none-eabi-gcc --version
arm-none-eabi-gdb --version
```

Wenn das Setup erfolgreich war sollte der der in Abbildung 5 dargestellte Inhalt der Konsole zu sehen sein.

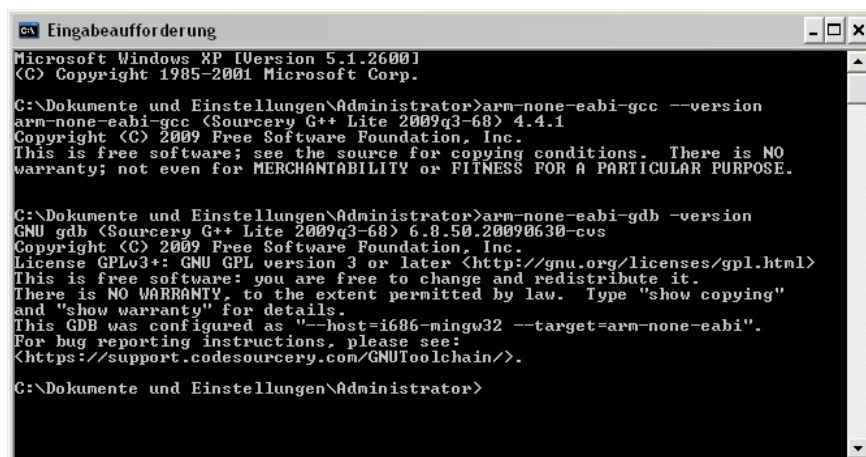


Abbildung 5 (Verifikation des CodeSourcery Compiler Setups)

1.4 openOCD 0.3.1 Setup

Nachdem das auf Freddie Chopin's Homepage (Link siehe [**Chop10**]) angebotene openOCD 0.3.1 Archiv heruntergeladen wurde muss die darin enthaltene Datei „*openocd.msi*“ entpackt werden, und das Setup durch einen doppelklick auf diese Datei gestartet werden. Nach dem Akzeptieren der Lizenzbestimmungen kann der Installationsumfang den Bedürfnissen des Benutzers angepasst werden, was aber in unserem Fall nicht nötig ist weshalb der Dialog durch klicken auf „*Next*“ übersprungen werden kann (Siehe Abbildung 6).

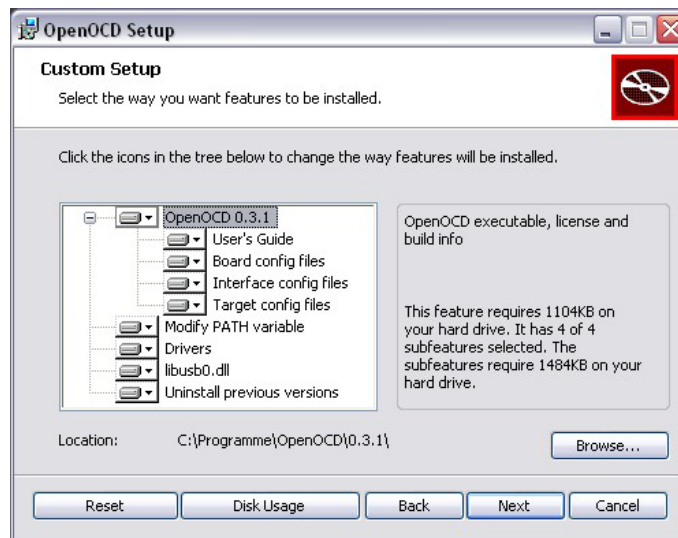


Abbildung 6 (openOCD 0.3.1 Setup Optionen)

Im folgenden Dialog wird die Installation durch klicken auf „*Install*“ gestartet. Danach kann der openOCD Setup Assistent durch klicken auf „*Finish*“ abgeschlossen werden. Um zu Überprüfen ob das Setup fehlerfrei ausgeführt wurde, wird die Windows XP Eingabeaufforderung geöffnet und der folgende Befehl eingegeben:

```
openocd --version
```

Wenn das Setup erfolgreich war sollte der in Abbildung 7 dargestellte Inhalt der Konsole zu sehen sein.

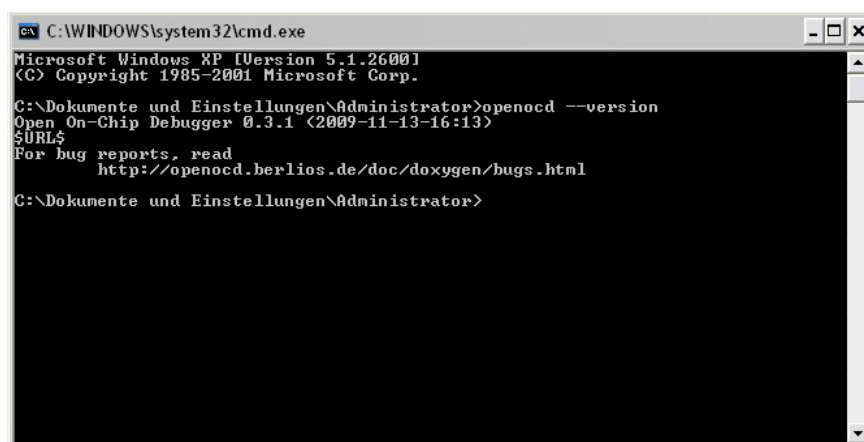


Abbildung 7 (Test der openOCD 0.3.1 Installation)

1.4.1 openOCD Treiber Installation (Amontec JTAG-Key Tiny)

Im letzten Schritt der openOCD Installation müssen die passenden Treiber zum verwendeten JTAG-Adapter installiert werden. Zu diesem Zweck müssen die mit der openOCD Installation gelieferten Treiberpakete aber erst entpackt werden, da sie sonst von Windows XP nicht gefunden werden können. Diese Treiberpakete können im Unterordner „drivers“ der OpenOCD Installation gefunden werden (konkret im Ordner „C:\Programme\OpenOCD\0.3.1\drivers“ auf dem Computer mit dem dieses Tutorial erstellt wurde), da aber in diesem Projekt lediglich der Amontec JTAG-Key Tiny verwendet wird muss nur das Archiv „libusb-win32_ft2232_driver-091105.zip“ entpackt werden. Die in diesem Archiv enthaltenen Dateien können direkt in denselben Ordner entpackt werden (Siehe Abbildung 8), wodurch ein neuer Ordner namens „libusb-win32_ft2232_driver-091105“ entstehen sollte (siehe Abbildung 9).



Abbildung 8 (Extrahieren der Treiberdateien)

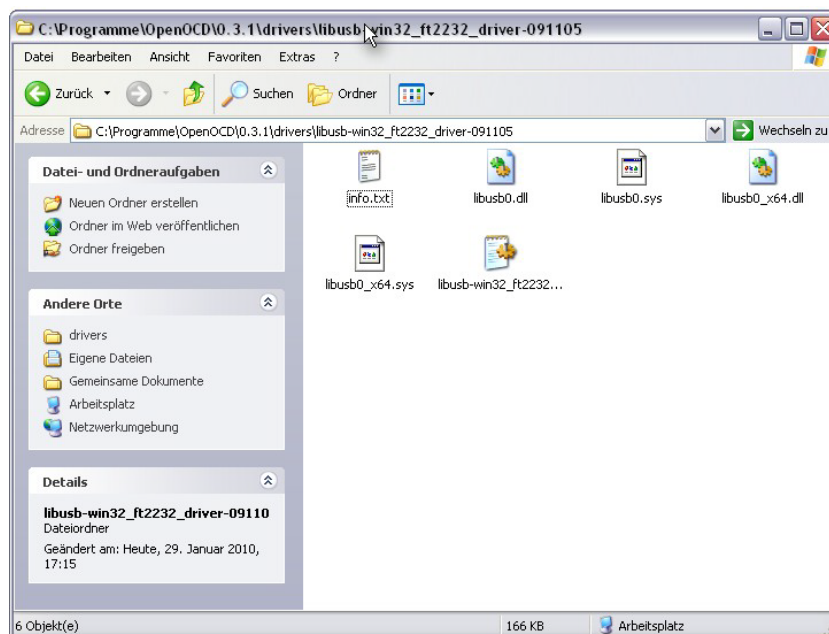


Abbildung 9 (libUSB ftdi Treiber Dateien)

Nun ist es an der Zeit den Amontec JTAG-Key Tiny erstmals an den Computer anzuschließen. Windows wird dabei nach einem passenden Treiber verlangen, welcher sich in

dem soeben entpackten Ordner befindet. Daher wird die Option „Software von einer Liste oder bestimmten Quelle installieren“ gewählt, und der Ordner mit den Treiberdateien angegeben (siehe Abbildung 10).

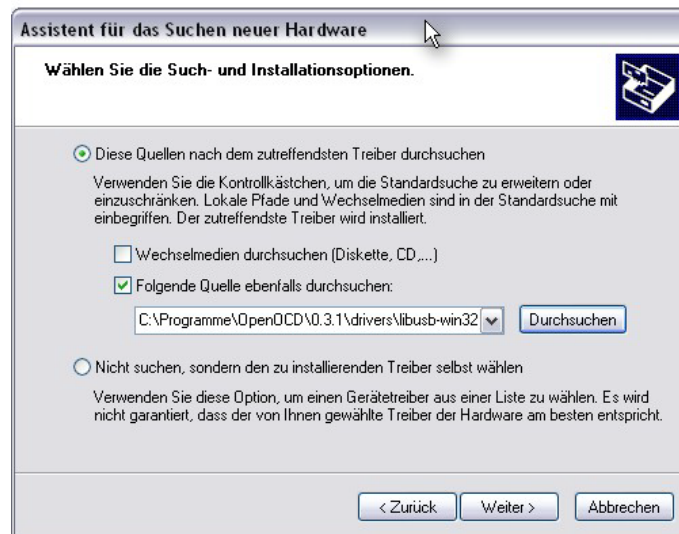


Abbildung 10 (Amontec JTAGkey Tiny Treiber Setup)

Windows sollte nun mit Hilfe des in dem angegebenen Ordner gefundenen Treibers ein Gerät namens „Amontec JTAGkey (Channel A)“ erkennen und korrekt installieren (siehe Abbildung 11). Nachdem die Treiberinstallation für dieses Gerät abgeschlossen ist, sollte ein weiteres Gerät von Windows erkannt werden. Die Vorgehensweise ist in diesem Fall genau dieselbe wie zuvor, jedoch wird nun ein Gerät namens „Amontec JTAGkey (Channel B)“ erkannt. Dies liegt daran das der im Amontec JTAG-Key Tiny benutzte FTDI Chip immer zwei Kommunikationskanäle besitzt, wobei allerdings nur einer genutzt wird.



Abbildung 11
(Treibersetup Amontec JTAGkey Channel A)



Abbildung 12
(Geräte manager nach dem Treiber setup)

Nachdem nun beide Kommunikationskanäle des JTAGkey Tiny korrekt installiert wurden (sie sollten nun im Geräte manager unter „LibUSB-Win32 Devices“ zu finden sein, siehe Abbildung 12) ist das Setup von openOCD 0.3.1 abgeschlossen.

1.5 Eclipse

Da Eclipse auf Java basiert, ist es zwingend erforderlich, dass eine aktuelle Java Version auf dem betreffenden Computer installiert ist. Ist man sich nicht sicher ob, bzw. welche Version von Java installiert ist, so kann dies durch Eingabe des Befehls

```
Java -version
```

in der Kommandozeile verifiziert werden (siehe Abbildung 13).

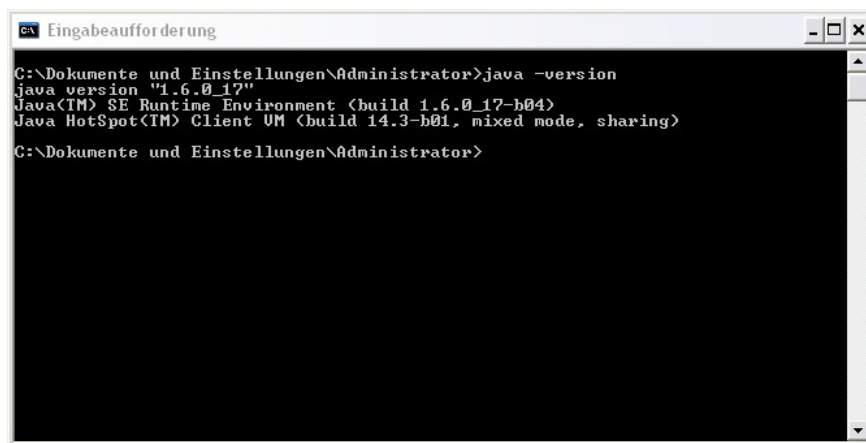


Abbildung 13 (Verifikation der Java Installation)

Ist Java für Windows noch nicht installiert oder nicht aktuell so ist die Installation an dieser Stelle auf jeden Fall nachzuholen, bzw. ein Update durchzuführen. (Details siehe [Jav09]).

Nachdem das Eclipse Ganymede SR2 C/C++ ZIP Archiv von der Eclipse CDT Homepage [Ecl09] heruntergeladen wurde gestaltet sich das Setup denkbar einfach, da es sich dabei nicht um ein Setup im eigentlichen Sinne handelt, sondern lediglich um das Entpacken des Archivs. Daher muss nun die Archivdatei „*eclipse-cpp-ganymede-SR2-win32.zip*“ lediglich an einem nahezu beliebigen Ort entpackt werden (es empfiehlt sich hierfür jedoch einen Ort wie z.B. „C:“ oder „C:\Programme“ zu verwenden).

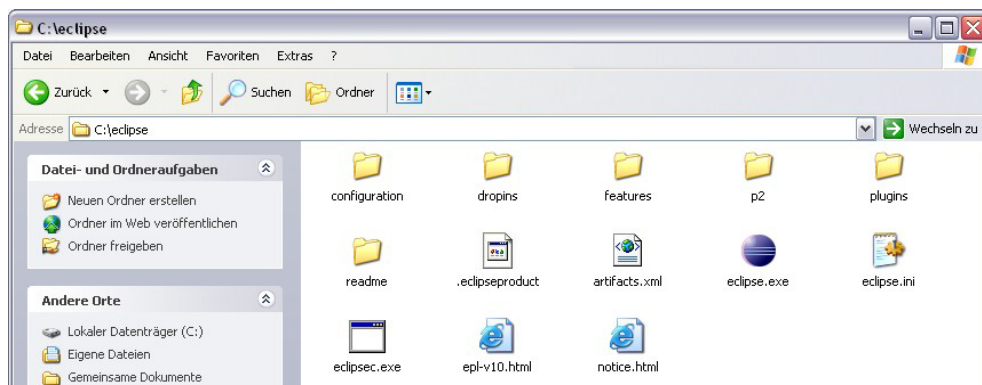


Abbildung 14 (Eclipse Programmordner)

In diesem Tutorial wurde das Archiv nach „C:“ entpackt, weshalb nach dem Entpacken ein Ordner namens „eclipse“ im Laufwerk C:\ mit dem in Abbildung 14 dargestelltem Inhalt existiert.

Es empfiehlt sich eine Verknüpfung mit der Datei „eclipse.exe“ anzulegen, und diese an einem komfortablen Ort zur späteren Benutzung abzulegen (z.B. durch einen Rechtsklick auf „eclipse.exe → senden an → Desktop (Verknüpfung erstellen)“). Nun kann die Eclipse IDE erstmals durch die soeben erstellten Verknüpfung oder durch starten von „eclipse.exe“ im Eclipse Programmordner gestartet werden. Bevor die Eclipse IDE aber vollständig geladen wurde, wird man dazu aufgefordert einen sogenannten Workspace zu wählen.

Bei dem Eclipse Workspace handelt es sich um das Verzeichnis in dem in weiterer Folge alle Projekte sowie einige Parameter der Eclipse IDE abgelegt werden, weshalb es empfehlenswert ist einen Pfad zu wählen der frei von Leerzeichen oder problematischen Sonderzeichen ist (Underscores „_“ sind allerdings kein Problem). Aus diesem Grund wurde in diesem Tutorial der Pfad „C:\eclipse_workspace“ gewählt. Wenn man nur sehr selten oder so gut wie nie den zu verwendenden Workspace wechseln muss/will, so kann man diesen Dialog durch wählen der Option „Use this as the default and do not ask again“ in Zukunft unterdrücken (siehe Abbildung 15).

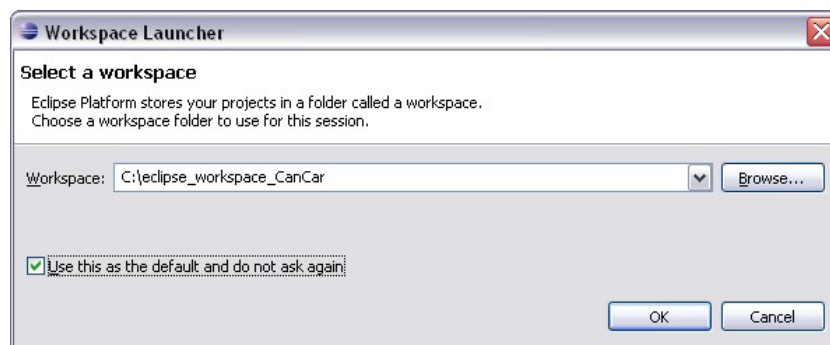


Abbildung 15 (Auswahl des Eclipse Workspace)



Abbildung 16 (Eclipse Willkommen Seite)

Nachdem der Startvorgang abgeschlossen ist wird die Willkommen Seite der Eclipse IDE angezeigt. Diese kann aber sofort durch einen Klick auf „Workbench“ (siehe Abbildung 16)

verlassen werden, wodurch die C/C++ Standard Ansicht (Perspective) gezeigt werden sollte (siehe Abbildung 17)

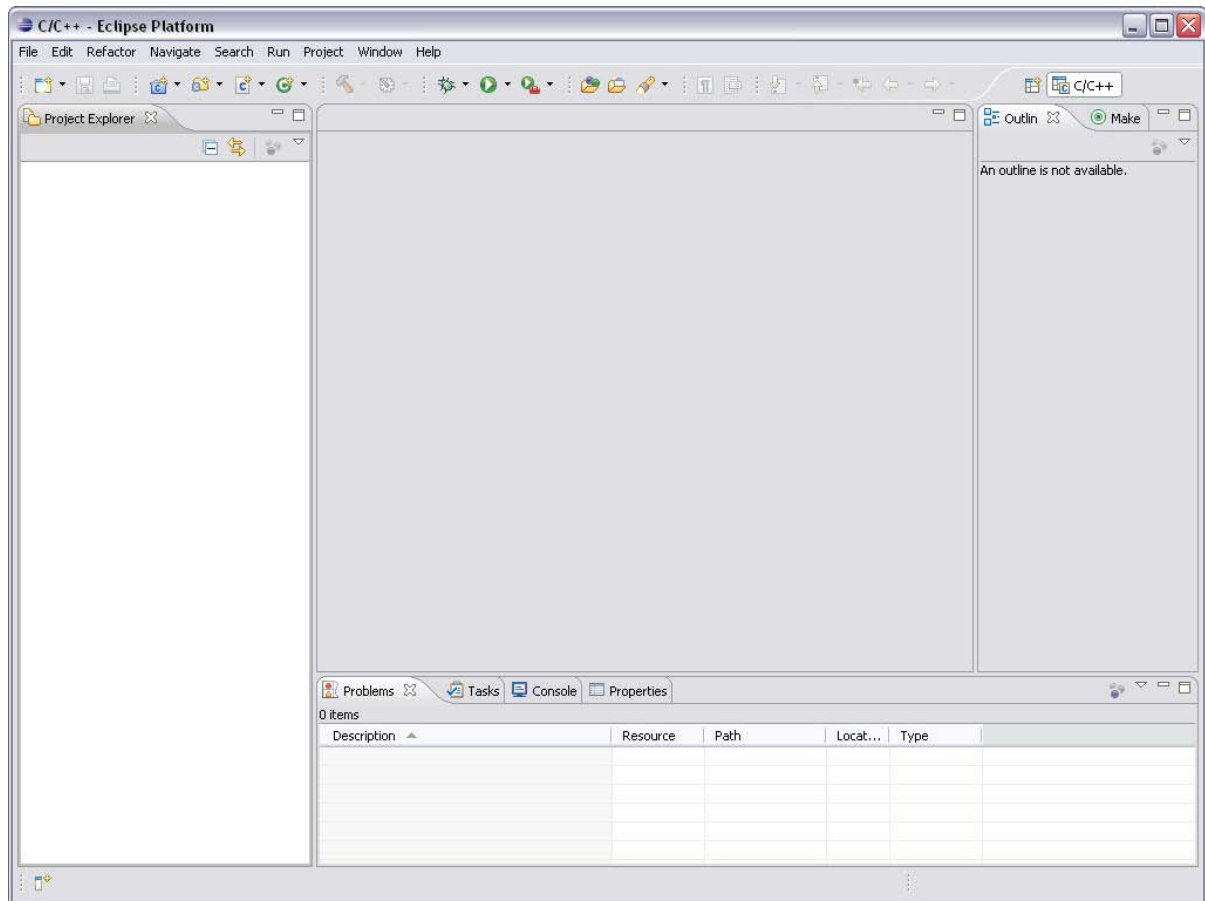


Abbildung 17 (Eclipse Standard C/C++ Ansicht)

1.6 Installation der nötigen Eclipse Plug-ins

Zum jetzigen Zeitpunkt ist die Eclipse IDE noch nicht in der Lage mit dem CodeSourcery GNU Compiler bzw. openOCD oder freeRTOS zusammenzuarbeiten. Daher müssen als erstes die entsprechenden Plug-ins hinzugefügt werden um die Eclipse IDE entsprechend zu erweitern.

1.6.1 GNU ARM Eclipse sowie Zylind CDT Plug-in

GNU ARM Eclipse Plug-In

Dieses Plug-In verknüpft die Eclipse IDE mit dem CodeSourcery GNU Compiler, sowie mit anderen ARM GNU Compilern sofern diese ebenfalls bereits installiert wurden. Das Plug-in kann auf der Projekthomepage [GAEP10] als ZIP Archiv für eine offline Installation heruntergeladen werden, oder aber direkt über eine Eclipse Update Site installiert werden.

Zylin CDT Plug-In

Dieses Plug-In stellt die noch fehlende Brücke zwischen der Eclipse IDE, dem CodeSourcery GDB und dem open OCD Server her. Es kann, wie im Folgenden gezeigt, über eine Eclipse Update Site installiert werden.

Zur Installation beider Plug-Ins muss die Eclipse IDE gestartet werden und der Menüpunkt „*Help* → *Software updates...*“ gewählt werden (siehe Abbildung 18).

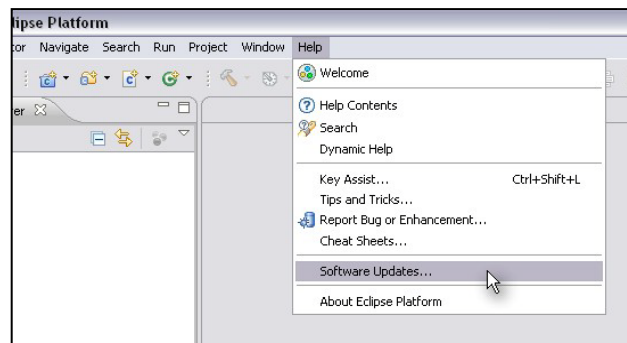


Abbildung 18 (Starten des Eclipse Software Update Managers)

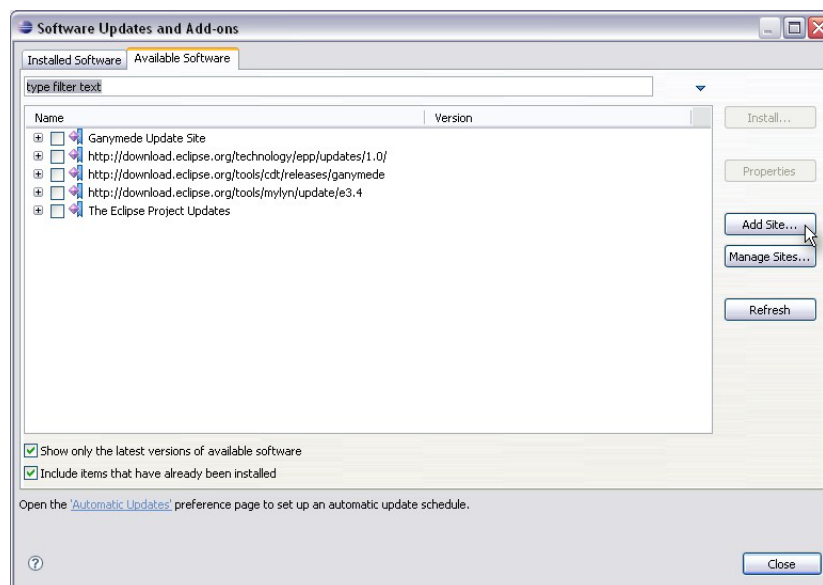


Abbildung 19 (Eclipse Software Update und Add-on Dialog)

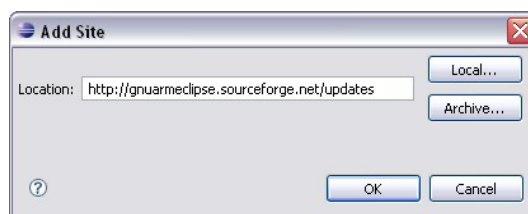


Abbildung 20 (Hinzufügen einer Eclipse Update Site)

In dem nun geöffneten Fenster (siehe Abbildung 19) muss in der Registerkarte „*Available Software*“ der Button „*Add Site*“ gewählt werden, um im folgenden Dialog (siehe Abbildung

20) eine Eclipse Update Site anzugeben. Die erste Update Site die hinzugefügt werden soll ist die des GNU ARM Eclipse Plugins die wie folgt lautet:

`http://gnuarmclipse.sourceforge.net/updates`

Der Dialog kann nun durch Klicken auf „Ok“ geschlossen werden, um die zweite für das Zylind CDT Plug-In nötige Update Site hinzuzufügen. Dies wird auf dieselbe Art und Weise wie zuvor bewerkstelligt, allerdings lautet die Adresse der Update Site wie folgt:

`http://opensource.zylin.com/zylincdt`

Wurde auch diese Update Site hinzugefügt, und der Dialog wieder mit „Ok“ Bestätigt, so sind in der Liste der verfügbaren Update Sites zwei neue Einträge vorhanden (siehe Abbildung 21), welche nun wie abgebildet ausgewählt werden und durch einen Klick auf den Button „Install...“ installiert werden.

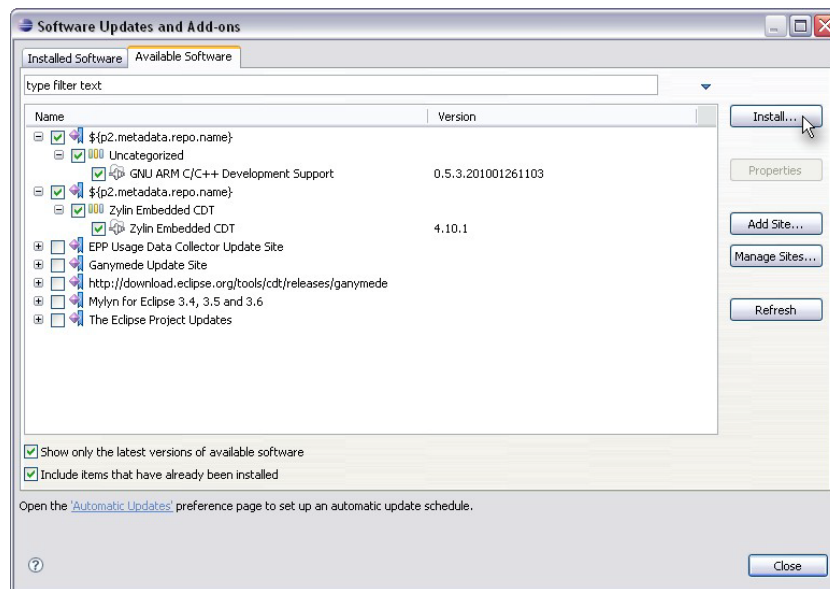


Abbildung 21 (Auswahl der hinzugefügten Eclipse Update Sites)

Nach einer eventuell etwas längeren Wartezeit (es werden Abhängigkeiten zwischen den verschiedenen Eclipse Plug-Ins geprüft) soll der folgende Dialog mit einem Klick auf „Next“ bestätigt werden. Danach müssen die Lizenzbestimmungen akzeptieren um das Setup abzuschließen. Ist das Setup abgeschlossen wird man zu einem Neustart der Eclipse IDE aufgefordert, welcher auch durchgeführt werden sollte.

Nach dem Neustart kann man die Installation des GNU ARM Eclipse Plug-ins durch wählen des Menüpunkts „File → New → C Project“ überprüft werden. Ist alles korrekt installiert, so sollte es im folgenden Dialog (siehe Abbildung 22) die Projekt Typen „ARM Cross Target Application“ sowie „ARM Cross Target Static Library“ geben, und die Toolchains „GNUARM, WinWARM, Yagarto“ sowie „Sourcery G++ Lite“ auswählbar sein (Dialog mit „Cancel“ wieder verlassen).

Die Installation des Zylind CDT Plug-Ins wird durch öffnen des „*Debug Configurations*“ Dialogs (im Menü „*Run* → *Debug Configurations...*“) überprüft. Ist alles korrekt installiert so sollte es in der Liste der verfügbaren Debug Konfigurationen die beiden Einträge „*Zylin Embedded debug (cygwin)*“ sowie „*Zylin Embedded debug (native)*“ geben (siehe Abbildung 23):

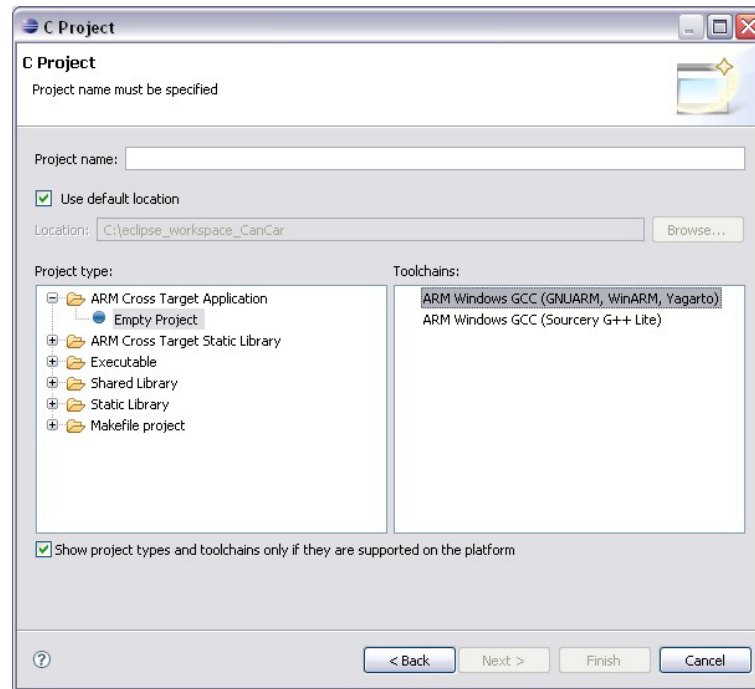


Abbildung 22 (Verifikation der GnuARM Eclipse Plug-in Installation)

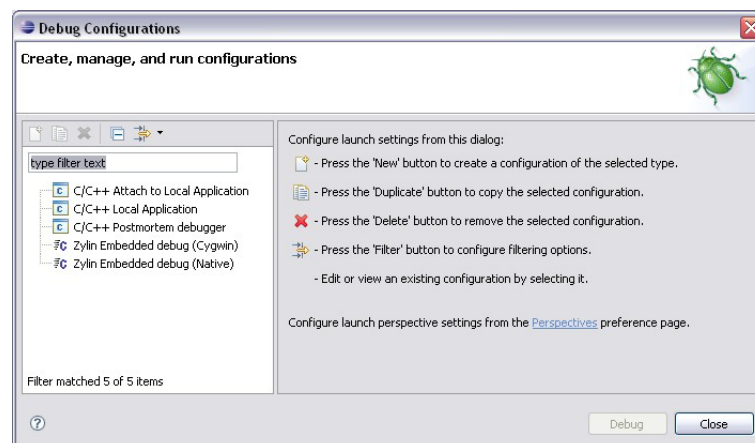


Abbildung 23 (Verifikation des Zylin Embedded CDT Plug-Ins)

1.6.2 FreeRTOS Stateviewer Plug-in (optional)

Ist die Entwicklung von Echtzeitanwendungen geplant, welche ein entsprechendes Echtzeitbetriebssystem benötigen, so kann für das freie Echtzeitbetriebssystem freeRTOS (siehe [RTOS10]) ein zusätzliches Plug-In installiert werden welches beim Debuggen solcher Applikationen behilflich sein kann. Dieses Plug-In wird von der Firma Wittenstein

HighIntegritySystems nach einer schnellen und unkomplizierten Registrierung gratis zur Verfügung gestellt (siehe [Wit10]).

Ist der Einsatz eines Echtzeitbetriebssystems nicht geplant, so kann dieser Schritt übersprungen werden und mit Kapitel 2 fortgefahren werden. Falls zu einem späteren Zeitpunkt ein entsprechender Bedarf entstehen sollte, so kann dieses Plug-In gemäß der folgenden Anleitung auch nachträglich installiert werden.

Im Ersten Schritt muss man sich auf der Homepage des Plug-Ins (siehe [Wit10]) registrieren um zur eigentlichen Downloadseite zu gelangen. Dort muss das Plug-In für die Eclipse Umgebung heruntergeladen werden, wobei es sich im Gegensatz zu den meisten anderen Eclipse Plug-Ins um eine EXE Datei („STATEVIEWER for Ecipse v107.exe“) handelt. Ist der Download abgeschlossen so wird die Installation des Plug-Ins durch ausführen der eben heruntergeladenen Datei gestartet.

Nachdem der erste Dialog mit „Next“ übersprungen wurde, werden im zweiten Dialog dieser Installation Hinweise für die Integration in die Eclipse IDE angezeigt, was ebenfalls mit „Next“ Übersprungen werden kann. Danach müssen noch die Lizenzbedingungen akzeptiert werden und der Installationspfad angegeben werden. Dieser soll unverändert bleiben, weshalb durch einen Klick auf „Next“ die Installation beginnt. Nach dem Abschluss der Installation muss nun der Installationspfad im Windows Explorer geöffnet werden (der Standard Pfad lautet: „C:\HighIntegritySystems\Eclipse STATEVIEWER“). In diesem Ordner befindet sich ein Unterordner namens „plugins“ in dem sich das eigentliche Plug-In in Form einer JAR Datei befindet. Diese Datei muss nun in den Ordner „plugins“ der gewünschten Eclipse Installation kopiert werden, wobei es ratsam wäre die Eclipse IDE zuvor zu schließen.

Befindet sich die Eclipse Installation z.B. im Ordner „C:\Eclipse“ so muss die angesprochene JAR Datei („rtos.openrtos.viewer_1.0.7.jar“) in den Ordner „C:\Eclipse\plugins“ kopiert werden. Beim nächsten Start der Eclipse IDE wird das neue Plug-In automatisch erkannt.

Ist die Eclipse IDE neu gestartet, so muss das neue Plug-In noch zur Standard Debug Ansicht hinzugefügt werden. Dazu wird als erstes die Debug Perspective geöffnet (im Menü der Eclipse IDE: „Window → Open Perspective → Debug“, siehe Abbildung 24)

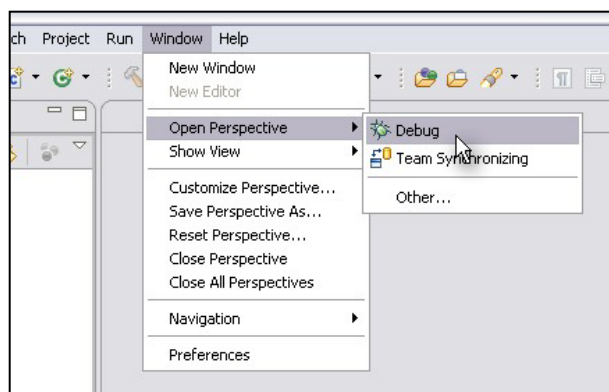


Abbildung 24 (Öffnen der Debug Perspective)

Nun müssen die Zusatzansichten, welche durch das freeRTOS Stateviewer Plug-In hinzugekommen sind geöffnet werden. Dazu wird im Menü der Eclipse IDE unter „Window → Show View → Other...“ (siehe Abbildung 25) ein Dialog geöffnet der alle verfügbaren

Ansichten anzeigt. In diesem Dialog muss nun in der Kategorie „*OpenRTOS Viewer*“ die „*Queue Table*“ geöffnet werden (siehe Abbildung 26). Da immer nur eine Ansicht geöffnet werden kann sind diese Schritte für die „*Task Table*“ Ansicht zu wiederholen.

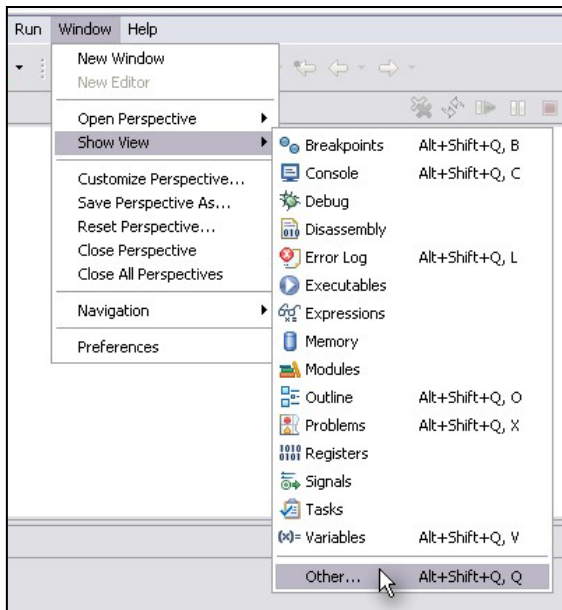


Abbildung 25 (öffnen des Show View Dialogs)

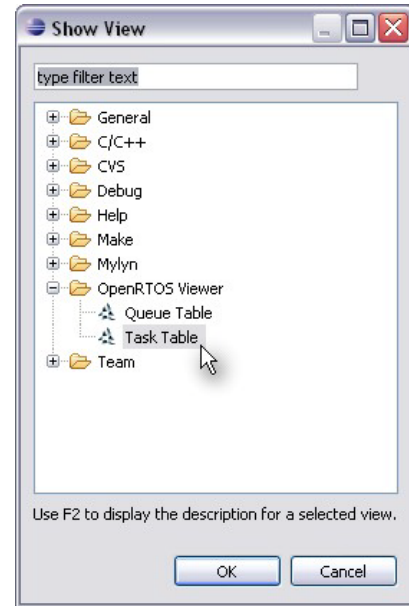


Abbildung 26 (Auswahl der neuen Ansichten)

Nun sollten die neuen Ansichten (wie in Abbildung 27 zu sehen) dargestellt werden, wobei diese erst dann von Nutzen sind wenn eine auf einem FreeRTOS basierende Applikation am Mikrocontroller debuggt wird.

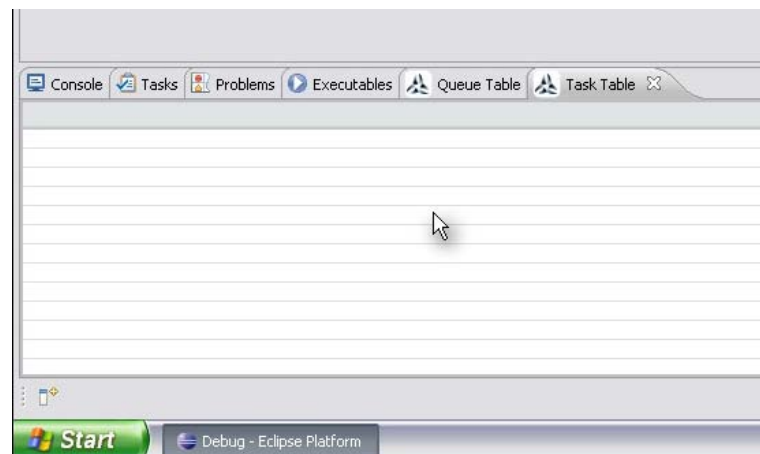


Abbildung 27 (FreeRTOS Stateviewer Ansichten in der Debug Perspective)

Um das tun zu können müssen allerdings noch ein paar Einstellungen in der Eclipse Umgebung gesetzt werden, wobei wir nun zur Konfiguration der Eclipse IDE übergehen.

2 Konfiguration

Da es sich bei openOCD um eine Konsolenanwendung ohne Eclipse Plug-In handelt, muss nun eine Verbindung zwischen der Eclipse IDE und openOCD geschaffen werden die eine komfortable Benutzung ermöglicht. Dazu wird als erstes der „*External Tools Configurations*“ Dialog geöffnet, indem im Eclipse Hauptmenü „*Run → External Tools → External Tools Configurations...*“ ausgewählt wird (siehe Abbildung 28).

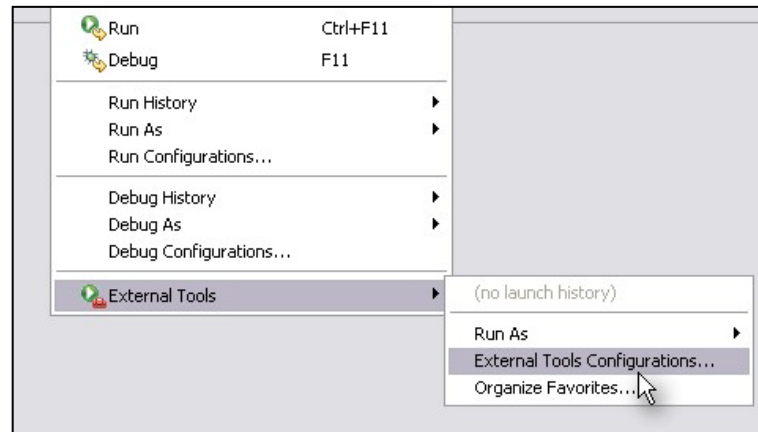


Abbildung 28 (Öffnen des External Tools Configurations Dialogs)

In diesem Dialog wird eine neue Konfiguration angelegt indem der „*New*“ Button ausgewählt wird (siehe Abbildung 29), dazu muss aber zuvor der Punkt Program ausgewählt werden.

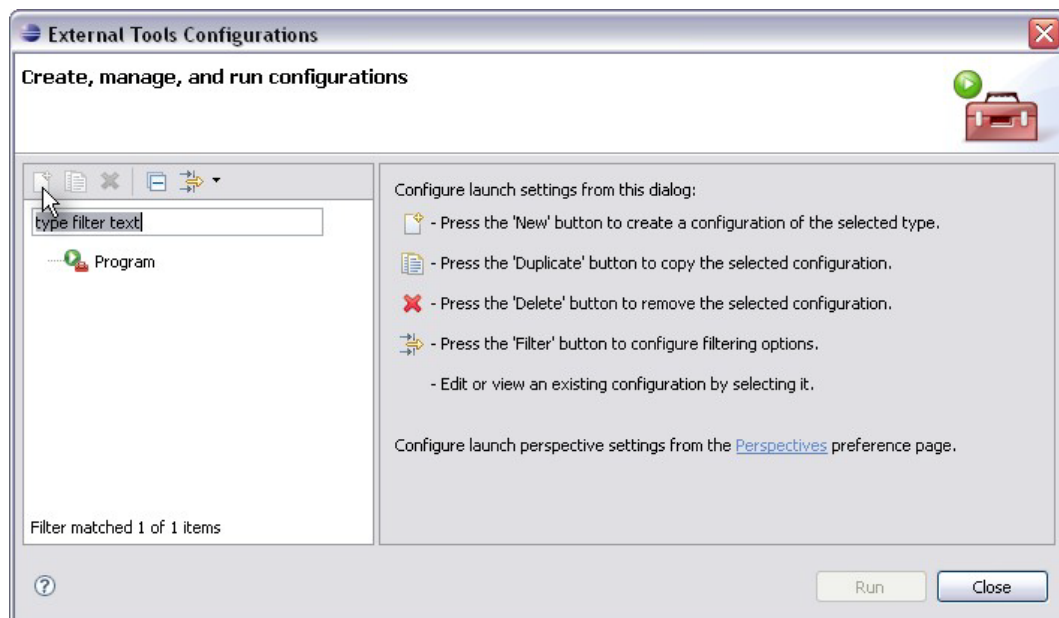


Abbildung 29 (Anlegen einer neuen Konfiguration)

Nun muss die Konfiguration wie in Abbildung 30 dargestellt eingestellt werden. Dazu wird als erstes ein sinnvoller Name für diese Konfiguration gewählt. Da es sich bei dieser Konfiguration in weiterer Folge um den openOCD Server zum Debuggen handelt, wurde die

Bezeichnung „*openOCD – Server*“ gewählt. Als „*Location*“ muss die Datei „*openocd.exe*“ im bin Ordner der openOCD Installation angegeben werden.

Als Working Directory wird die folgende Eclipse Variable angegeben:

```
${workspace_loc}/${project_name}/openOCD}
```

Sie verweist auf den Ordner openOCD im jeweils aktiven Projekt im gerade genutzten Eclipse Workspace, welches zwar im Moment noch nicht existiert weil ja noch kein Projekt angelegt wurde, aber das spielt im Moment noch keine Rolle.

Als Arguments wird folgender String eingetragen:

```
-f openocd_stm32.cfg
```

Er gibt an, dass openOCD ein externes Skriptfile ausführen soll das „*openOCD_Server.cfg*“ heißt, welches dann später im openOCD Unterordner des jeweiligen Projekts zu finden sein wird (auch hier bitte ich den Leser um Geduld, Details folgen im Kapitel 3).

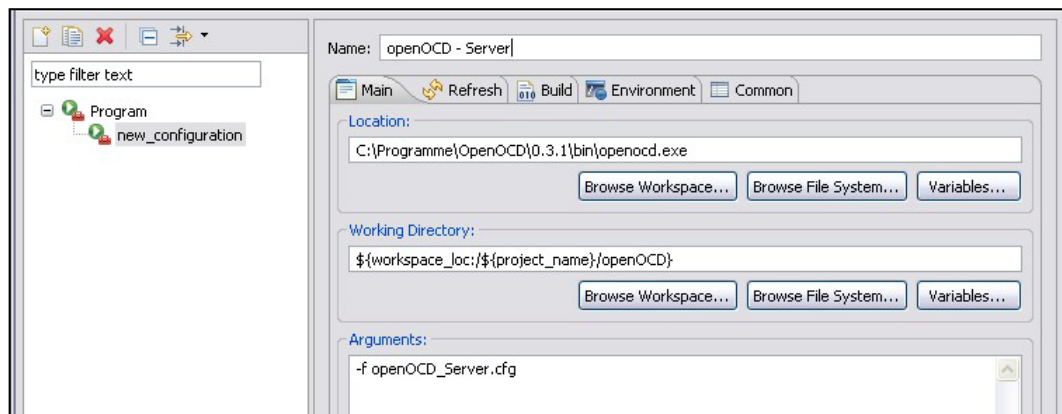


Abbildung 30 (Konfiguration des Extenen Tools (1 von 2))

Sind diese Felder wie angegeben ausgefüllt wurden, so sollte noch in der Registerkarte „*Build*“ die Option „*build before launch*“ deaktiviert werden, wobei das stark vom Geschmack des letztendlichen Users abhängt, da so eingestellt werden kann ob das Projekt vor dem Ausführen des Tools neu kompiliert werden soll. Inder Registerkarte „*Common*“ sollte noch die Option „*External Tools*“ im Fensterteil „*Display in favorites menu*“ gewählt werden um den Start des Tools zu vereinfachen (siehe Abbildung 31).

Danach wird die Konfiguration durch einen Klick auf „*Apply*“ gespeichert und zwei Mal kopiert, indem der Button „*Duplicate*“ zwei Mal gedrückt wird (dieser Button ist neben dem „*New*“ Button, siehe Abbildung 29). Danach sollte es insgesamt drei Konfigurationen geben, wie in Abbildung 32 zu sehen ist.

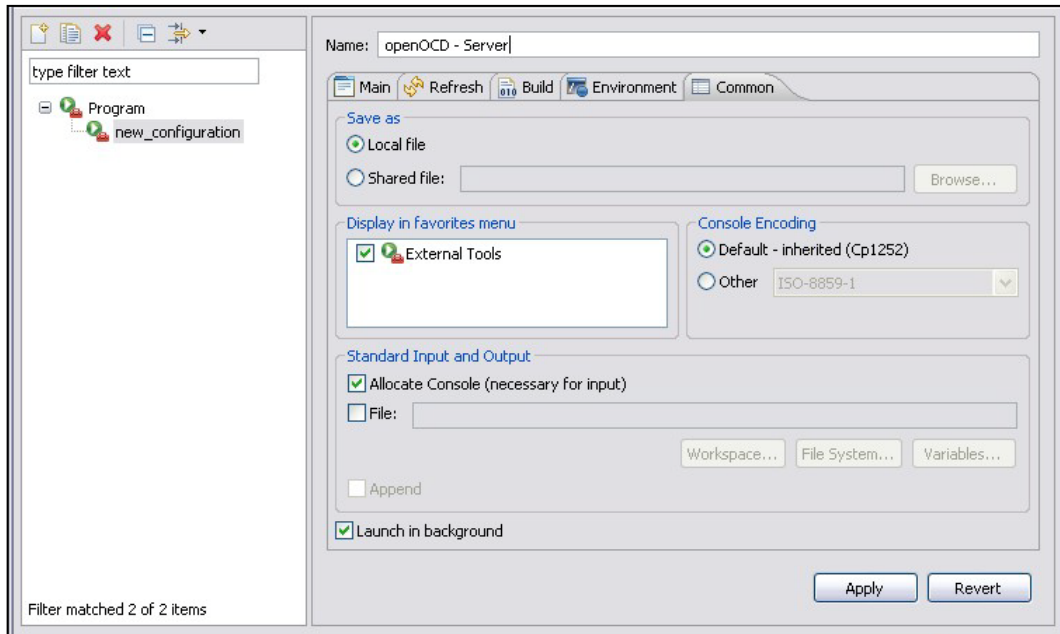


Abbildung 31 (Konfiguration des Extenen Tools (2 von 2))

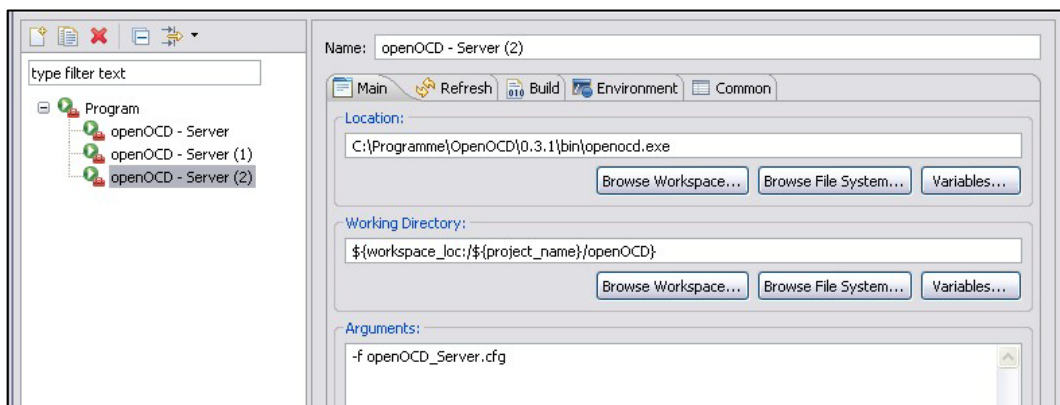


Abbildung 32 (Kopierte Konfigurationen)

Da diese Konfigurationen alle gleich sind (bis auf den Namen) und man insgesamt drei leicht unterschiedliche Konfigurationen benötigt (eine für den openOCD Server und jeweils eine zum Flashen der Debug- und Release Programmversion), müssen diese nun umbenannt werden und die jeweiligen Argumente angepasst werden. Dabei wird eine der eben kopierten Konfigurationen in „*openOCD – Flash Debug*“ umbenannt indem das „*Name*“ Feld entsprechend angepasst wird. Zudem muss das „*Arguments*“ Feld angepasst werden sodass es folgenden Eintrag enthält:

```
-f flash_stm32_debug.cfg
```

Nachdem die Änderungen wieder mit „*Apply*“ übernommen wurden wird die zweite Konfiguration in „*openOCD – Flash Release*“ umbenannt und ihr Arguments Feld wie folgt geändert:

```
-f flash_stm32_release.cfg
```

Die Änderungen werden wie immer mit einem Klick auf „*Apply*“ übernommen, wonach der „*External Tools Configurations*“ Dialog den in Abbildung 33 dargestellten Inhalt anzeigen sollte. Danach kann das Dialogfenster mit einem Klick auf „*Close*“ geschlossen werden.

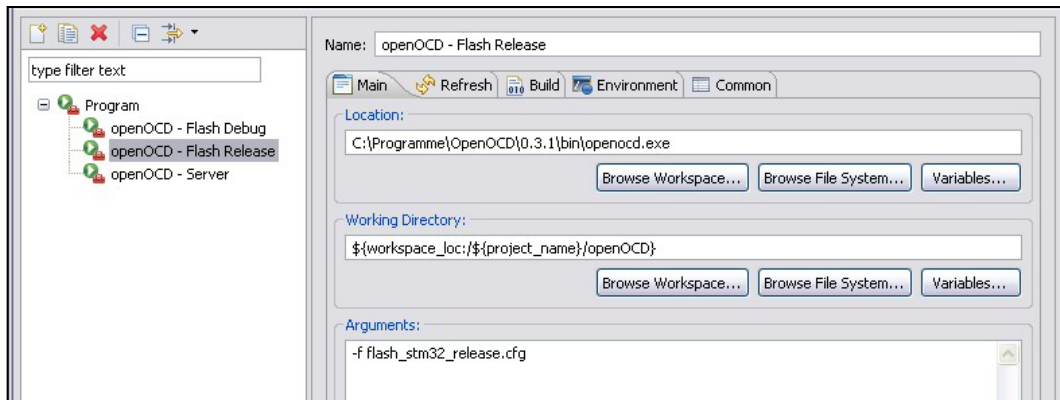


Abbildung 33 (Erstelle Tool Konfigurationen)

Nun sollten sich die erstellten Konfigurationen im Untermenü des „*External Tools*“ Symbols in der Eclipse Symbolleiste befinden, wie es in Abbildung 34 dargestellt ist.

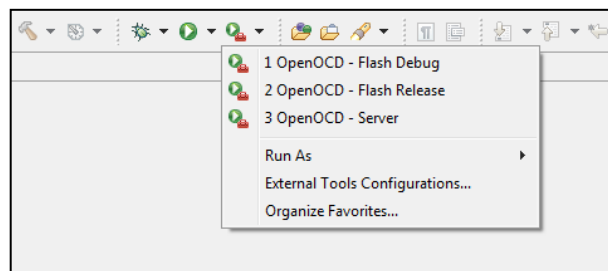


Abbildung 34 (External Tools in der Symbolleiste)

Als letzten kann je nach Geschmack des Users noch die Funktion „*Build Automatically*“ deaktiviert werden, welche dafür sorgt dass die Projekte im Workspace ständig automatisch kompiliert werden. Sie ist im Hauptmenü unter dem Menüpunkt „*Project*“ zu finden. Wie gesagt obliegt es jedem User dieses Feature zu deaktivieren, die Erfahrung hat aber gezeigt dass diese Funktion dem Komfort eher abträglich werden kann.

Damit sind die allgemeinen Konfigurationsarbeiten vor dem Anlegen erster Beispiele abgeschlossen und es wird Zeit für ein klassisches blink Led Beispiel.

3 Beispiel Anwendung

3.1 Ein neues Projekt anlegen

Um ein neues Projekt anzulegen muss im Hauptmenü der Eclipse IDE „*File → New → Project*“ gewählt werden. Daraufhin öffnet sich ein Dialogfenster, in dem in der Kategorie „C“ der Unterpunkt „*C Projekt*“ gewählt werden muss. Nach einem Klick auf „*Next*“ muss in der Kategorie „*ARM Cross Target Application*“ der Unterpunkt „*Empty Project*“, sowie die „*ARM Windows GCC (Sourcery G++ Lite)*“ Toolchain ausgewählt werden. Zudem braucht das neue Projekt auch einen adäquaten Namen, weshalb auf den Namen „*blinky*“ getauft werden soll (Siehe Abbildung 35).

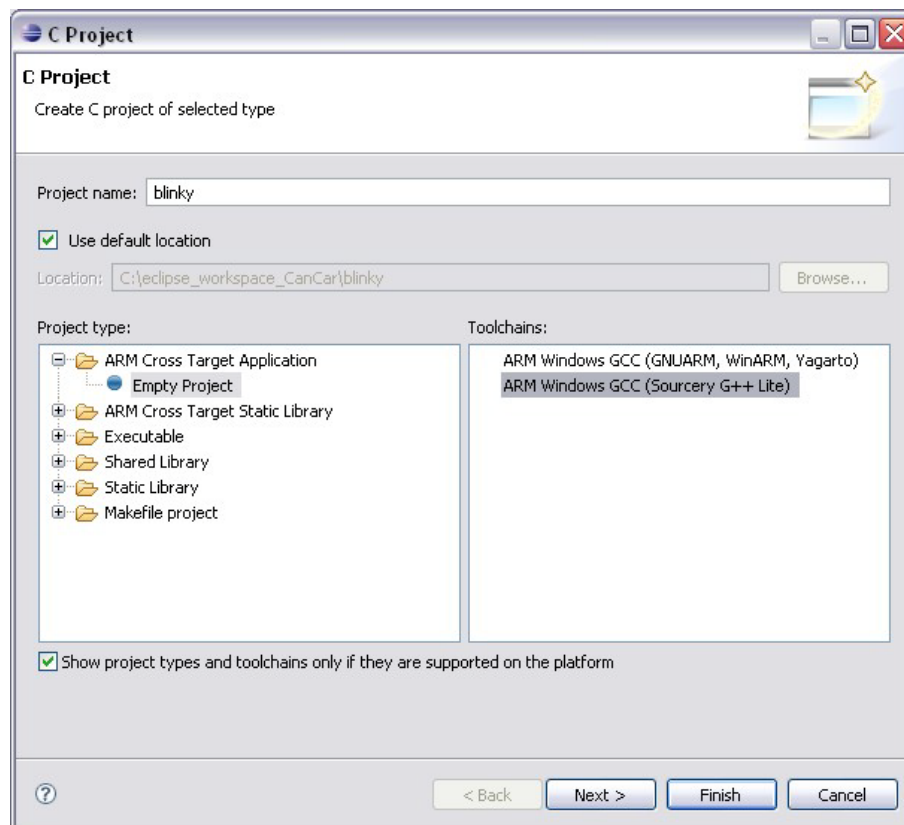


Abbildung 35 (Anlegen eines Neuen ARM Projekts)

Dieser Dialog kann nun mit einem Klick auf „*Finish*“ abgeschlossen werden, woraufhin das Projekt erstellt wird und im Projektexplorer angezeigt wird. Eventuell öffnet sich ein Dialogfenster in dem die Eclipse IDE vorschlägt die C/C++ Perspective zu aktivieren da dies die Standartansicht für ein derartiges Projekt ist. Dies geschieht nur wenn diese Ansicht nicht aktiv war als das neue Projekt angelegt wurde und solle in dem Fall auch mit einem Klick auf „*Yes*“ beantwortet werden.

Da ein Projekt für einen Arm Cortex M3 innerhalb dieser Toolchain aus mehreren Dateien besteht (Startupcode, Linker Skripte, openOCD Skripte, usw.) wäre es zu aufwendig jede Zeile von jedem File in dieses Tutorial zu integrieren, weshalb der in diesem Tutorial benutzte Quellcode in einem Zipfile zur Verfügung gestellt wird. In diesem Archiv gibt es einen

Ordner Namens „*blinky*“, und dessen Inhalt soll nun vollständig in den gleichnamigen Ordner im benutzten Eclipse Workspace kopiert werden. Praktischerweise unterstützt die Eclipse IDE Drag and Drop, weshalb die Dateien direkt aus dem Archiv in das Projekt im Projektexplorer geschoben werden können. Danach sollten folgende Dateien im Projekt enthalten sein:

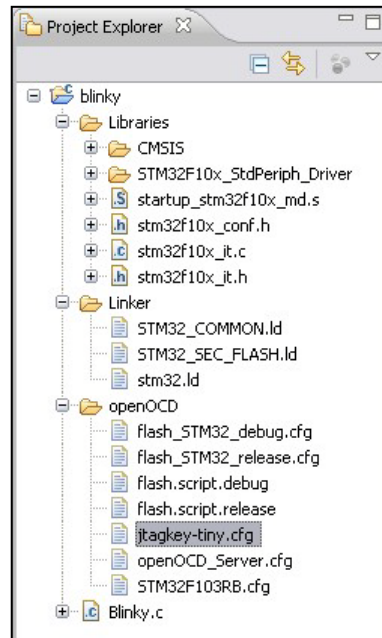


Abbildung 36 (blinky Projektdateien)

Im Ordner „*Libraries*“ befindet sich der Startupcode („*startup_stm32f10x_md.s*“) und die von der STM Standard Peripheral Library benötigten Dateien. Die STM Standard Peripheral Library wird vom Hersteller des Mikrocontrollers gratis zur Verfügung gestellt und enthält alle Funktionen die zur Benützung der in dem Mikrocontroller enthaltenen Peripherien notwendig bzw. nützlich sind. Dazu gehört der bereits angesprochene Startupcode, notwendige Interruptserviceroutinen und so weiter. Diese Library kann auf der STM Homepage (siehe [STM10b]) jederzeit heruntergeladen werden, weshalb nur die vom blinky Projekt benötigten Dateien der STM Library in dem beigefügten Archiv enthalten sind.

Im Ordner „*Linker*“ befinden sich die nötigen Linkerskripte und im Ordner „*openOCD*“ befinden sich die bereits angesprochenen openOCD Skriptfiles.

3.2 Anpassen der openOCD Skripte

Zwei dieser Skriptfiles müssen aber noch an den Pfad des Eclipse Workspace angepasst werden, weshalb zuerst die Datei „*flash.script.debug*“ geöffnet werden soll.

In dieser Datei muss der Pfad zur hex Datei eventuell angepasst werden (je nachdem wo sich der Eclipse Workspace befindet). Wichtig ist hierbei jedoch dass nur der im Folgenden markierte Teil des Pfades angepasst werden soll, da der Projektunterordner „*debug*“ erst beim Kompilieren erstellt werden wird.

```
flash write_image erase C:/eclipse_workspace/blinky/debug/blinky.hex ihex
```


Selbiges gilt für die Datei „*flash.script.release*“ welche für das Flashen der kompilierten Release hex Datei zuständig ist.

```
flash write_image erase C:/eclipse_workspace/blinky/release/blinky.hex ihex
```

3.3 Das neue Projekt Konfigurieren

Nun müssen noch die Projekteinstellungen angepasst werden, was durch einen Rechtsklick auf den Projektordner „*blinky*“ im Projektexplorer und Auswahl von „*Properties*“ geschieht. Daraufhin öffnet sich ein Dialogfenster in dem auf der linken Seite die Kategorie „*C/C++ Build* → *Settings*“ ausgewählt werden soll (siehe Abbildung 37).

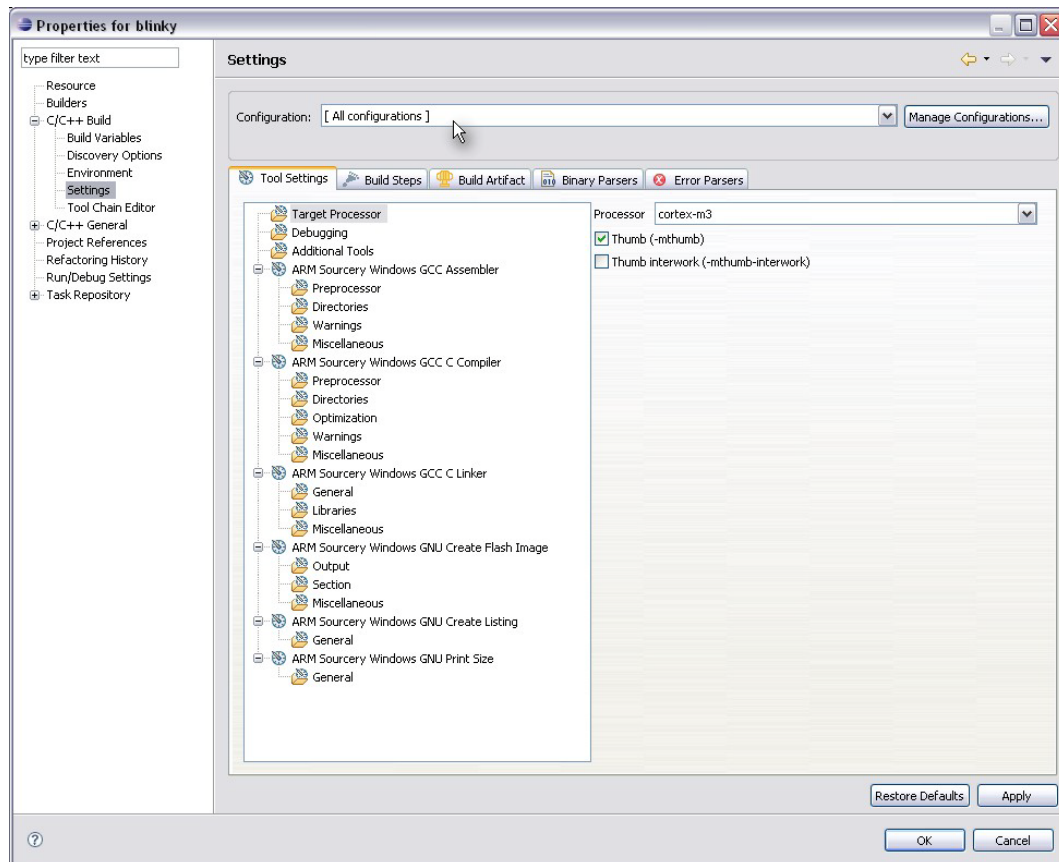


Abbildung 37 (Projekteinstellungen)

Als erstes wird unter „*Configurations*“ die Option „*All configurations*“ ausgewählt, um gleichzeitig für die Debug- und Release Konfiguration alle wichtigen Einstellungen zu treffen. Im Feld „*Target Processor*“ ist bereits der richtige Prozessor (cortex-m3) für dieses Projekt ausgewählt (siehe Abbildung 37), weshalb als nächstes die Kategorie „*ARM Sourcery windows GCC C Compiler* → *Directories*“ ausgewählt wird. Hier müssen die in diesem Projekt benutzten Include Pfade angegeben werden, was durch klicken auf das „*Add...*“ Symbol erledigt werden kann. In dem sich öffnenden „*Add directory path*“ Dialog wird der Button „*Workspace...*“ gewählt und der Unterordner „*Libraries*“ im „*blinky*“ Projekt angegeben.

Nachdem dieser Dialog mit „Ok“ geschlossen wurde werden auf diese selbe Art und Weise zwei weitere Unterordner des Projekts hinzugefügt:

```
/Libraries/CMIS  
/Libraries/STM32F10X_StdPeriph_Driver/inc
```

Das letztendliche Ergebnis ist in Abbildung 38 zu sehen:

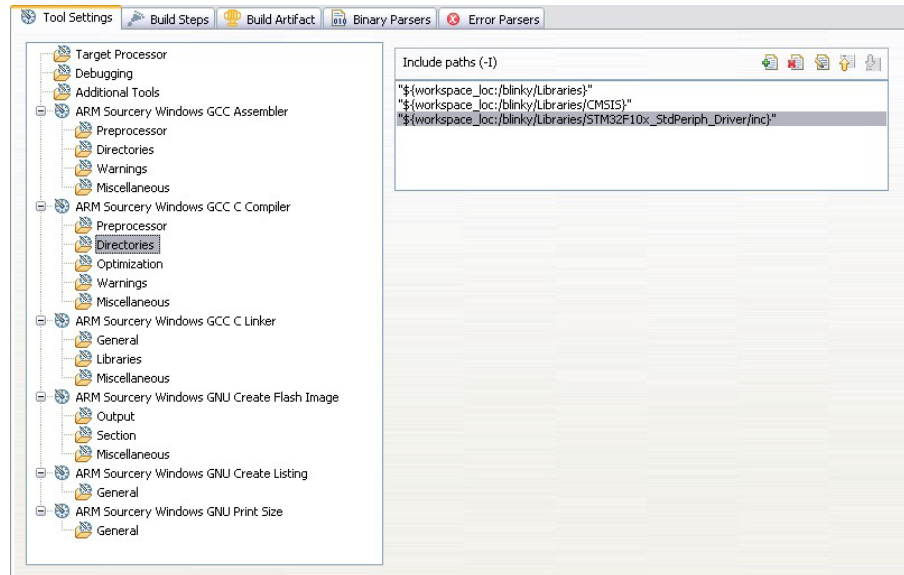


Abbildung 38 (Angabe der benötigten Include Pfade)

Ist man mit dem Angeben der Includepfade fertig, so wird als nächstes die Kategorie „*ARM Sourcery windows GCC C Compiler* → *Warnings*“ ausgewählt um die Option „*Pedantic*“ zu aktivieren, wir wollen ja sauberen Code schreiben (siehe Abbildung 39).

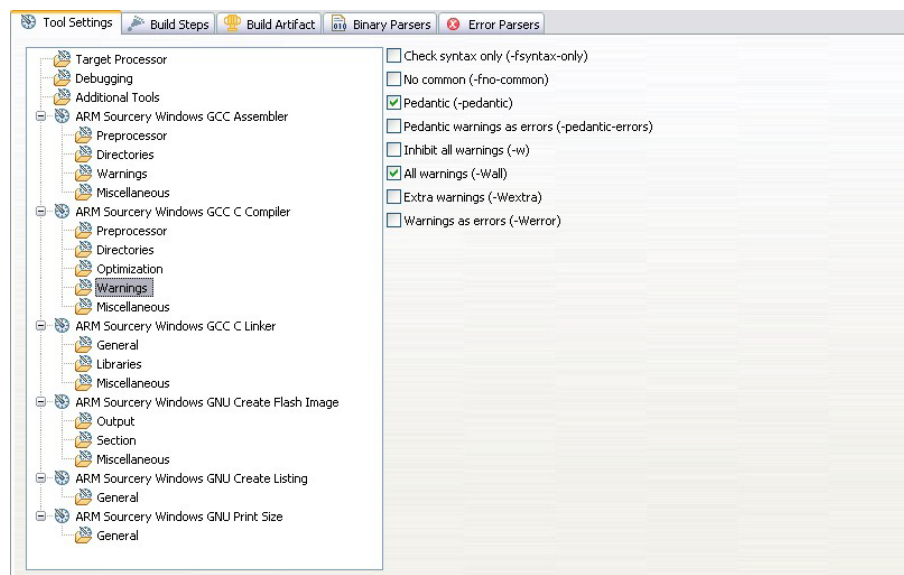


Abbildung 39 (C Compiler Einstellungen)

Ist auch das erledigt, wenden wir uns der Kategorie „*ARM Sourcery windows GCC Linker* → *General*“ zu um das zu verwendende Linker Skript anzugeben. Um sich das Leben leichter zu machen empfiehlt es sich den Button „*Browse*“ zu wählen, um im sich daraufhin öffnenden Dialogfenster die Datei „*stm32.ld*“ anzugeben die sich im Projektunterordner „*Linker*“ befindet. Das Endergebnis ist in Abbildung 40 dargestellt.

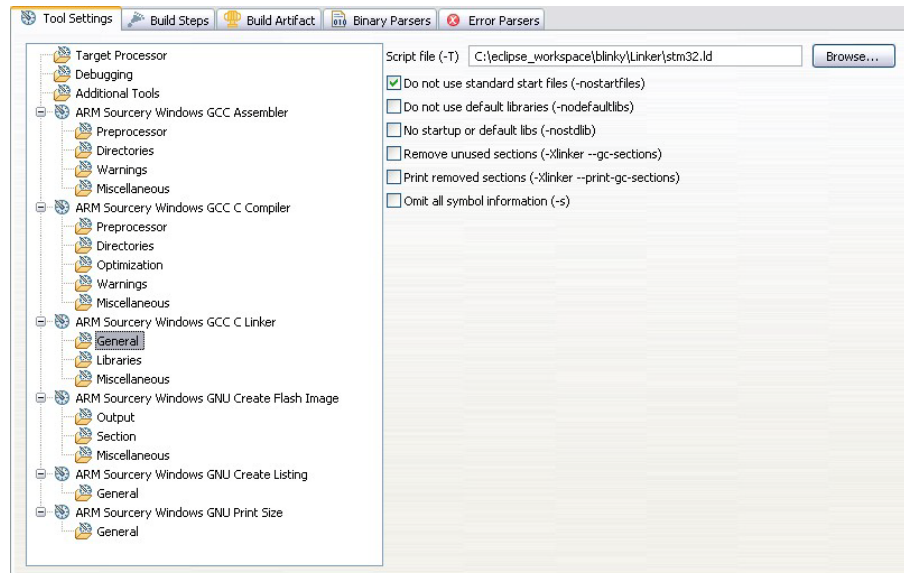


Abbildung 40 (Angabe des Linker Skripts)

Ist das erledigt bleibt nur noch eins zu tun, und zwar muss dem Linker noch mitgeteilt werden wo sich die weiteren Linkerskripte befinden auf welche im zuvor angegebenen Linkerskript verwiesen wird. Dazu muss zuerst die Kategorie „*ARM Sourcery windows GCC Linker* → *Libraries*“ ausgewählt werden, und dort als „*Library search path*“ der Order „*linker*“ im Projektordner angegeben werden. Dies geschieht durch einen klick auf das „*Add...*“ Symbol (wie schon zuvor bei den Include pfaden), wobei der Pfad wieder relativ zum Eclipse Workspace angegeben wird. Das Endergebnis ist in Abbildung 41 dargestellt.

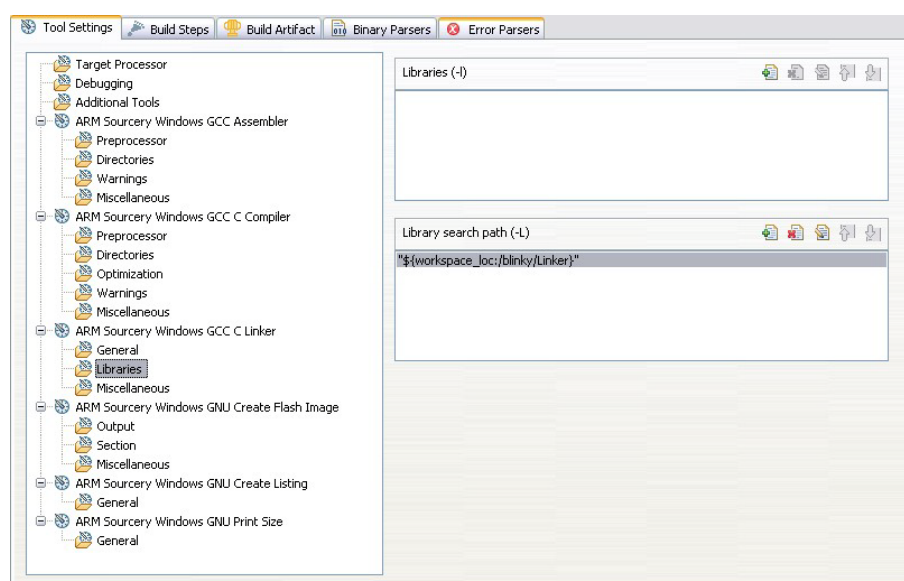


Abbildung 41 (Angabe der Linker Librarys)

Nun ist es an der Zeit alle angegebenen Pfade und Einstellungen durch einen Klick auf „Apply“ zu speichern und den Dialog durch einen Klick auf „Ok“ zu schließen womit die Konfiguration der Projekteinstellungen abgeschlossen ist.

3.4 Kompilieren des Projekts

Nach all den Mühen der Installation und Konfiguration kommen wir nun zum eigentlichen Sinn und Zweck des ganzen Aufwands. Um das Projekt zu Kompilieren wird als erstes eine C Datei des Projekts geöffnet (z.B. „*blinky.c*“). Das ist nötig da eine zum Projekt gehörige Datei im Editor geöffnet sein muss um Kompilieren zu können. Im Moment erscheint das zwar unsinnig, aber wenn man später an mehreren Projekten gleichzeitig arbeiten sollte ist diese Funktion praktisch da dann bei einem Klick auf Kompilieren immer das Projekt kompiliert wird an dem man gerade arbeitet, selbst wenn mehrere Projekte offen sind.

Bevor das Projekt nun durch einen Klick auf das „Build“ Symbol in der Eclipse Symbolleiste kompiliert (siehe Abbildung 42) wird, kann im Dropdown Menü dieses Symbols (kleiner Pfeil nach unten, rechts neben dem Symbol) die Projektkonfiguration ausgewählt werden. Fürs Erste soll nun die „Debug“ Konfiguration ausgewählt werden (sollte bereits standardmäßig aktiv sein).

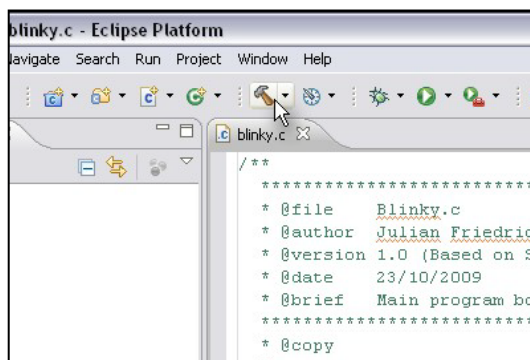


Abbildung 42 (Kompilieren des Projekts)

Daraufhin ist im Konsolenfenster der Output des Compiler und des Linker zu sehen was im Fehlerfall hilfreich sein kann. Ist das Projekt erfolgreich kompiliert, so sollte schlussendlich der in Abbildung 43 dargestellte Inhalt der Eclipse Konsole zu sehen sein.

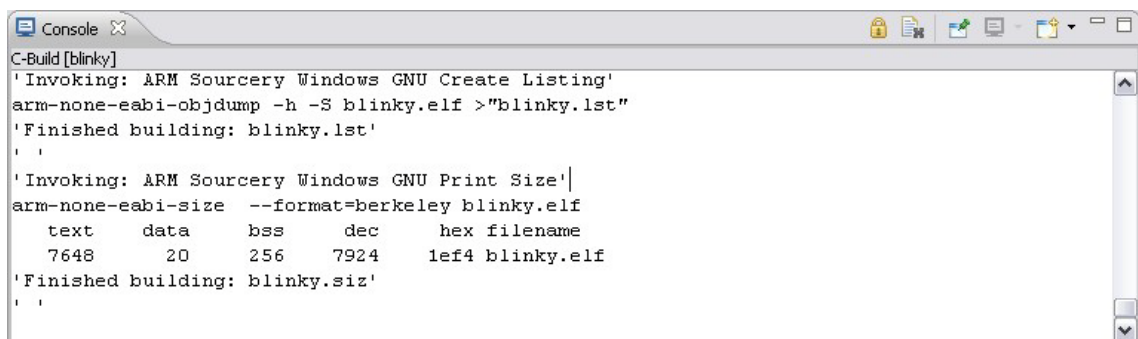


Abbildung 43 (Build Protokoll nach einem erfolgreichen Buildprozess)

Sollte beim Build etwas schief gegangen sein, befindet sich im Anhang das Protokoll eines erfolgreichen Build Durchlaufs, was mit dem eigenen Protokoll verglichen werden kann um den Fehler ein wenig einzugrenzen.

Nun ist es Zeit das Projekt in den Mikrocontroller zu übertragen um es ausführen und/oder debuggen zu können.

3.5 Download in den Flash

Sind alle Pfade in den openOCD Skript Dateien wie im Kapitel 3.2 beschrieben angepasst und das Projekt erfolgreich kompiliert worden, so muss nun lediglich die bereits angelegte openOCD Konfiguration Namens „*openOCD – Flash Debug*“ gestartet werden (siehe Abbildung 44).

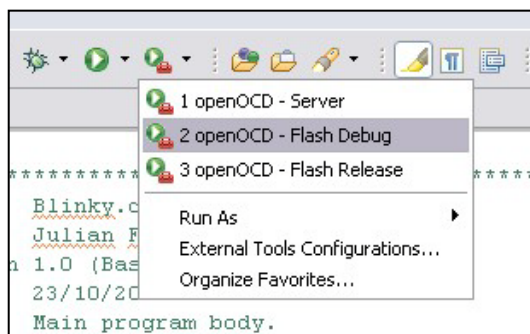


Abbildung 44 (Download der Applikation)

Daraufhin ist in der Konsole der openOCD Output zu sehen, welcher diesmal im Gegensatz zum Compiler in rot gehalten ist (keine Sorge das alleine ist kein Hinweis auf einen Fehler). Nach dem Ende des Vorgangs sollte der in Abbildung 45 dargestellte Output zu sehen sein, zur Kontrolle befindet sich der gesamte openOCD Output des Flash Vorgangs im Anhang.

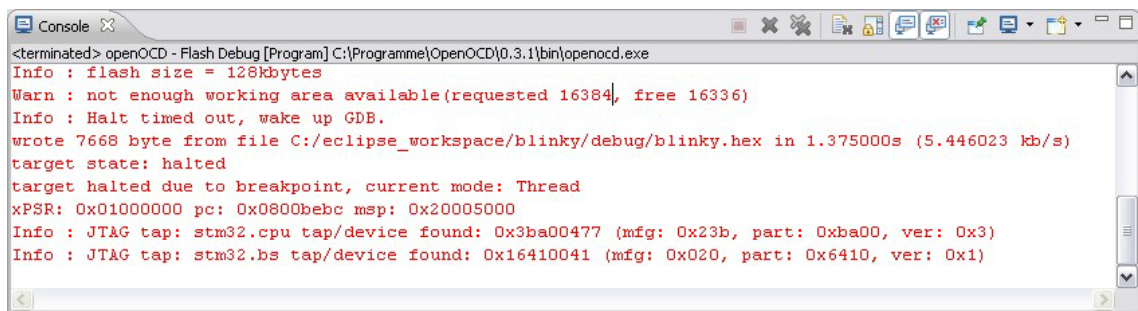


Abbildung 45 (openOCD Output nach dem Flashen)

Im letzten Teil des Tutorials soll nun demonstriert werden wie diese einfache Anwendung debuggt werden kann.

3.6 Debuggen der Applikation

3.6.1 Anlegen einer Debug Konfiguration

Um das Projekt debuggen zu können ist als erstes eine Debug Konfiguration anzulegen. Diese wird erstellt in dem im Hauptmenü der Eclipse IDE der Menüpunkt „*Run → Debug Configurations...*“ gewählt wird. In diesem Fenster wird als erstes auf der linken Seite die Kategorie „*Zylin Embedded debug (native)*“ gewählt wird, und dann eine neue Konfiguration erstellt wird, indem der New Button betätigt wird (siehe Abbildung 46).

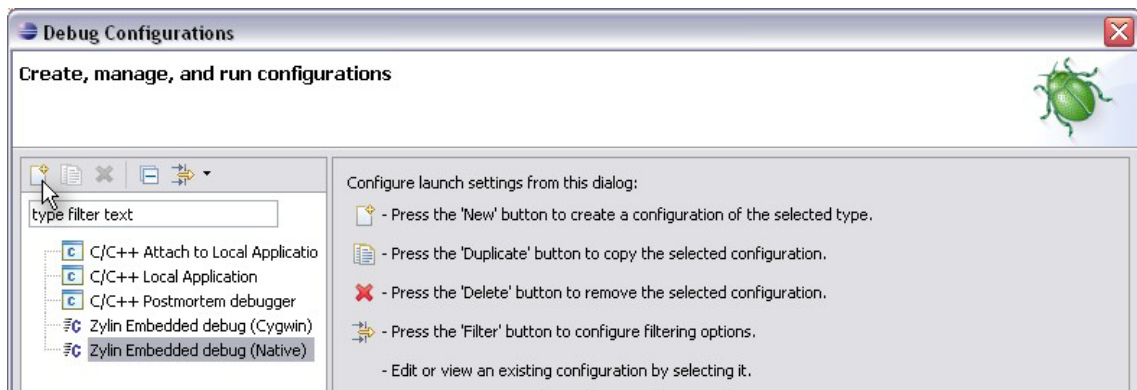


Abbildung 46 (Anlegen einer neuen Debug Konfiguration)

Dieser neuen Konfiguration muss als erstes ein passender Name gegeben werden. Da die Eclipse IDE diese Debug Konfiguration für jedes Projekt separat abspeichert, jedoch mehrere Projekte gleichzeitig geöffnet werden dürfen, müssen sich die Namen aller Debug Konfigurationen unterscheiden. Daher sollte der Name wie folgt aufgebaut werden:

Debug - <Projektname>

Die Debug Konfiguration wird in dem Fall laut obigen Schema „*Debug – blinky*“ genannt. Als nächstes muss in der „*Main*“ Registerkarte der Debug Konfiguration das zu debuggende Projekt angegeben werden sowie die C/C++ Applikation (siehe hierzu Abbildung 47). Letzteres geschieht durch einen klick auf „*Search Project...*“, Auswahl der „*blinky.elf*“ Datei und schließen des Dialogs mit „*Ok*“.

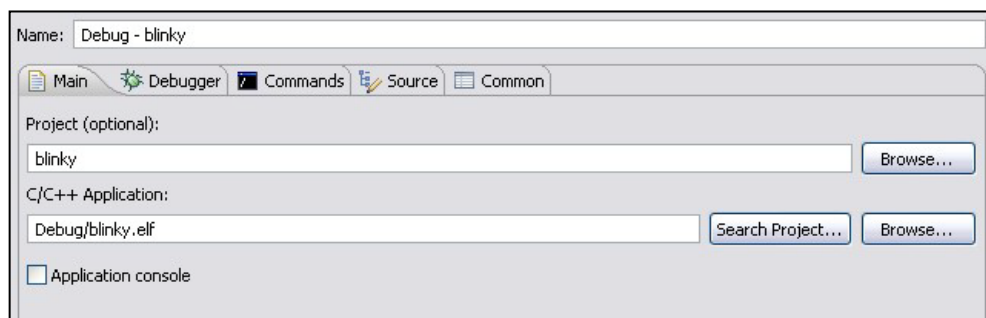


Abbildung 47 (Einstellungen der Debug Konfiguration (1 von 4))

Im nächsten Schritt muss in der Registerkarte Debugger der CodeSourcery Gnu Debugger ausgewählt werden welcher sich im bin Unterordner der CodeSourcery Installation befindet (die Datei trägt den Namen „*arm-none-eabi-gdb.exe*“). Zudem muss der Eintrag im „*GDB*

command file“ Feld entfernt werden. Zur Kontrolle sind die zu treffenden Einstellungen in der Registerkarte „*Debugger*“ in Abbildung 48 abgebildet.

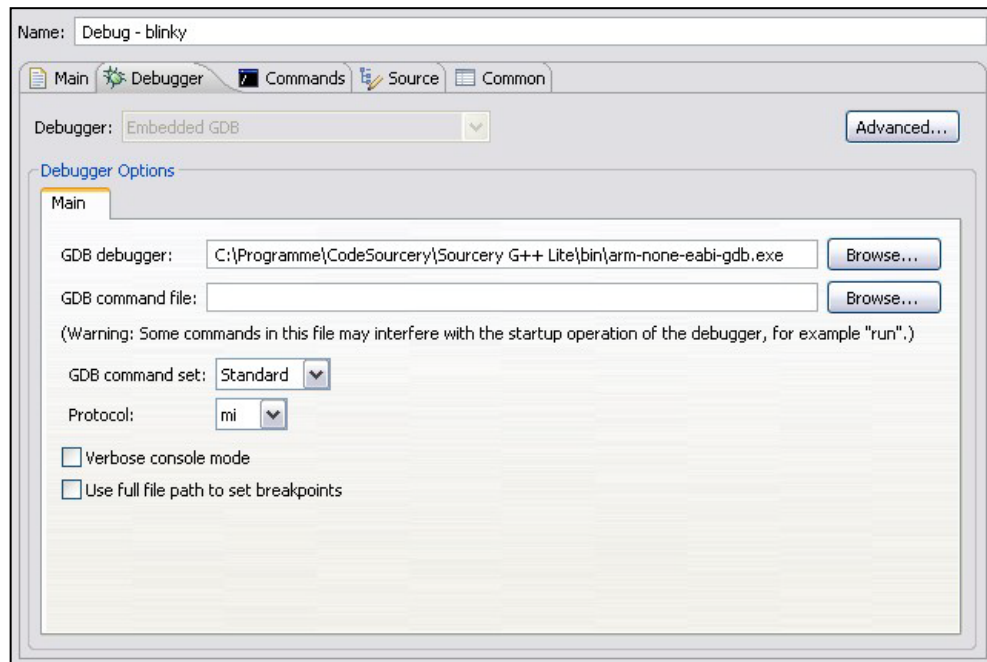


Abbildung 48 (Einstellungen der Debug Konfiguration (2 von 4))

Im nächsten Schritt müssen die zur Initialisierung des Gnu Debuggers notwendigen Initialisierungsbefehle in der Registerkarte „*Commands*“ angegeben werden (siehe Abbildung 49). Die anzugebenden Kommandos lauten wie folgt:

```
target extended-remote localhost:3333
b main
monitor soft_reset_halt
monitor sleep 500
continue
clear main
```

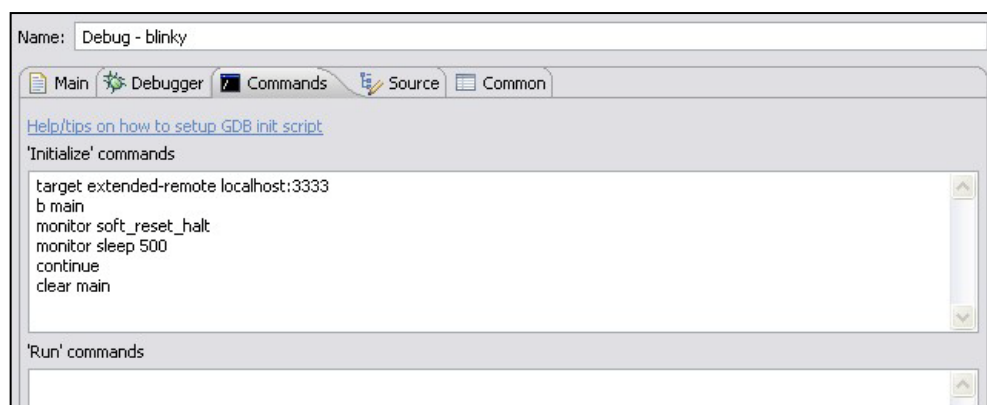


Abbildung 49 (Einstellungen der Debug Konfiguration (3 von 4))

Als Letztes muss in der Registerkarte „*Common*“ die Option „*Debug*“ im „*Display in favorites menu*“ Fensterteil aktiviert werden (siehe Abbildung 50).

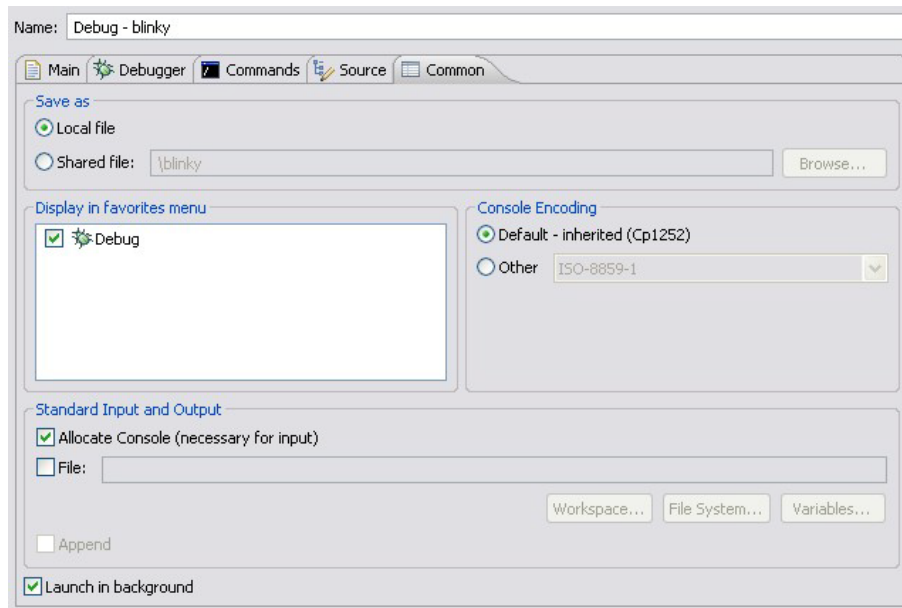


Abbildung 50 (Einstellungen der Debug Konfiguration (4 von 4))

Danach müssen die Angaben mit einem Klick auf den Button „*Apply*“ gespeichert werden und der Dialog durch Klicken auf „*Close*“ geschlossen werden.

3.6.2 Starten einer Debug Session

Ist wie zuvor angegeben eine Debug Konfiguration angelegt, so kann mit dem Debuggen begonnen werden. Zunächst muss aber noch der openOCD Server gestartet werden, was wie zuvor beim Flashen des Mikrocontrollers durch Auswahl der entsprechenden openOCD Konfiguration geschieht (siehe Abbildung 51)

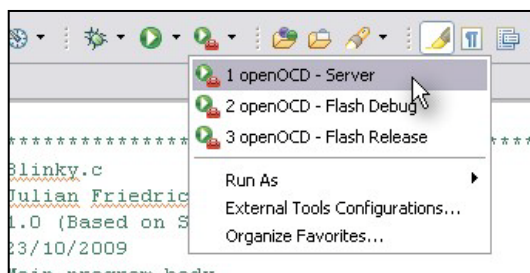


Abbildung 51 (Starten des openOCD Servers)

Der openOCD Server wird daraufhin gestartet, und sollte in der Eclipse Konsole den in Abbildung 52 dargestellten Output erzeugen.

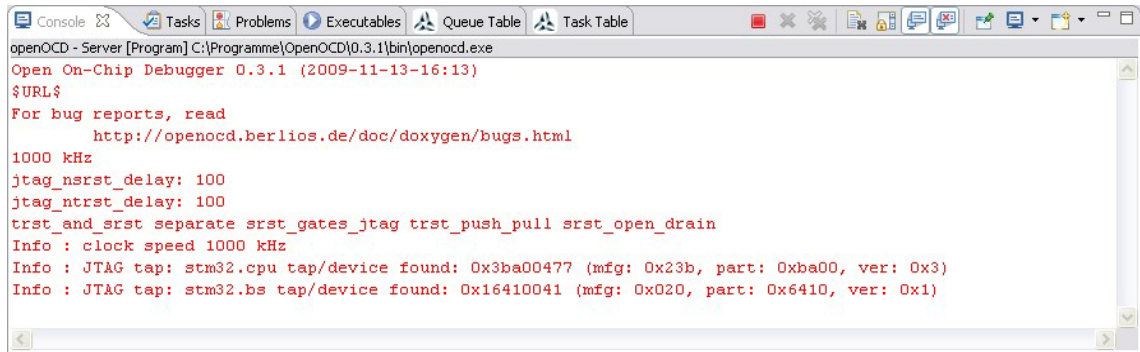


Abbildung 52 (openOCD Server Output)

Danach muss noch der eigentliche Debugger gestartet werden indem die vorhin erstellte Debug Konfiguration wie in Abbildung 53 gezeigt gestartet wird.

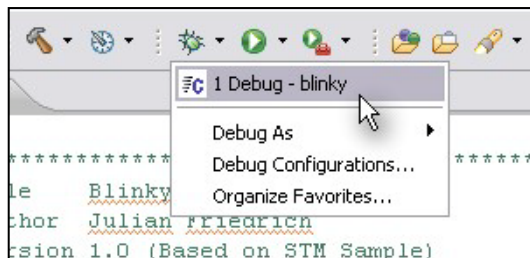


Abbildung 53 (Starten des Debuggers)

Die Eclipse IDE schlägt daraufhin vor, dass die Debug Perspective angezeigt werden soll da dies die Standardansicht während dem Debuggen ist. Das ist jedenfalls mit „Yes“ zu beantworten da nur so die für das Debuggen notwendigen Symbolleisten verfügbar sind.

Nun sollte sich die Eclipse IDE wie in Abbildung 55 dargestellt präsentieren. Im Fenster oben links steht die Kontrolle über den Debugger und die externen Tools zur Verfügung, rechts daneben werden die Variablen der aktuellen Funktion dargestellt. Im unteren Teil werden die Ausgaben des Debuggers angezeigt, und im mittleren Teil auf der linken Seite wird der Quellcode und die Position an der sich der Debugger gerade befindet angezeigt.

Abbildung 54 zeigt die Symbolleiste mit Hilfe derer der Debugger gesteuert werden kann, ein Klick auf das „Play“ Symbol startet das Programm, wer mag kann es aber auch im Einzelschrittmodus durch die Applikation steppen (entweder mit Hilfe der Symbole, oder der Tasten F5 („step into“) bzw. F6 („step over“)).



Abbildung 54 (Symbolleiste zur Kontrolle des Debuggers)

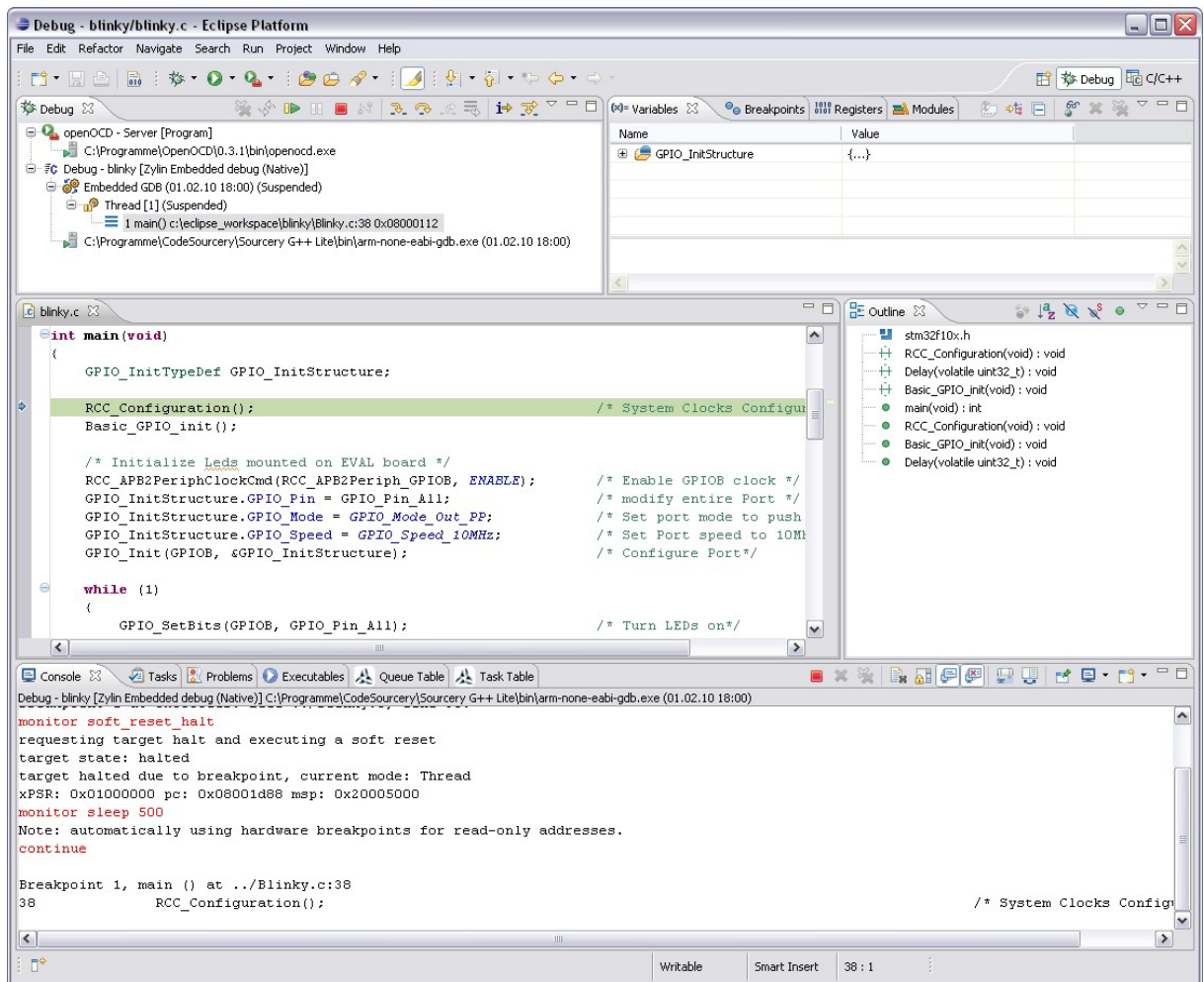


Abbildung 55 (Eclipse Debug Perspective)

3.6.3 Beenden einer Debug Session

Ist man mit dem Debuggen fertig so muss zunächst der Debugger beendet werden, bevor auch der openOCD Server beendet werden kann. Dies wird durch Auswahl der Debug Konfiguration („*Debug – blinky [Zylin Embedded debug (Native)]*“) und einen anschließen Klick auf das „*Stopp*“ Symbol in der Debug Symbolleiste bewerkstelligt (siehe Abbildung 56), danach wird mit dem openOCD Server genauso verfahren.

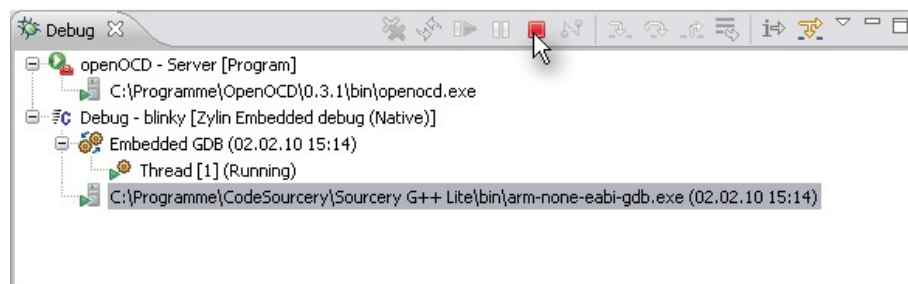


Abbildung 56 (Beenden einer Debug Session)

Sind beide Prozesse terminiert (siehe Abbildung 57) so kann die Debug Perspective durch einen Klick auf das C/C++ Perspective Symbol verlassen werden um wieder auf die gewohnte Ansicht zum editieren des Quellcodes umzuschalten (siehe Abbildung 58).

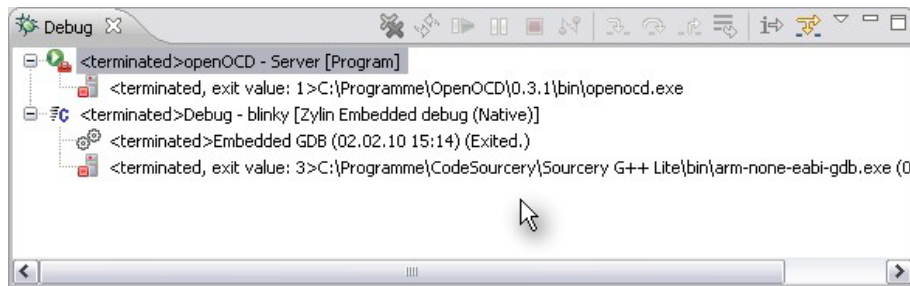


Abbildung 57 (Terminierte Debug Prozesse)

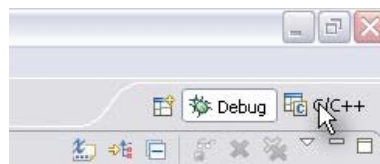


Abbildung 58 (Umschalten auf die C/C++ Perspective)

4 How To ...

An dieser Stelle sei noch auf die wichtigsten Dinge beim Debuggen bzw. Entwickeln hingewiesen:

4.1 Code Editor

4.1.1 Code Completion bzw Content Assist

In der Eclipse IDE wird das Feature das viele User als Code Completion kennen Content Assist genannt (Als Hinweis für die Suche in der Eclipse Hilfe). Zur Benutzung dieser Funktion muss beim Schreiben des Quellcodes einfach die Kombination „*STRG* + *Leertaste*“ gedrückt werden um ein Popup mit Vorschlägen aufzurufen. Mit den Pfeiltaten kann dann in den Vorschlägen navigiert werden, zur Reduktion der Vorschläge (wenn das was bis jetzt getippt wurde zu allgemein ist) einfach weiterschreiben, das Popup passt sich laufend an.

4.1.2 Folding Aktivieren

Die Optionen welche das (automatische) Zusammenklappen von Kommentaren und oder Funktionen usw. konfigurieren werden im Menü unter „*Window* → *Preferences*“ und in dem sich öffnenden Fenster in der Rubrik „*C/C++* → *Editor* → *Folding*“ gefunden. En der Rubrik Editor befinden sich die Einstellungen der meisten Komfort Funktionen

4.1.3 Automatisch Speichern vor einem Build

Wird in der Eclipse IDE ein Build Prozess gestartet, so werden geöffnete und nicht gespeicherte Dateien Standardmäßig NICHT gespeichert, was bedeutet das nicht das kompiliert wird was im Editor zu sehen ist. Wer das ändern möchte muss im Menü unter „*Window* → *Preferences*“ und dort unter „*General* → *Workspace*“ die Option „*Save automatically before build*“ aktivieren. Bei der Gelegenheit empfiehlt es sich auch die Option „*Refresh automatically*“ zu aktivieren um den Projektexplorer immer aktuell zu halten.

4.1.4 Suchfunktion

Ein weiteres sehr sinnvolles Feature der Eclipse IDE ist die integrierte Suchfunktion. Sie kann durch einen klick auf die Taschenlampe in der Eclipse Symbolleiste aufgerufen werden und ermöglicht eine Projektübergreifende suche, wobei die Ergebnisse in einem such baum dargestellt werden, wodurch der gesucht Eintrag schnell gefunden werden kann.

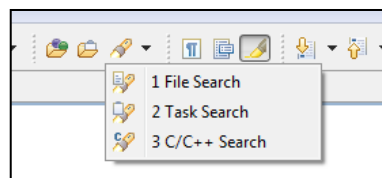


Abbildung 59 (Aufruf der Eclipse Suchfunktion)

4.1.5 Zum Funktionsaufruf gehörende Funktion öffnen

Will man den zu einer im Code benutzten Funktion gehörigen Quellcode betrachten so reicht es aus den Cursor in den Funktionsaufruf zu setzen und die Taste F3 zu drücken. Daraufhin zeigt die Eclipse IDE den Code der jeweiligen Funktion an, sofern dieser nicht in einer präkompilierten Library enthalten ist, und somit nicht mehr als Quellcode zur Verfügung steht.

4.1.6 Sprung zwischen C und H Dateien

Will man schnell zwischen zwei zusammengehörigen C und H Dateien wechseln so reicht es eine der beiden im Editor zu öffnen, um mit der Tastenkombination STRG+TAB die andere Datei zu öffnen.

4.2 Debuggen

4.2.1 Einen Breakpoint hinzufügen bzw. entfernen

Das wird durch einen doppelklick in den grauen Bereich am linken Rand des Code Editors erledigt, Soll der Breakpoint entfernt werden, so reicht ebenfalls ein Doppelklick auf den Breakpoint.

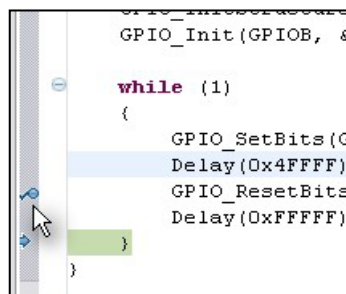


Abbildung 60 (Hinzufügen bzw. entfernen von Breakpoints)

4.2.2 Einen Ausdruck zum Expressions Fenster hinzufügen

Das hat den Vorteil das man ganze Therme auf einmal Überwachen kann. Ein Beispiel:

```
If(((A&&B)<<4)+20) < 0) then do something();
```

Hier könnte man einen Teil der dargestellten IF Bedingung betrachten, z.B. nur $((A \& B) \ll 4)$. Um das zu tun muss man den Ausdruck Markieren, Rechtsklick in die Markierung und „Add Watch Expression...“ auswählen. Daraufhin steht der Ausdruck im Fenster „Expressions“ zur Verfügung. Sollte das nicht angezeigt werden kann man es im Menü unter „Window → Show View“ wieder anzeigen.

4.2.3 Darstellungsart einer Variablen ändern

Das funktioniert leider nur im „Variables“ Fenster indem man einen Rechtsklick auf die betreffende Variable auswählt und dann unter Format das gewünscht Darstellungsformat auswählt. Wer mag kann die Standard Darstellung auch im Menü unter „Window → Preferences“ und dort in der Kategorie „C/C++ → Debug“ einstellen.

Literaturverzeichnis

- [Amo10] Amontec. Amontec Homepage. [Online]. <http://www.amontec.com/index.shtml>,
Zugriff am: 29.01.2010
- [ARM10] ARM. Vorstellung der ARM Cortex-M3 Prozessor Familie. [Online].
http://www.arm.com/products/CPUs/ARM_Cortex-M3.html,
Zugriff am: 24.01.2010
- [Chop10] Freddie Chopin. openOCD 0.3.1 Download Site. [Online].
<http://www.freddiechopin.info/index.php/en/download/category/4-openocd>,
Zugriff am: 29.01.2010
- [CoSo10a] CodeSourcery. Sourcery G++ Lite - GNU Toolchain for ARM Processors.
[Online]. <http://www.codesourcery.com/sgpp/lite/arm>, Zugriff am: 24.01.2010
- [CoSo10b] CodeSourcery. Sourcery G++ Lite Edition User Forum. [Online].
<http://www.codesourcery.com/archives/arm-gnu/maillist.html>,
Zugriff am: 24.01.2010
- [Zyl10] Zylín Consulting. Zylín Embedded CDT Plug-In. [Online].
<http://opensource.zylin.com/embeddedcdt.html>, Zugriff am: 28.01.2010
- [Ecl09] Eclipse Ganymede Download Seite. [Online].
<http://www.eclipse.org/downloads/packages/release/ganymede/sr2>,
Zugriff am: 11.Nov.2009
- [Spa10] SparkFun Electronics. SparkFun Electronic Forum. [Online].
<http://forum.sparkfun.com/index.php>, Zugriff am: 24.01.2010
- [RTOS10] freeRTOS. freeRTOS Homepage. [Online]. <http://www.freertos.org/>, Zugriff
am: 28.01.2010
- [GAEP10] (2010) GNU ARM Eclipse Plug-in Download Seite. [Online].
<http://sourceforge.net/projects/gnuarmeclipse/>, Zugriff am: 11.01.2010
- [Wit10] Wittenstein HighIntegritySystems. freeRTOS Stateviewer Eclipse Plug-In.
[Online]. [http://www.highintegritysystems.com/index.php?option=com_](http://www.highintegritysystems.com/index.php?option=com_chronocontact&Itemid=67)
[chronocontact&Itemid=67](http://www.highintegritysystems.com/index.php?option=com_chronocontact&Itemid=67), Zugriff am: 28.01.2010
- [Jav09] (Nov., 2009) Java Download Seite. [Online].
<http://www.java.com/de/download/manual.jsp>, Zugriff am: 06.Nov.2009
- [Mic10] Micro4you. Micro4you STM Development Board. [Online].
http://www.micro4you.com/store/STM32-Development-Board/prod_57.html ,
Zugriff am: 24.01.2010

- [Mik10] Mikrocontroller.net. Microcontroller.net GCC Forum. [Online].
<http://www.mikrocontroller.net/forum/gcc>, Zugriff am: 24.01.2010
- [STM10b] STMicroelectronics. STM Standard Peripheral Library for STM32F10x.
[Online]. <http://www.st.com/mcu/inchtml.php?fdir=pages&fnam=stm32lib>,
Zugriff am: 24.01.2010
- [STM10a] STMicroelectronics. STM32F103RB Spezifikation. [Online].
<http://www.st.com/stonline/products/literature/ds/13587.pdf>,
Zugriff am: 24.01.2010

Abbildungsverzeichnis

Abbildung 1 (Amontec JTAG-Key Tiny)	4
Abbildung 2 (STM32 Evalboard).....	4
Abbildung 3 (Codesourcery G++ Lite EABI Setup, Schritt 4)	4
Abbildung 4 (Codesourcery G++ Lite EABI Setup, Schritt 6)	5
Abbildung 5 (Verifikation des CodeSourcery Compiler Setups).....	5
Abbildung 6 (openOCD 0.3.1 Setup Optionen)	6
Abbildung 7 (Test der openOCD 0.3.1 Installation)	6
Abbildung 8 (Extrahieren der Treiberdateien)	7
Abbildung 9 (libUSB ftdi Treiber Dateien).....	7
Abbildung 10 (Amontec JTAGkey Tiny Treiber Setup).....	8
Abbildung 11 (Treiberssetup Amontec JTAGkey Channel A).....	8
Abbildung 12 (Gerätemanager nach dem Treiberssetup)	8
Abbildung 13 (Verifikation der Java Installation).....	9
Abbildung 14 (Eclipse Programmordner)	9
Abbildung 15 (Auswahl des Eclipse Workspace)	10
Abbildung 16 (Eclipse Willkommen Seite).....	10
Abbildung 17 (Eclipse Standard C/C++ Ansicht)	11
Abbildung 18 (Starten des Eclipse Software Update Managers)	12
Abbildung 19 (Eclipse Software Update und Add-on Dialog).....	12
Abbildung 20 (Hinzufügen einer Eclipse Update Site)	12
Abbildung 21 (Auswahl der hinzugefügten Eclipse Update Sites)	13
Abbildung 22 (Verifikation der GnuARM Eclipse Plug-in Installation)	14
Abbildung 23 (Verifikation des Zylind Embedded CDT Plug-Ins)	14
Abbildung 24 (Öffnen der Debug Perspective).....	15
Abbildung 25 (öffnen des Show View Dialogs)	16
Abbildung 26 (Auswahl der neuen Ansichten)	16
Abbildung 27 (FreeRTOS Stateviewer Ansichten in der Debug Perspective).....	16
Abbildung 28 (Öffnen des External Tools Configurations Dialogs).....	17
Abbildung 29 (Anlegen einer neuen Konfiguration).....	17
Abbildung 30 (Konfiguration des Extenen Tools (1 von 2))	18
Abbildung 31 (Konfiguration des Extenen Tools (2 von 2))	19
Abbildung 32 (Kopierte Konfigurationen)	19
Abbildung 33 (Erstelle Tool Konfigurationen)	20
Abbildung 34 (External Tools in der Symbolleiste).....	20
Abbildung 35 (Anlegen eines Neuen ARM Projekts)	21
Abbildung 36 (blinky Projektdateien)	22
Abbildung 37 (Projekteinstellungen).....	23
Abbildung 38 (Angabe der benötigten Include Pfade)	24
Abbildung 39 (C Compiler Einstellungen).....	24
Abbildung 40 (Angabe des Linker Skripts).....	25
Abbildung 41 (Angabe der Linker Librarys)	25
Abbildung 42 (Kompilieren des Projekts).....	26
Abbildung 43 (Build Protokoll nach einem erfolgreichen Buildprozess)	26
Abbildung 44 (Download der Applikation)	27
Abbildung 45 (openOCD Output nach dem Flashen)	27
Abbildung 46 (Anlegen einer neuen Debug Konfiguration).....	28

Abbildung 47 (Einstellungen der Debug Konfiguration (1 von 4))	28
Abbildung 48 (Einstellungen der Debug Konfiguration (2 von 4))	29
Abbildung 49 (Einstellungen der Debug Konfiguration (3 von 4))	29
Abbildung 50 (Einstellungen der Debug Konfiguration (4 von 4))	30
Abbildung 51 (Starten des openOCD Servers)	30
Abbildung 52 (openOCD Server Output).....	31
Abbildung 53 (Starten des Debuggers)	31
Abbildung 54 (Symbolleiste zur Kontrolle des Debuggers)	31
Abbildung 55 (Eclipse Debug Perspective)	32
Abbildung 56 (Beenden einer Debug Session).....	32
Abbildung 57 (Terminierte Debug Prozesse).....	33
Abbildung 58 (Umschalten auf die C/C++ Perspective).....	33
Abbildung 59 (Aufruf der Eclipse Suchfunktion)	34
Abbildung 60 (Hinzufügen bzw. entfernen von Breakpoints)	35

Buildprotokoll des Projekt Blinky

**** Build of configuration Debug for project blinky ****

```
cs-make all
'Building file: ../Blinky.c'
'Invoking: ARM Sourcery Windows GCC C Compiler'
arm-none-eabi-gcc -I"C:\eclipse_workspace\blink\Libraries" -I"C:\eclipse_workspace\blink\Libraries\CMSIS"
-I"C:\eclipse_workspace\blink\Libraries\STM32F10x_StdPeriph_Driver\inc" -O0 -pedantic -Wall -fsigned-char -c
-fmessage-length=0 -MMD -MP -MF"Blinky.d" -MT"Blinky.d" -mcpu=cortex-m3 -mthumb -g3 -gdwarf-2 -o"Blinky.o"
"../Blinky.c"
'Finished building: ../Blinky.c'
'
'
'Building file: ../Libraries/startup_stm32f10x_md.s'
'Invoking: ARM Sourcery Windows GCC Assembler'
arm-none-eabi-gcc -x assembler-with-cpp -Wall -c -fmessage-length=0 -MMD -MP
-MF"Libraries/startup_stm32f10x_md.d" -MT"Libraries/startup_stm32f10x_md.d" -mcpu=cortex-m3 -mthumb -g3
-gdwarf-2 -o"Libraries/startup_stm32f10x_md.o" " ../Libraries/startup_stm32f10x_md.s"
'Finished building: ../Libraries/startup_stm32f10x_md.s'
'
'
'Building file: ../Libraries/stm32f10x_it.c'
'Invoking: ARM Sourcery Windows GCC C Compiler'
arm-none-eabi-gcc -I"C:\eclipse_workspace\blink\Libraries" -I"C:\eclipse_workspace\blink\Libraries\CMSIS"
-I"C:\eclipse_workspace\blink\Libraries\STM32F10x_StdPeriph_Driver\inc" -O0 -pedantic -Wall -fsigned-char -c
-fmessage-length=0 -MMD -MP -MF"Libraries/stm32f10x_it.d" -MT"Libraries/stm32f10x_it.d" -mcpu=cortex-m3 -mthumb
-g3 -gdwarf-2 -o"Libraries/stm32f10x_it.o"
" ../Libraries/stm32f10x_it.c"
'Finished building: ../Libraries/stm32f10x_it.c'
'
'
'Building file: ../Libraries/STM32F10x_StdPeriph_Driver/src/misc.c'
'Invoking: ARM Sourcery Windows GCC C Compiler'
arm-none-eabi-gcc -I"C:\eclipse_workspace\blink\Libraries" -I"C:\eclipse_workspace\blink\Libraries\CMSIS"
-I"C:\eclipse_workspace\blink\Libraries\STM32F10x_StdPeriph_Driver\inc" -O0 -pedantic -Wall -fsigned-char -c
-fmessage-length=0 -MMD -MP -MF"Libraries/STM32F10x_StdPeriph_Driver/src/misc.d"
-MT"Libraries/STM32F10x_StdPeriph_Driver/src/misc.d" -mcpu=cortex-m3
-mthumb -g3 -gdwarf-2 -o"Libraries/STM32F10x_StdPeriph_Driver/src/misc.o"
```

```

"../Libraries/STM32F10x_StdPeriph_Driver/src/misc.c"
'Finished building: ../Libraries/STM32F10x_StdPeriph_Driver/src/misc.c'
'
'Building file: ../Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_exti.c'
'Invoking: ARM Sourcery Windows GCC C Compiler'
arm-none-eabi-gcc -I"C:\eclipse_workspace\blink\Libraries" -I"C:\eclipse_workspace\blink\Libraries\CMSIS"
-I"C:\eclipse_workspace\blink\Libraries\STM32F10x_StdPeriph_Driver\inc" -O0 -pedantic -Wall -fsigned-char -c
-fmessage-length=0 -MMD -MP -MF"Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_exti.d"
-MT"Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_exti.d" -mcpu=cortex-m3 -mthumb -g3 -gdwarf-2
-o"Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_exti.o"
"../Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_exti.c"
'Finished building: ../Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_exti.c'
'
'Building file: ../Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_gpio.c'
'Invoking: ARM Sourcery Windows GCC C Compiler'
arm-none-eabi-gcc -I"C:\eclipse_workspace\blink\Libraries" -I"C:\eclipse_workspace\blink\Libraries\CMSIS"
-I"C:\eclipse_workspace\blink\Libraries\STM32F10x_StdPeriph_Driver\inc" -O0 -pedantic -Wall -fsigned-char -c
-fmessage-length=0 -MMD -MP -MF"Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_gpio.d"
-MT"Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_gpio.d" -mcpu=cortex-m3 -mthumb -g3 -gdwarf-2
-o"Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_gpio.o"
"../Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_gpio.c"
'Finished building: ../Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_gpio.c'
'
'Building file: ../Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_rcc.c'
'Invoking: ARM Sourcery Windows GCC C Compiler'
arm-none-eabi-gcc -I"C:\eclipse_workspace\blink\Libraries" -I"C:\eclipse_workspace\blink\Libraries\CMSIS"
-I"C:\eclipse_workspace\blink\Libraries\STM32F10x_StdPeriph_Driver\inc" -O0 -pedantic -Wall -fsigned-char -c
-fmessage-length=0 -MMD -MP -MF"Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_rcc.d"
-MT"Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_rcc.d" -mcpu=cortex-m3 -mthumb -g3 -gdwarf-2
-o"Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_rcc.o"
"../Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_rcc.c"
'Finished building: ../Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_rcc.c'
'
'Building file: ../Libraries\CMSIS/core_cm3.c'
'Invoking: ARM Sourcery Windows GCC C Compiler'
arm-none-eabi-gcc -I"C:\eclipse_workspace\blink\Libraries" -I"C:\eclipse_workspace\blink\Libraries\CMSIS"
-I"C:\eclipse_workspace\blink\Libraries\STM32F10x_StdPeriph_Driver\inc" -O0 -pedantic -Wall -fsigned-char -c
-fmessage-length=0 -MMD -MP -MF"Libraries\CMSIS/core_cm3.d" -MT"Libraries\CMSIS/core_cm3.d" -mcpu=cortex-m3
-mthumb -g3 -gdwarf-2 -o"Libraries\CMSIS/core_cm3.o" " ../Libraries\CMSIS/core_cm3.c"
'Finished building: ../Libraries\CMSIS/core_cm3.c'

```

```

' ,
'Building file: ../Libraries/CMSIS/system_stm32f10x.c'
'Invoking: ARM Sourcery Windows GCC C Compiler'
arm-none-eabi-gcc -I"C:\eclipse_workspace\blinkys\Libraries" -I"C:\eclipse_workspace\blinkys\Libraries\CMSIS"
-I"C:\eclipse_workspace\blinkys\Libraries\STM32F10x_StdPeriph_Driver\inc" -O0 -pedantic -Wall -fsigned-char -c
-fmessage-length=0 -MMD -MP -MF"Libraries\CMSIS\system_stm32f10x.d" -MT"Libraries\CMSIS\system_stm32f10x.d"
-mcpu=cortex-m3 -mthumb -g3 -gdwarf-2 -o"Libraries\CMSIS\system_stm32f10x.o"
"../Libraries\CMSIS\system_stm32f10x.c"
'Finished building: ../Libraries\CMSIS\system_stm32f10x.c'
' ,
'Building target: blinky.elf'
'Invoking: ARM Sourcery Windows GCC C Linker'
arm-none-eabi-gcc -T"C:\eclipse_workspace\blinkys\Linker\stm32.ld" -nostartfiles -nodefaultlibs -
L"C:\eclipse_workspace\blinkys\Linker" -Wl,-Map,blinkys.map -mcpu=cortex-m3 -mthumb -g3 -gdwarf-2 -o"blinkys.elf"
./Blinky.o ./Libraries/startup_stm32f10x_md.o ./Libraries/stm32f10x_it.o
./Libraries/STM32F10x_StdPeriph_Driver/src/misc.o ./Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_exti.o
./Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_gpio.o
./Libraries/STM32F10x_StdPeriph_Driver/src/stm32f10x_rcc.o ./Libraries\CMSIS/core_cm3.o
./Libraries\CMSIS/system_stm32f10x.o
'Finished building target: blinkys.elf'
' ,
'Invoking: ARM Sourcery Windows GNU Create Flash Image'
arm-none-eabi-objcopy -O ihex blinkys.elf "blinkys.hex"
'Finished building: blinkys.hex'
' ,
'Invoking: ARM Sourcery Windows GNU Create Listing'
arm-none-eabi-objdump -h -S blinkys.elf >"blinkys.lst"
'Finished building: blinkys.lst'
' ,
'Invoking: ARM Sourcery Windows GNU Print Size'
arm-none-eabi-size --format=berkeley blinkys.elf
  text      data      bss      dec      hex      filename
  7648       20       256     7924     1ef4     blinkys.elf
'Finished building: blinkys.siz'
' ,

```

openOCD Output während des Flashen

```
Open On-Chip Debugger 0.3.1 (2009-11-13-16:13)
$URL$
For bug reports, read
    http://openocd.berlios.de/doc/doxygen/bugs.html
1000 kHz
jtag_nsrst_delay: 100
jtag_ntrst_delay: 100
trst_and_srst separate srst_gates_jtag trst_push_pull srst_open_drain
Info : clock speed 1000 kHz
Info : JTAG tap: stm32.cpu tap/device found: 0x3ba00477 (mfg: 0x23b, part: 0xba00, ver: 0x3)
Info : JTAG tap: stm32.bs tap/device found: 0x16410041 (mfg: 0x020, part: 0x6410, ver: 0x1)
Info : JTAG tap: stm32.cpu tap/device found: 0x3ba00477 (mfg: 0x23b, part: 0xba00, ver: 0x3)
Info : JTAG tap: stm32.bs tap/device found: 0x16410041 (mfg: 0x020, part: 0x6410, ver: 0x1)
auto erase enabled
Info : device id = 0x20036410
Info : flash size = 128kbytes
Warn : not enough working area available(requested 16384, free 16336)
Info : Halt timed out, wake up GDB.
wrote 7668 byte from file C:/eclipse_workspace/blinkys/debug/blinkys.hex in 1.375000s (5.446023 kb/s)
target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x01000000 pc: 0x0800bebc msp: 0x20005000
Info : JTAG tap: stm32.cpu tap/device found: 0x3ba00477 (mfg: 0x23b, part: 0xba00, ver: 0x3)
Info : JTAG tap: stm32.bs tap/device found: 0x16410041 (mfg: 0x020, part: 0x6410, ver: 0x1)
```