

Debugging with Eclipse

INDINF10



27. Februar 2015

Stefan Erceg

4AHITT

Inhalt

[1. Aufgabenstellung 1](#_Toc412892602)

[2. Arbeitsdurchführung 2](#_Toc412892603)

[2.1 Allgemeines 2](#_Toc412892604)

[2.2 gcc-arm-none-eabi updaten 2](#_Toc412892605)

[2.3 Eclipse Software updaten 2](#_Toc412892606)

[2.4 GNU ARM Eclipse Plugin herunterladen 2](#_Toc412892607)

[2.5 GDB Hardware Debugging Plugin herunterladen 3](#_Toc412892608)

[2.6 Projekt anlegen und dessen Einstellungen bearbeiten 3](#_Toc412892609)

[2.7 Flash-Tool konfigurieren und Mikrocontroller flashen 4](#_Toc412892610)

[2.8 Debugging konfigurieren und ausführen 5](#_Toc412892611)

[3. Literaturverzeichnis 6](#_Toc412892612)

*Github-Link:* [*https://github.com/serceg-tgm/4AHITT-SYT/tree/indinf/IndInf10*](https://github.com/serceg-tgm/4AHITT-SYT/tree/indinf/IndInf10)

*Github-Tag: erceg\_indinf10*

# 1. Aufgabenstellung

Dokumentieren Sie alle notwendigen Schritte um in eclipse das Blink Beispiel zu debuggen.

Zeigen Sie auch alle notwendigen Schritte für den Debugging Prozess.

Geben Sie das Protokoll als PDF ab (Metaregeln bezüglich Design/Stunden/Testing entfallen).

# 2. Arbeitsdurchführung

## 2.1 Allgemeines

Die Übung wurde auf der vom Herr Professor Borko zur Verfügung gestellten virtuellen Maschine, welche das Betriebssystem Debian Wheezy besitzt, durchgeführt, da auf der Maschine alle notwendigen Installationen und Konfigurationen für den Mikrocontroller sowie Eclipse Kepler CDT enthalten sind.

## 2.2 gcc-arm-none-eabi updaten

Zu Beginn muss gcc-arm-none-eabi auf die neueste Version 4.9 aktualisiert werden. Dazu habe ich die tar.bz2-Datei für Linux auf folgender Seite heruntergeladen: [1].

Diese Datei wurde im Verzeichnis /home/serceg/opt mittels folgendem Befehl entpackt: tar xfvj archiv.tar.bz2 und somit die alte Version durch die neue ersetzt.

## 2.3 Eclipse Software updaten

Bevor die notwendigen Tools in Eclipse heruntergeladen werden, werden alle verfügbaren Pakete aktualisiert. Dazu geht man in Eclipse auf die Menüleiste Help und klickt auf Check for Updates.

## 2.4 GNU ARM Eclipse Plugin herunterladen

Zuerst habe ich im Internet nach dem GNU ARM Eclipse Plugin – Link [2] gesucht. Welchen Link man hinzufügen muss und wie man das Plugin herunterlädt, habe ich auf der gnuarmeclipse-Seite [3] herausgefunden. Nachdem ich den Link gefunden habe, habe ich folgende Schritte in Eclipse durchgeführt:

auf die Menüleiste Help gegangen 🡪 Install New Software… geklickt 🡪 bei Work with den Plugin-Link [2] hinzugefügt 🡪 Button Add angeklickt und als Namen GNU ARM Plugin eingegeben 🡪 Gesamten Punkt GNU ARM C/C++ Cross Development Tools ausgewählt 🡪 next geklickt 🡪 AGBs akzeptiert 🡪 finish geklickt

Als die Warnung „You are installing software that contains unsigned content…“ auftauchte, habe ich auf Ok geklickt und die Installation wurde weiter ausgeführt.

Nachdem das Plugin installiert wurde, tauchte die Meldung auf, dass Eclipse neugestartet werden soll, damit die Änderungen übernommen werden.

## 2.5 GDB Hardware Debugging Plugin herunterladen

Als nächsten Schritt holte ich mir den Link [4] für das GDB Hardware Debugging Plugin. Welchen Link man hinzufügen muss und wie man das Plugin herunterlädt, habe ich auf einem Blog [5] herausgefunden. Folgende Schritte wurden in Eclipse durchgeführt:

auf die Menüleiste Help gegangen 🡪 Install New Software… geklickt 🡪 bei Work with den Plugin-Link [4] hinzugefügt 🡪 Button Add angeklickt und als Namen GDB Debugger eingegeben 🡪 beim Unterpunkt CDT Optional Features das Feature GDB Hardware Debugging ausgewählt 🡪 2 x next geklickt 🡪 AGBs akzeptiert 🡪 finish geklickt

Als die Warnung „You are installing software that contains unsigned content…“ auftauchte, habe ich auf Ok geklickt und die Installation wurde weiter ausgeführt.

Auch nach dieser Installation musste Eclipse neugestartet werden, damit die Änderungen übernommen werden.

## 2.6 Projekt anlegen und dessen Einstellungen bearbeiten

Nun konnte ein neues STM32F3xx C/C++ Projekt angelegt werden.

Dazu wurden in Eclipse folgende Schritte durchgeführt:

auf die Menüleiste File gegangen 🡪 New -> C Project geklickt 🡪 unter Project name „IndInf10“ eingegeben 🡪 beim Project type „STM32F3xx C/C++ Project“ ausgewählt 🡪 next geklickt 🡪 unter Content „Blinky (blink a led)“, welches das Projekt mit einem Beispiel füllt, bei dem eine LED auf dem Mikrocontrollerboard permanent blinkt, ausgewählt 🡪 3 x next geklickt 🡪 finish geklickt

Einige Einstellungen mussten nun noch bearbeitet werden, z.B. der Standard auf ISO C99 gesetzt werden. Dazu geht man in Eclipse wie folgt vor:

auf die Menüleiste Project gegangen 🡪 Properties geklickt 🡪 bei C/C++ Build den Unterpunkt Settings ausgewählt 🡪 bei der Drop-Down-Liste der Configuration

„[ All Configurations ]“ ausgewählt

🡪 bei der Tool Settings – Registerkarte auf den Unterpunkt von Cross ARM C Compiler „Optimization“ geklickt 🡪 als Language standard „ISO C99 (-std=c99)“ ausgewählt

🡪 bei der Build Steps – Registerkarte bei Post-build steps als Command

„arm-none-eabi-objcopy -O binary ${ProjName}.elf ${ProjName}.bin“ eingegeben

## 2.7 Flash-Tool konfigurieren und Mikrocontroller flashen

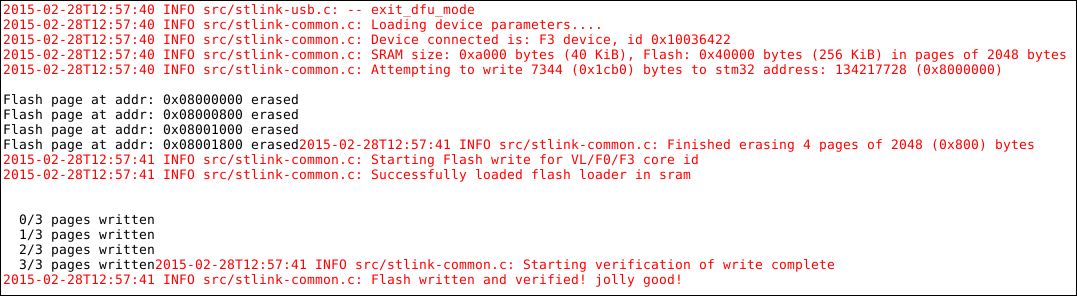
Damit der Mikrocontroller unter Eclipse geflasht werden kann, wurde folgendermaßen vorgegangen:

auf die Menüleiste Run gegangen 🡪 External Tools -> External Tools Configurations… geklickt 🡪 Rechtsklick auf Program und New ausgewählt

🡪 bei der Main – Registerkarte folgendes eingegeben:

* Location: „/home/serceg/repositories/stlink/st-flash”
* Working Directory: „${workspace\_loc:/IndInf10/Debug}”
* Arguments: „--reset write ${project\_name}.bin 0x08000000”

🡪 Apply und Run geklickt 🡪 Mikrocontroller wurde geflasht und die Ausgabe in der Konsole sah folgendermaßen aus:



*Abbildung 1: Mikrocontroller flashen [Stefan Erceg, gemacht am 27.02.2015]*

Nun wurde das Programm auf den Mikrocontroller übertragen und eine LED fing permanent an zu blinken.

## 2.8 Debugging konfigurieren und ausführen

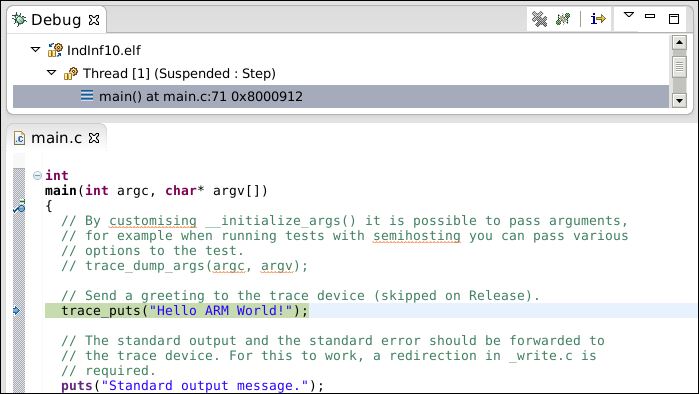
Da wir das Programm noch debuggen möchten, mussten hier ebenfalls bestimmte Einstellungen konfiguriert werden. Welche Schritte hier ausgeführt werden mussten, habe ich auf der gnuarmeclipse-Seite [6] nachgelesen. Folgende Schritte wurden in Eclipse durchgeführt:

auf die Menüleiste Run gegangen 🡪 Debug Configurations… geklickt 🡪 Rechtsklick auf GDB OpenOCD Debugging und New ausgewählt

🡪 bei der Debugger – Registerkarte bei den Config options

„-f /usr/share/openocd/scripts/board/stm32f3discovery.cfg“ eingegeben

🡪 Apply und Debug geklickt:



*Abbildung 2: Debugging ausführen [Stefan Erceg, gemacht am 27.02.2015]*

Das Debugging für das jeweilige Programm wurde nun gestartet. Breakpoints können links neben den Codezeilen festgelegt werden. Mittels Step Over (F6) kann jede Codezeile Schritt für Schritt durchgegangen und das Verhalten des Programms überprüft werden:

# 3. Literaturverzeichnis

[1] Canoncial Ltd. (2004, 2015). GNU Tools for ARM Embedded Processors [Online].

Available at: https://launchpad.net/gcc-arm-embedded/+download

[zuletzt abgerufen am 27.02.2015]

[2] Dice Holdings, Inc. (2013, 2015). GNU ARM Eclipse Plug-ins [Online]. Available at: <http://sourceforge.net/projects/gnuarmeclipse/files/Eclipse/updates/>

[zuletzt abgerufen am 27.02.2015]

[3] GNU ARM Eclipse (2015). GNU ARM Eclipse Plug-ins install [Online]. Available at: <http://gnuarmeclipse.livius.net/blog/plugins-install/>

[zuletzt abgerufen am 27.02.2015]

[4] The Eclipse Foundation (2014). Eclipse CDT Plugins [Online]. Available at: <http://download.eclipse.org/tools/cdt/releases/8.5/>

[zuletzt abgerufen am 27.02.2015]

[5] Andrei Istodiresco (November 2012). STM32F3 Discovery + Eclipse + OpenOCD [Online]. Available at: <http://www.engineering-diy.blogspot.ro/2012/11/stm32f3-discovery-eclipse-openocd.html> [zuletzt abgerufen am 27.02.2015]

[6] GNU ARM Eclipse (2015). The OpenOCD debugging Eclipse plug-in [Online].

Available at: <http://gnuarmeclipse.livius.net/blog/openocd-debugging/>

[zuletzt abgerufen am 27.02.2015]