



Winterthur, 30. August 2011

swissT.meeting

Embedded Computing Conference

Open Source Tools für ARM Cortex-M3 basierte Mikrocontroller

Referent: Matthias Meier
FHNW, Institut für Automation

Open Source Tools für Cortex-M3 Mikrocontroller

Inhalt:

- Was ist ein Cortex-M3 basierter Mikrocontroller
- Vor-/Nachteile von Open Source Tools
- Für wen ist eine Open Source Lösung geeignet
- Überblick über empfohlene SW-Komponenten
- Zusammenhänge und Tipps
- ev. Kurzdemo
- Fragen

ARM Cortex-M3 basierte MCUs

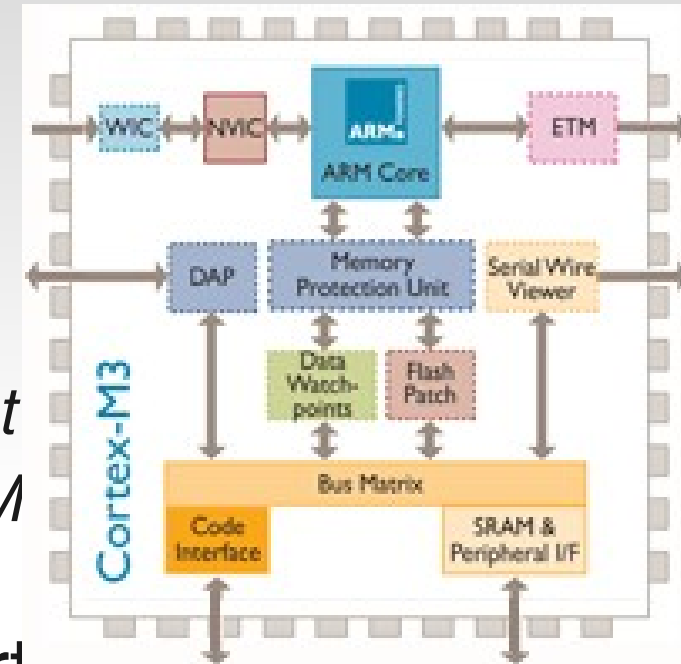
<http://www.arm.com/products/processors/cortex-m/cortex-m3.php>

■ Allen gemeinsam: ARM Cortex-M3 Kern

- 32-Bit CPU (ARMv7-M)
- SysTimer, Interrupt Controller, Pwr Mgmt
- Opt. Debug Interface: JTAG, SWD, ETM

■ Herstellerabhängig implementiert

- Speicher (RAM, Flash) u. Flash-Programming
- Peripherieblöcke (zwischen Herstellern inkompatibel)
 - Hersteller liefern zweck Programmierung eigene inkompatible C Libraries
 - Peripherie unterschiedlich betreffend Leistung und Umfang
 - z.B. unterstützen nur wenige Cortex-M3-Hersteller USB2 - High Speed



Vor- und Nachteile von Open Source C Entwicklungstools

■ Vorteile:

- *Keine Lizenzkosten, keinen Linzenzadministrationsaufwand*
- *Keine Grauzone, keine rechtlichen Probleme*
- *Keine künstlichen Leistungseinschränkungen*
- *Gewisser kostenloser Support (Community-Foren)*
- *Qualitativ hochstehenden C/C++ Compiler (GNU GCC)*
- *Hohe Flexibilität, Cross-Platform (Host und Target)*
- *Langfristig nutzbares Wissen*

■ Nachteile:

- *Vielen separat zu installierenden Komponenten*
- *Installation (ev.) anspruchsvoll und verwirrend*
- *Keine definierte Hotline, kein kommerzieller Support*
- *Oft auch wenig Unterstützung durch Controller-Herstellern*

Für wen ist eine Open Source C Entwicklungsumgebung geeignet?

■ Eignung:

- *Schulen, heterogenes Umfeld*
- *Low Budget Projekte (Private, KMUs)*
- *Weiter-/Entwicklung von Open-Source Software*
- *Langfristige Verwendung/Recycling des Sourcecodes*

■ Notwendige Voraussetzungen:

- *Erst-Installationsphase nicht unter Zeitdruck*
- *Übung in Problemsuche (Google-/Forensuche)*
- *Etwas PC-Kenntnisse (Treiber- und PATH-Probleme lösen)*

Empfohlene Komponenten für Embedded C/C++ Entwicklungen

- GUI: **Eclipse** + **CDT-Plugin** (C/C++ Development Tools)
 - *Komfortabler Program-Editor sowie Debugging-Frontend*
 - GNU Cross-Compiler Toolchain:
 - **arm-none-eabi-gcc** (Host: x86 Win/Linux, Target: ARM)
 - GNU Utility: **make** (sowie ev. weitere GNU-Tools)
 - *Steuerung des Build-Vorganges (compile & link)*
 - Flash-/Debugger-Backend Software: **OpenOCD**
 - *Ermöglicht Flashen diverser Controller via JTAG*
 - *Emuliert „gdbserver“, also das Backend zum GDB Debugger*
 - *benötigt hostseitig **libusb**-Treiber passend zu...*
 - Debug-Hardware: **USB-JTAG Interface**
 - *verbindet PC/USB mit JTAG-Interface des Target-Controllers*
- Für Linux-Host z.B. **CodeSourcery G++ Lite ARM EABI**
- Für Windows-Host z.B. **yagarto GNU-ARM Toolchain + yagarto Tools**

Embedded Entwicklungen mit/ohne Source-Code-Debugging

- **Problemlos: ohne Sourcecode-Debugging:**
 - *Programm-Editor (ASCII-Editor, ..., Eclipse/CDT)*
 - *Cross-Compiler Toolchain für ARM + Build-Utility (make)*
 - *Flash-Programming-Software des Controller-Herstellers*
 - *Download via Serielle Schnittstelle - meist nur Windows Platform unterstützt*
- **Knacknuss: Sourcecode-Debugging via JTAG:**
 - *Open Source Lösung: OpenOCD (Open On-Chip Debugger)*
 - *Unterstützt nebst Debuggen auch Flashen via JTAG*
 - *Wird Controller von OpenOCD unterstützt? (Clocking und Flash-Routinen)*
 - *Weitere Stolperfallen*
 - *Wurde OpenOCD kompiliert für Ihr USB-JTAG-Interface/Treiber (FT2232, libusb)*
 - *Windows: manuelle USB-Treiber-Install. / falsch installierten Treiber ersetzen*
 - *Host-Platform: Linux 32/64, Windows 32/64Bit (letzteres Treiber-Zert. abschalten)*
 - *OpenOCD-Command-Syntax abhängig vom OpenOCD-Release*
 - *OpenOCD unterstützt nur JTAG, hingegen SWD und ETM derzeit noch nicht*

low-cost USB-JTAG Interfaces

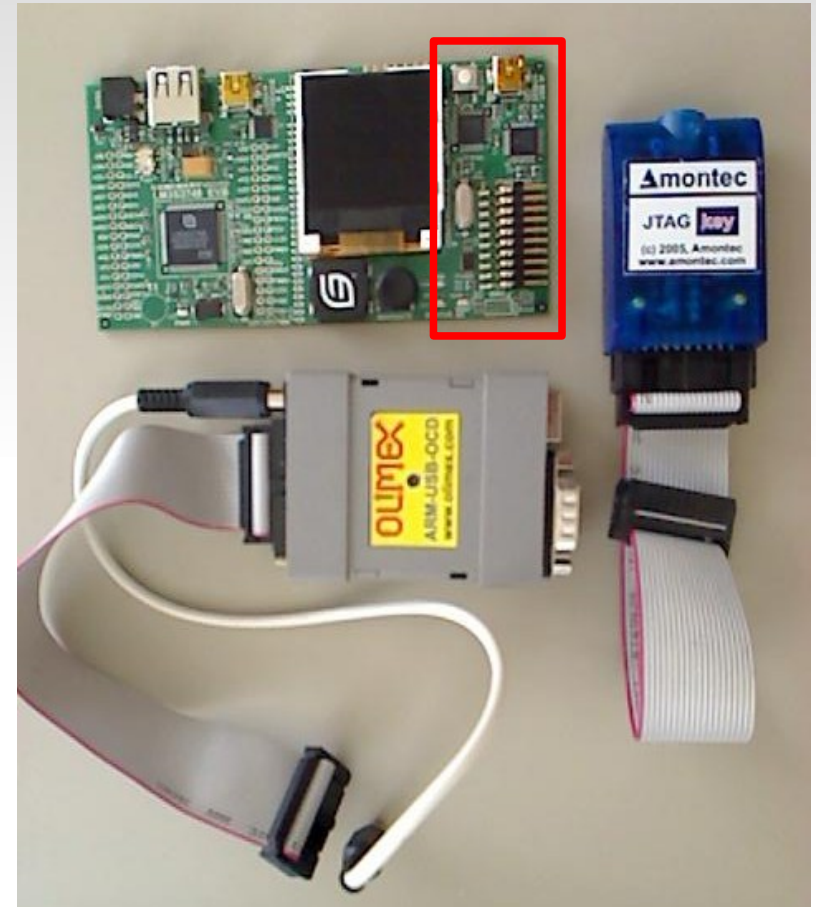
auf Basis FT2232, erfolgreich getestet mit OpenOCD

- Amontec JTAGkey

- *Bezugsquelle:*
<http://www.amontec.com/>

- Olimex ARM-USB-OCD

- *Bezugsquellen:*
www.olimex.com
www.redacom.ch



- TI – Stellaris Evaluation Kits

- *Target Boards inkl. USB-JTAG-Interface, z.B. EKC-LM3S3748*
- *Bezugsquelle:* www.mouser.com, www.digikey.com

einige Cortex-M3 Eval. Boards

erfolgreich getestetete mit OpenOCD

■ Atmel

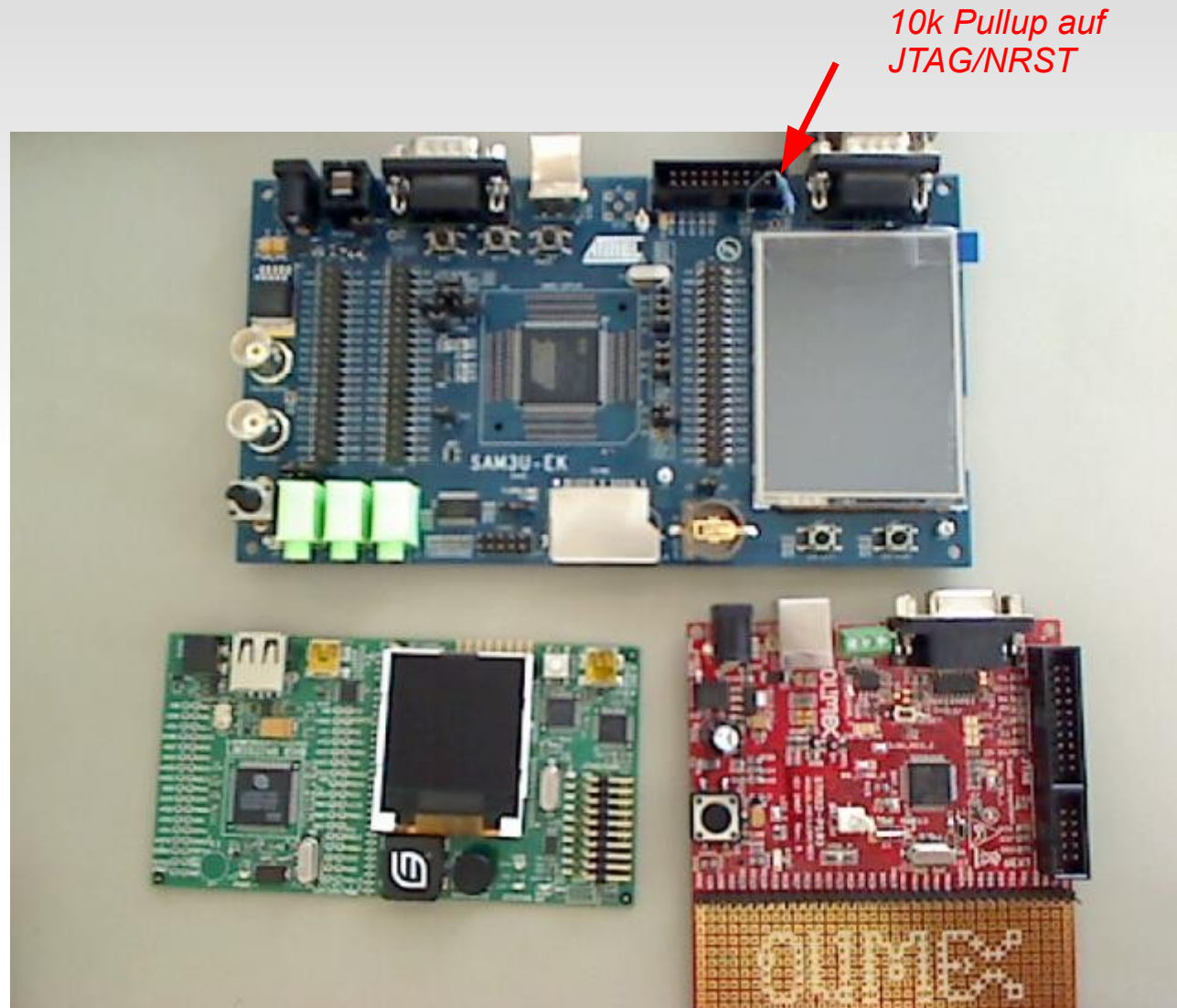
- *SAM3U-EK*
- *rel. umfangreich*
- *rel. teuer*

■ Olimex

- *STM32-P103*
- *sehr spartanisch*

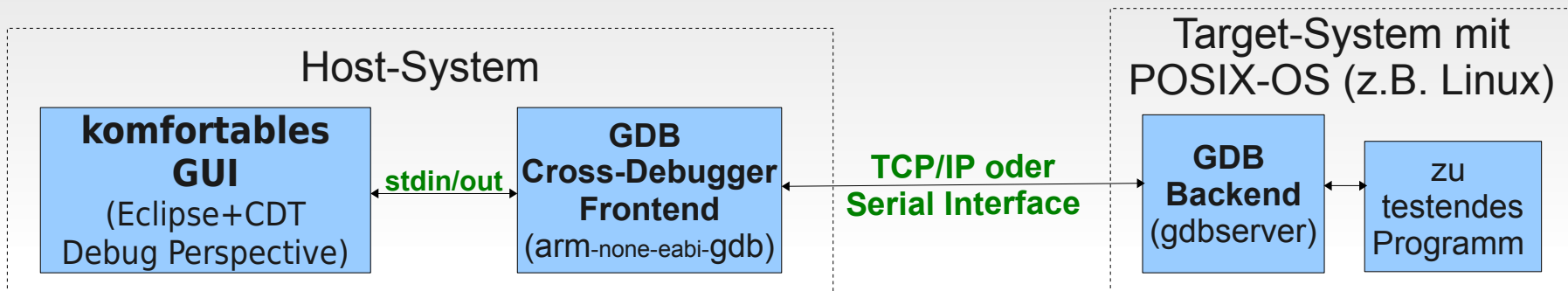
■ TI Stellaris

- *div. Boards*
 - z.B. *EKC-LM3S3748*
- *sehr günstig*
- *inkl. USB-JTAG Interface (auch für externe Controller verwendbar)*

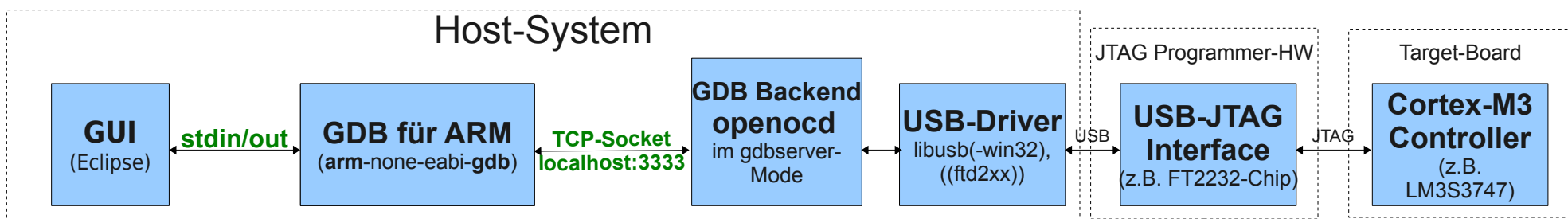


GDB Remotedebugging prinzipiell (GDB = GNU Debugger)

- Konventionelles Remote-Debugging mit GDB:



- On-Chip Debugging mit Open OCD:



Steuerung des Build-Vorgangs bei Eclipse „C/C++ Makefile Projects“

- Programm *make(.exe)* (eine GNU Utility)
 - *Steuert den Buildvorgang (also nicht Eclipse selbst)*
- interpretiert projektspezifische Steuerdatei *Makefile*
 - *Manuell erstellen/anpassen auf Basis von Demo-Example!*
 - *In Makefile wird in „Rules“ (etwas kryptisch) definiert...*
 - *Aus welchen Sourcefiles das C/C++ Projekt besteht*
 - *Wie/welche Cross-Compiler & -Linker aufgerufen werden (inkl. Compiler-Flags)*
 - *In welchen Verzeichnissen sich die Libraries und Includefiles befinden*
 - *Eclipse/CDT erwartet zwecks (Re-)Build minimal die Rules...*
 - *„all“ beschreibt, wie das Projekt (nach Änderung der Source) erstellt wird.*
 - *„clean“ um alle erstellten Objektfiles zu löschen zwecks Archivierung/Neugen.*
 - *Eigens zugefügte Rules... (Unter Eclipse ausgeführt als „External Commands“)*
 - *„flash“ zwecks Flashen des Controllers via OpenOCD & USB-JTAG Interface*
 - *„gdbserver“ zwecks starten von OpenOCD im gdbserver-Modus (für Debugging)*

Tipp: Projekt-Konfiguration

■ Projekt-Konfiguration soweit wie möglich im **Makefile** definieren

- *Nebst Build-Vorgang auch Flashen und Starten des gdbservers*
- *vermeidet weitere projektspezifische Shellscripts, Batchfiles, Konfigfiles*
- *Vereinfacht Dokumentation, Archivierung, Portierung*

■ Auf Eclipse-Ebene nur noch

- *Definition des GDB Cross-Debugger-Frontends (leider projektspez.)*
 - *arm-none-eabi-gdb* sowie Name des ausführbaren Programms (z.B. main.elf)
- *External Commands*
 - *zwecks Flashen per „make flash“ und Start des gdbservers per „make gdbserver“*
- *optional Indexer für Include-Files für Programm-.Editor*

■ Auf Ebene Host-OS (Windows/Linux)

- *PATH zu GNU Toolchain, Make-Utility, OpenOCD*

Tipp: Einbinden von Libraries in eigene Projekte

■ Libraries wie...

➤ Hersteller-abhängigen Peripherie-Libraries

- TI Stellaris „StellarisWare“, Atmel „at91lib“, ST Microelectronics „STM32 Standard Periphery Library“

➤ Funktionale Libraries wie

- FreeRTOS, FsFAT, ...

■ Nicht direkt in Projektverzeichnisse kopieren!

➤ Sondern jeweils als separates Eclipse „General Project“ unter dem Workspace importieren (nicht als Makefile-Project!)

➤ Diese Projekt(-Verzeichnis) in den Makefile-Projekten bei Bedarf referenzieren

➤ Dadurch mehr Ordnung und vereinfachter Library-Update

- Unter Linux ev. zudem Symlinks verwenden

➤ Bedingt meist Anpassung des Makefiles (Include/Source-Pfade)

➤ Bedingt bei Wechsel zwischen Projekten: **Project > Clean**

Tipp: Projekt-Versionsverwaltung (Revision Control System)

- (CVS)
 - *sehr verbreitet, technisch aber veraltet (nicht für Neuprojekte verwenden!)*
 - *Version History auf CVS-Server*
- (SVN – Subversion)
 - *neuer, Version History auf SVN- oder WebDAV-Server*
- Git
 - *Lokale Kopie der gesamten Project Version History (!)*
 - *im Projekt-Unterverzeichnis '.git/'*
 - *Auch rein lokale Projektverwaltung möglich – ohne Git-Server*
 - *Benutzung*
 - *entweder per Git Commandline Tools (div. Tutorials auf Internet)*
 - *oder mittels Eclipse Git-Plugin „EGit“*

Ev. Kurzdemo Eclipse

- Eclipse Workspace
- Eclipse „C/C++ Makefile-Projekt“ erstellen
 - *ursprünglich basierend auf Eval-Kit/FreeRTOS Demo-Project*
 - *weitere Folgeprojekte danach per „Projekt kopieren“ (in Eclipse)*
- Build all, Clean & Rebuild
- Program Memory des Controllers Flashen
 - *über Makefile-Rule „flash“ OpenOCD mit entspr. Options starten*
 - *aufgerufen direkt aus Eclipse per „External Command“*
- Debug-Session
 - *GDB Backend starten*
 - *über Makefile Rule „gdbserver“ OpenOCD im gdbserver-Modus starten*
 - *Debug-Frontend starten (Bestandteil von Eclipse)*
 - *Switch Perspective (Debug-Frontend <--> C/C++)*
 - *Debug-Session beenden*

Folien-Anhang

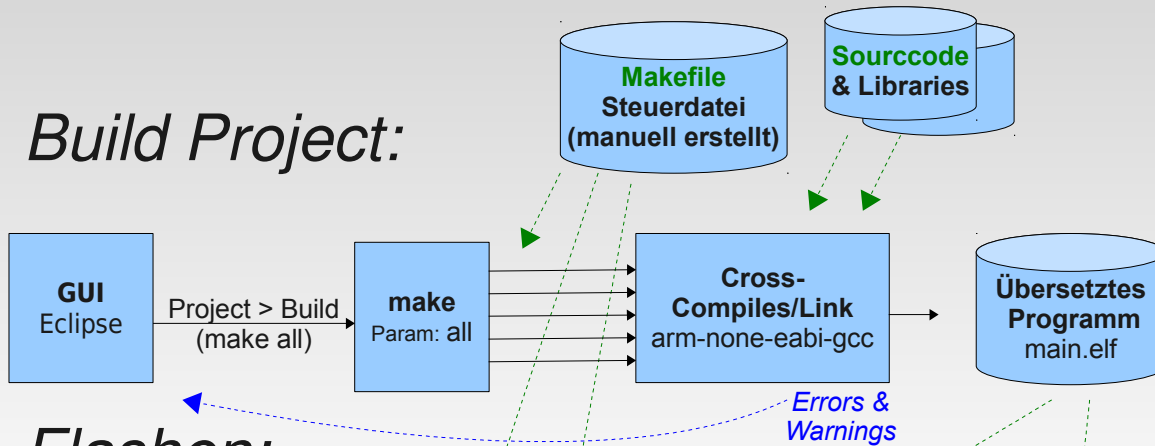
- Eclipse - OpenOCD – Workflow
- Installation und Trouble Shooting auf Windows und Linux Host
- Makefile Rules für Flashen/Debuggen via OpenOCD
- Compiler Options für Sourcecode-Debugging
- Screenshots wichtiger Eclipse-Config.-Dialoge

Open Source Tools für ARM Cortex-M3 basierte Mikrocontroller

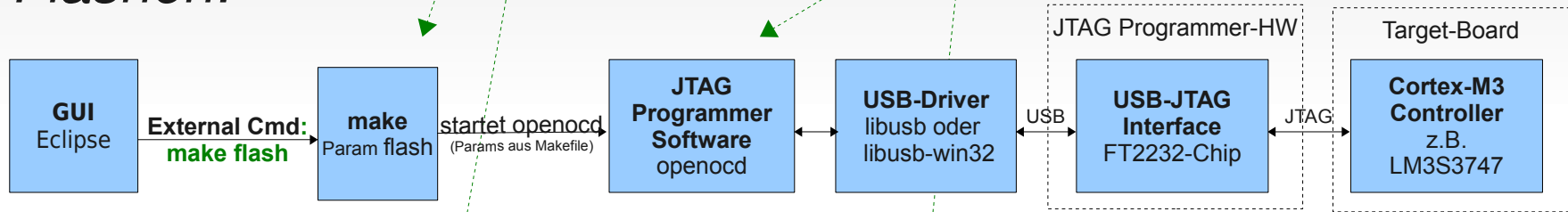
Fragen?

Eclipse - OpenOCD - Workflow

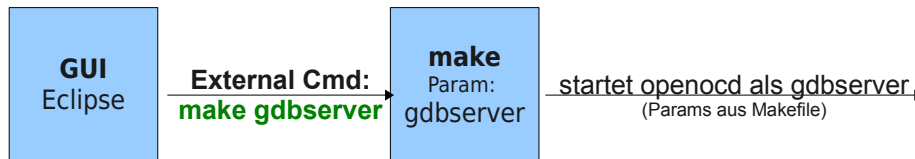
➤ Build Project:



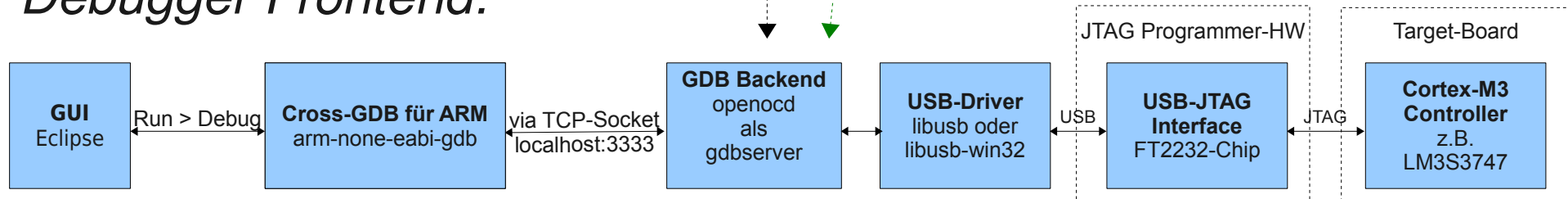
➤ Flashen:



➤ Debugger Backend starten:



➤ Debugger Frontend:



Installation auf Windows Host-System

- Eclipse, Eclipse CDT
 - <http://www.eclipse.org/downloads/>
 - entweder *Eclipse IDE for C/C++ Developers*
 - oder: beliebiges Eclipse und *CDT-Plugin* nachinstallieren
- Yagarto (ARM Cross-Toolchain für Windows)
 - <http://www.yagarto.de/>
 - „YAGARTO GNU-ARM-Toolchain“ (beinhaltend u.a. *arm-none-eabi-gcc*)
 - „YAGARTO Tools“ (beinhaltend u.a. *make.exe*)
- OpenOCD 0.4.0 (für Windows übersetzt)
 - <http://www.freddiechopin.info/index.php/en/download/category/4-openocd>
 - Übersetzt für *libusb-win32-Treiber*
 - *libusb-win32-Treiber* unter *Installationsverzeichnis*
 - auch geeignet für Win64, Treibersignierung, zwecks Verwendung beim Booten jew. abschalten

Welche Eclipse Installer-Package?

- Download ab: <http://www.eclipse.org/downloads/>
- Sie programmieren...
 - *Nur in C/C++:*
 - *Eclipse IDE for C/C++ Developers* (enthält bereits das erforderliche *CDT Plugin*)
 - *Sowohl in C/C++ als auch in Java:*
 - 1. *Eclipse IDE for Java EE Developers* installieren
 - 2. das fehlende **CDT Plugin nachträglich in Eclipse** zufügen per:
Help > Install Software...
Work with: -- All available Sites --
Programming Languages > [x] Eclipse C/C++ Development Tools
Next> ... Finish...
- Anmerkung zu Eclipse Installer-Packages:
 - *Eclipse-Installationspackages sind bloss Archiv-Files (*.zip , *.tar.gz)*
 - *Deshalb manuell entpacken und verschieben auf z.B. auf C:\Progs\ Linux: /opt/*
 - *und danach manuell eine Startverknüpfung erstellen auf*
C:\Progs\eclipse\eclipse.exe (resp. bei Linux auf */opt/eclipse/eclipse*)

Bemerkungen zur make Utility für Windows

- make (etc.) wurde für POSIX-Systeme entwickelt
 - *also für UNIX, Linux, BSD, ...*
- Für Windows deshalb nur mit entspr. Anpassung lauffähig
 - *z.B. über API-Libraries: „MinGW“ (Minimal Gnu for Windows) oder „CYGWIN“*
- „Yagarto-Tools“ enthalten ein derart angepasstes 'make.exe'
- Nach Installation auf Windows-Command-Ebene kontrollieren ob make gefunden wird und korrekte Version hat:
 - *make -version --> Version 3.81 (ok)*
 - *andernfalls Windows PATH-Variable manuell ergänzen!*
- Bei Problemen ev. alternative MinGW-Packages installieren:
 - *z.B. das MinGW-Package „MSYS“*
 - *hingegen ist der MinGW GCC Compiler kein (ARM-)Cross-Compiler!*

Häufige Probleme unter Windows

- Leerschläge/Umlaute/Sonderzeichen (abgesehen von '-' und '_')
 - *in Workspace-, Ordner- und Dateinamen*
- Tools nicht im System-Suchpfad
 - *im Speziellen: Cross-Compiler, make, OpenOCD*
 - *Umgebungsvariable „PATH“ entspr. anpassen*
- Versionskonflikt GDB / OpenOCD
 - *ggf. mit verschiedenen Versionen spielen*
- Eclipse-Crash nach zufügen von Eclipse CDT
 - *<WORKSPACE>/./metadata/.log inspizieren, googlen*
 - *Startparameter in eclipse.ini entspr. anpassen*
- Probleme OpenOCD mit USB-Treiber o. Hardware
 - *Zu alte OpenOCD-Version*
 - *Konfiguration unterstützt den Libusb-Treiber nicht*
 - *fehlender, nicht installierter, falscher oder nicht geladener libusb-win32 Treiber*
 - *Alter openocd-Prozess vor Neustart der Debug-Session beenden z.B. mittels:*
 - *External Command: 'taskkill /IM openocd.exe /F'*

Probleme unter Windows (...)

■ Perfide:

- *'make.exe' aus Yagarto-Tools funktioniert nicht da im Windows PATH der letzte Eintrag mit Anführungszeichen endete...*

■ Stolperfalle: Windows Vista / Windows 7

- *Windows gaukelt auf dem Desktop lokalisierte Ordnernamen vor*
 - *Auf OS-Ebene heissen diese jedoch C:\USERES oder C:\Program Files etc*
- *Besser eigene Stamm-Ordner verwenden:*
 - *Userdaten, Workspaces etc. z.B unter C:\Data*
 - *Auszuführende Programme (Eclipse, Toolchain, OpenOCD) z.B. unter C:\Progs*

■ Stolperfalle: Windows Vista / Windows 7 64 Bit

- *Obwohl der libusb-win32 Treiber nach manueller Angabe korrekt installiert wird, wird dieser doch nicht aktiv! (vgl. Gerätemanager!)*
- *In diesem Fall vor jeder Verwendung von OpenOCD Im Windows Bootmenu (F8) die Treibersignierung abschalten!!*

Installation unter Linux

■ Eclipse

- Download analog Windows ab <http://www.eclipse.org/downloads/>
 - auf `/opt/` entpacken und manuell Startverknüpfung im Desktop-Menu erstellen
 - Nicht empfohlen: verwenden von Eclipse aus der Linux-Distribution (Plugin-Konflikte und Eclipse dadurch nicht unabhängig von Linux upgradebar)

■ make - aus Linux-Distribution verwenden

- Installation bei Ubuntu z.B. per: `apt-get install build-essentials`

■ ARM GCC Cross-Compiler

- „*Codesourcery G++ Lite*“ für Target-OS „*EABI*“ downloaden <http://www.codesourcery.com/sgpp/lite/arm/portal/subscription3053>
 - Package „*EABI*“ ist für betriebssystemlose Targets ohne MMU geeignet!
 - auf `/opt/` entpacken und im File `~/.profile` die `PATH`-Variable setzen; kontrollieren!

■ OpenOCD (mind. V0.4.0)

- In Linux-Distribution (ev.) enthalten (z.B. ab Ubuntu 10.04)
- Wenn nicht oder wenn Probleme, diese wieder deinstallieren, OpenOCD-Source herunterladen, kompilieren, installieren...

Nur im Problemfall: OpenOCD für/unter Linux kompilieren

- OpenOCD 0.4.0 ab Quellcode Übersetzen
 - *Bei aktueller Ubuntu-Linux in Distribution nicht nötig!*
 - *Andernfalls oder falls neuerer OpenOCD-Relese nötig, OpenOCD Quellcode wie folgt selber kompilieren*

```
$ sudo apt-get install git libftdi-dev libusb-dev
$ git clone git://openocd.git.sourceforge.net/gitroot/openocd/openocd
$ cd openocd
$ ./bootstrap
$ git submodule init
$ git submodule update
$ ./configure --enable-ft2232_libftdi
$ make
$ make install
```

- *damit wird openocd installiert auf /usr/local/bin und /usr/local/lib/openocd/*
- *Achtung: OpenOCD aus Linux-Distribution entfernen, andernfalls ev. Konflikt!*

Flashen

Programmieren des Flash-EEPROMs des Controllers

- Welche Flash-Software verwenden?
 - *Flash-Tool des Controller-Herstellers?*
 - *Kommunikation erfolgt hierbei meist via Asynchrone serielle Schnittstelle*
 - *Oder Open Source Lösung „OpenOCD“ via JTAG-Interface?*
 - *Sinnvoll wenn auch Sourcecode-Debugging erwünscht*
- Aufruf der Flash-Software via **make** & **Makefile**
 - *Hierzu im **Makefile** geeignete Rule 'flash' erstellen,*
 - *und danach Flashen auf Konsoleebene per „**make flash**“*
 - *ermöglicht auch Aufruf aus Eclipse per „**External Command**“*
 - *„**External Commands**“ „**Flash**“ definieren mit Inhalt „**make flash**“*
 - *Ebenso kann OpenOCD als gdbserver gestartet werden z.B. via '**make gdb**'*

Makefile Rules für Flashen und Debuggen via OpenOCD (Beispiel: TI-Stellaris MCUs)

```
TARGET = main
OPENOCD = openocd

OOCD_INIT += -f board/ek-lm3s3748.cfg
OOCD_INIT += -c "jtag_khz 3000"
#OOCD_INIT = -f interface/olimex-arm-usb-ocd.cfg
#OOCD_INIT += -f target/stm32.cfg
#OOCD_INIT += -c "jtag_khz 1200"
OOCD_INIT += -c init
OOCD_INIT += -c "reset init"

OOCD_FLASH = -c "reset halt"
OOCD_FLASH += -c "targets"
OOCD_FLASH += -c "flash write_image erase $(TARGET).elf"
OOCD_FLASH += -c "verify_image $(TARGET).elf"
OOCD_FLASH += -c "reset run"
OOCD_FLASH += -c shutdown

flash: $(TARGET).elf
    $(OPENOCD) $(OOCD_INIT) $(OOCD_FLASH)

gdbserver:
    $(OPENOCD) $(OOCD_INIT)
```

➤ Voraussetzung: Korrekte Installation von OpenOCD

- inkl. für USB-JTAG-Interface passenden/geladenen libusb-Treiber!
- ev. OpenOCD interaktiv im Commandmode testen, nach Start mit „openocd -f board/ek-lm3s3748.cfg“ per „telnet 4444“

Compiler-Optionen für Debugging

■ Compiler-Optionen für Debugging

- *im Makefile meist via Variable CFLAGS:*
- **-g** *fügt Debug Info in Ausgabedatei (ELF-File) zu*
 - *Fügt Symbole für Quellcode-Zeilennummern, Variablennamen etc. zu*
- **-O0** *Schaltet Compiler-Optimierung aus*
 - *Verhindert „wegoptimieren“ von Variablen und Umstellung der Code-Reihenfolge*
 - *Ein Debugging ist auch mit Optimierung möglich (-Os oder -O2),
Der Ablauf erscheint aber ev. nicht sequenziell*

■ Vollständiger Rebuild nach jeder Makefile-Änderung

- *Project > Clean (führt 'make clean' aus)*
- *Project > Build All (führt 'make all' aus)*

Weshalb überhaupt Makefile manuell erstellen?

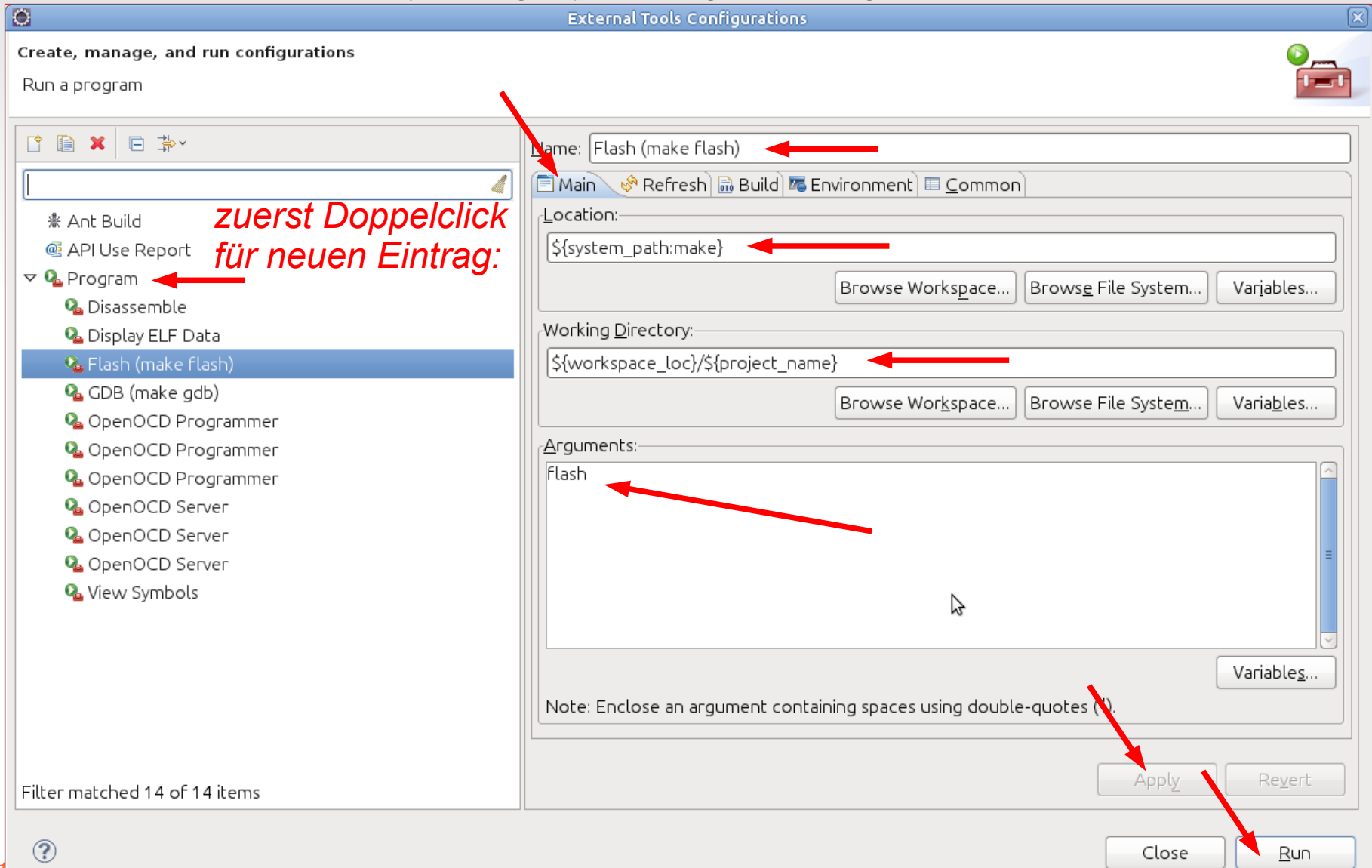
- Bei Embedded-Projekten (Cross-Compilation) Makefile immer manuell unterhalten!
 - *und nicht durch Eclipse automatisch erstellen lassen!*
 - *Damit bleibt Projekterstellung unabhängig vom Eclipse-Release und Eclipse-Konfiguration!*
 - *Erleichtert langfristige Wartung des Projektes! (minimale Host-Abhängigkeit)*
 - *Für einfache Projektänderungen kein Eclipse nötig*
 - *Projekt kann ausserhalb Eclipse auf Kommandozeile erstellt werden mit:*
 - „make clean“ *um den vorhandenen Objektcode des Projektes zu löschen*
 - „make all“ *um alles im Projekt neu zu compilieren/linken*
 - *Makefile z.B. auf Demo-Projekt des Eval-Kits aufbauen*
 - *Demo-Projekt beinhaltet meist Makefile, Hersteller-Libraries, Demo-Sourcecode*
 - *Leider unterstützen nicht alle Kit-Hersteller direkt eine GNU-Toolchain....*
 - *In diesem Fall das Makefile z.B. aus Opensource ARM-Projekten ableiten, z.B. jene von Martin Thomas http://gandalf.arubi.uni-kl.de/avr_projects/arm_projects/*

Eclipse Dialoge

- Nachfolgend einige für Cross-Debugging wichtigste Eclipse-Dialoge...

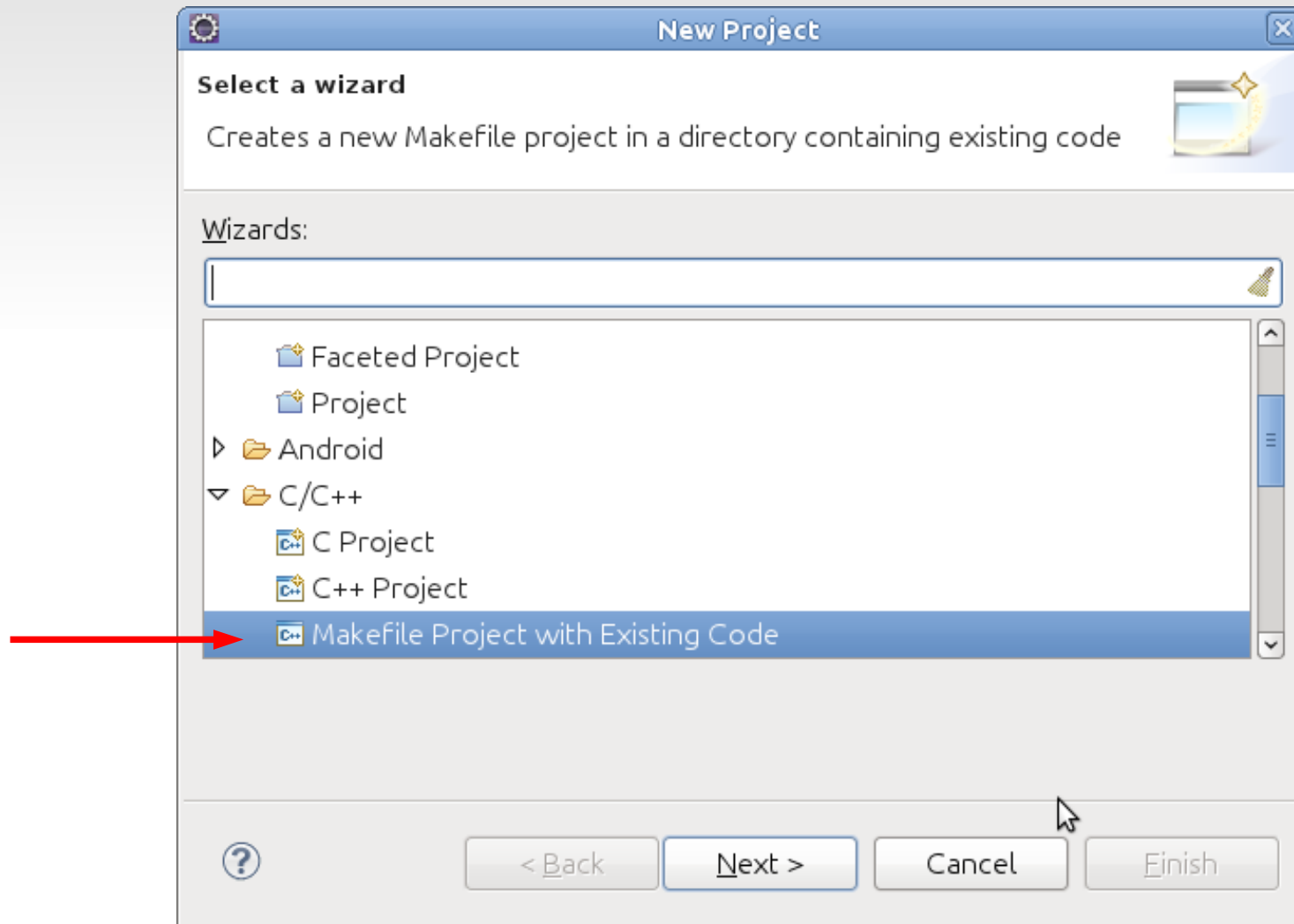
Konfig. Aufruf Flash / Debugger-Backend

- Run > External Tool Configuration...
 - Neuer Eintrag für „Flash (make flash)“ erstellen
 - dito für „GDB (make gdb)“ mit Arguments: gdb



Neues C/C++ Makefile Projekt erstellen

- File > New > Project... > C/C++ Makefile Project...



Scanner für Include Files einrichten

- Project > Properties...
 - C/C++ Build > Discovery Options

Properties for fs11_mc2_v5_ElevatorQueue

Discovery Options

Configuration: Default [Active] Manage Configurations...

Discovery profiles scope
Configuration-wide

Configuration:
Default

Automated discovery of paths and symbols

- Automate discovery of paths and symbols
- Report path detection problems

Discovery profile: GCC per file scanner info profile

Clear discovered Clear

Discovery profile options

- Enable build output scanner info discovery

Load build output from file Load

Browse... Variables...

Cancel OK

Konfig. Debugger Frontend (1/3)

- Run > Debug Configuration...

Create, manage, and run configurations

Name: fs11_mc2_v5_ElevatorQueue Default

Main Debugger Source Common

C/C++ Application:

main.elf (wie in Makefile def.) Search Project... Browse...

Project:

fs11_mc2_v5_ElevatorQueue projektspezifisch (leider) Browse...

Build (if required) before launching

Build configuration: Default

Enable auto build Disable auto build

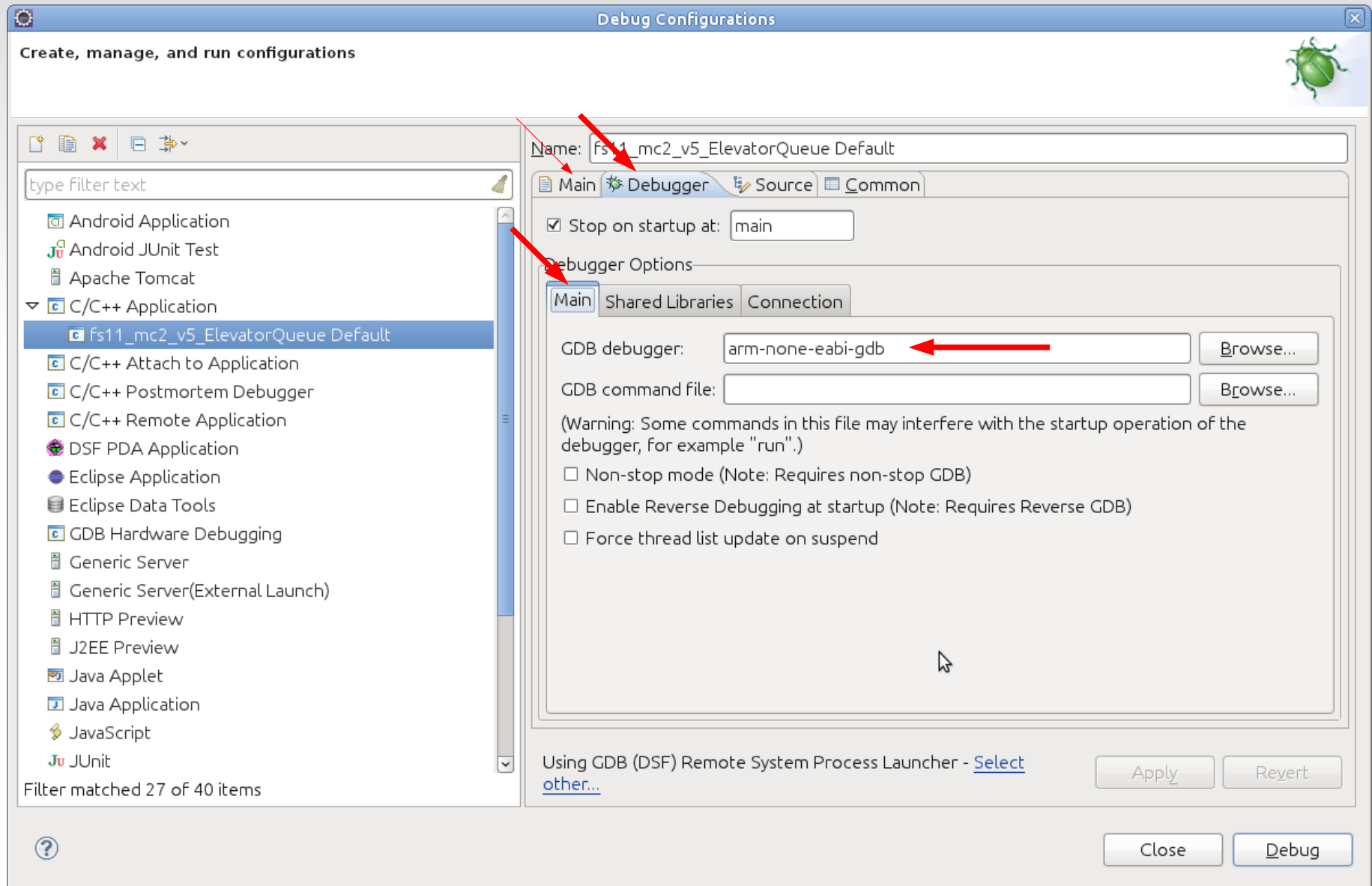
Use workspace settings [Configure Workspace Settings...](#)

Using GDB (DSF) Remote System Process Launcher - [Select other...](#)

Apply Revert

Close Debug

Konfig. Debugger Frontend (2/3)



Konfig. Debugger Frontend (3/3)

The screenshot shows the Eclipse IDE's 'Debug Configurations' dialog box. The left sidebar lists various configuration types, with 'fs11_mc2_v5_ElevatorQueue Default' selected under 'C/C++ Application'. The main area shows the configuration for this selected item. The 'Name' field is 'fs11_mc2_v5_ElevatorQueue Default'. The 'Debugger' tab is active, and the 'Connection' sub-tab is selected. The 'Stop on startup at:' field is set to 'main'. The 'Type' is 'TCP'. The 'Host name or IP address' is 'localhost' and the 'Port number' is '3333'. At the bottom, there are buttons for 'Apply', 'Revert', 'Close', and 'Debug'. A red arrow points to the 'Debug' button.

3rd Party Opensource Quellen

- Open Source RTOS
 - *FreeRTOS* - www.freertos.org
- Open Source FAT File System
 - *FatFs* - http://elm-chan.org/fsw/ff/00index_e.html
- ARM Projects by Martin THOMAS
 - http://gandalf.arubi.uni-kl.de/avr_projects/arm_projects/
- Achtung: Lizenztexte beachten!
 - *GPL ≠ Modified GPL ≠ LGPL ≠ Apache License ≠ BSD License ...*