

Embedded Systems / Eingebettete Systeme

Studiengang Informatik
Campus Minden

Matthias König



FH Bielefeld
University of
Applied Sciences

RÜCKBLICK

Was charakterisiert Eingebettete Systeme?

- “Eingebettete Systeme sind **informationsverarbeitende Systeme**, die in ein größeres Produkt **integriert** sind.”
[Quelle: Marwedel, Eingebettete Systeme]
- “Ein HW- und SW-System / Eingebettetes System besteht im Allgemeinen aus **Hardware, Software und eingebetteten Komponenten**. Ein Projekt, welches als Projektgegenstand ein HW- und SW-System / Eingebettetes System hat, wäre also zum Beispiel die Entwicklung des Eurofighters oder eines Schiffes. Weiterhin wird ein HW- und SW-System / Eingebettetes System charakterisiert durch die **Erfassung der Umwelt über Sensoren und Aktoren zur Interaktion mit seiner physischen Umgebung**. Dadurch werden auch kleinere Systeme adressiert, wie z.B. ein Mikrocontroller, der mit Hilfe seines Programms die Airbagauslösung im Kraftfahrzeug steuert.”

[Quelle: V-Modell® XT, Version 1.3]

Welche Anforderungen?

Effizient hinsichtlich

- Energieverbrauch (Energie kostet...)
- Codegröße (z.B. Festplatte oft ungewünscht wg. Verschleiß)
- Laufzeit (Einhalten von Zeitbedingungen)
- Gewicht (z.B. leicht trag- oder verbaubar)
- Preis (nur Teil eines System)

[Quelle: Marwedel, Eingebettete Systeme]

Welche Anforderungen?

Verlässlich hinsichtlich

- Zuverlässigkeit (nicht ausfallen)
- Wartbarkeit (schnell repariert bei Ausfall)
- Verfügbarkeit (hohe Zuverlässigkeit, hohe Wartbarkeit)
- Sicherheit (keinen Schaden bei Ausfall)
- Integrität (Gewährleistung von Daten- und Kommunikationssicherheit)

[Quelle: Marwedel, Eingebettete Systeme]

ENTWICKLUNGSUMGEBUNG UND TOOLCHAIN

Debug - TivaCBLINK/src/startup_gcc.c - Eclipse /Users/matthias/Documents/workspace

Debug C/C++ Resource Quick Access

Debug

OpenOCD [Program]
/opt/local/bin/openocd
TivaCBLINK Release [GDB Hardware Debugging]
TivaCBLINK.elf
Thread [1] (Suspended : Step)
ResetISR() at startup_gcc.c:252 0x302
gdb

Variables Breakpoints Registers Modules

Name	Value	Description
General Registers		
r0	0	
r1	536870912	
r2	0	
r3	0	
r4	0	

Name : r1
Hex:0x20000000
Decimal:536870912
Octal:04000000000
Binary:10000000000000000000000000000000
Default:536870912

Project Explorer

TivaCBLINK
Binaries
Includes
src
startup_gcc.c
TivaCBLINK.c
Debug
Release
META-INF
build.properties
mem.ld
spec.d

TivaCBLINK.c startup_gcc.c

```

extern uint32_t _edata;
extern uint32_t _bss;
extern uint32_t _ebss;

//*****
//
// This is the code that gets called when the processor first starts execution
// following a reset event. Only the absolutely necessary set is performed,
// after which the application supplied entry() routine is called. Any fancy
// actions (such as making decisions based on the reset cause register, and
// resetting the bits in that register) are left solely in the hands of the
// application.
//
//*****
void
ResetISR(void)
{
    uint32_t *pui32Src, *pui32Dest;

    //
    // Copy the data segment initializers from flash to SRAM.
    //
    pui32Src = &_edata;
    for(pui32Dest = &_data; pui32Dest < _edata; )
    {
        *pui32Dest++ = *pui32Src++;
    }

    //
    // Zero fill the bss segment.
    //
    __asm(
        "    ldr    r0, =_bss\n"
        "    ldr    r1, =_ebss\n"
        "    mov    r2, #0\n"
        "    .thumb_func\n"
    );
}

```

Outline Disassembly

Enter location here

```

000002fc: nop
000002fc: b.n 0x2fc <IntDefaultHandler>
000002fe: nop
000002fe:
ResetISR:
00000300: ldr r1, [pc, #104] ; (0x36c <ResetISR+108>)
00000302: ldr r3, [pc, #108] ; (0x370 <ResetISR+112>)
00000304: cmp r1, r3
00000306: push {r4, r5, r6}
00000308: bcs.n 0x342 <ResetISR+66>
0000030a: adds r2, r3, #3
0000030c: adds r4, r1, #4
0000030e: subs r5, r2, r4
00000310: ldr r0, [pc, #96] ; (0x374 <ResetISR+116>)
00000312: bic.w r2, r5, #3
00000316: adds r6, r2, #4
00000318: ldr r5, [r0, #0]
0000031a: movs r3, #4
0000031c: cmp r3, r6
0000031e: str r5, [r1, #0]
00000320: ubfx r2, r2, #2, #1
00000324: beq.n 0x342 <ResetISR+66>
00000326: cbz r2, 0x332 <ResetISR+50>
00000328: ldr r2, [r0, #4]
0000032a: movs r3, #8
0000032c: cmp r3, r6
0000032e: str r2, [r4, #0]
00000330: beq.n 0x342 <ResetISR+66>
00000332: adds r2, r3, #4
00000334: ldr r5, [r3, r0]
00000336: ldr r4, [r0, r2]
00000338: str r5, [r3, r1]
0000033a: adds r3, #8
0000033c: mov r3, r6

```

Console Problems EmbSys Registers

TivaCBLINK Release [GDB Hardware Debugging] gdb

The target endianness is set automatically (currently little endian)
Note: automatically using hardware breakpoints for read-only addresses.

Memory

Monitors

0x20000000 : 0x20000000

0 - 3	4 - 7	8 - B	C - F
DFF84040	DFF84050	D0F80080	B8F1000F
17D04768	4745F7D0	226002F1	040257F8
048BC4F8	0480A560	D4F80880	18F0010F
FAD18F42	28BF00F1	08074760	013B03B1
E2E700BE	00D00F40	010042A4	5C000020
5C000020	00000020	00010020	F5020000
F9020000	FD020000	FD020000	FD020000

0x20000000 : 0x20000000

0x20000000	4040F8DF	8000
0x20000004	5040F8DF	8000
0x20000008	8000F800	0000
0x2000000C	0F00F1B8	0000
0x20000010	6847D017	0000
0x20000014	D0F74547	0000
0x20000018	F1026022	0000

Elemente der Toolchain

- “Versteckt” in der IDE
 - Cross-Compiler und -Linker
 - Debugger
 - Flash-Werkzeug
 - Board circuit debug interface, z.B JTAG (IEEE 1149)
 - Build-Management-Tools (z.B. make)
- Manuelle Konfiguration oft mühsam, s. folgendes Beispiel

Kompilierung

1. Lexikalische (Scanner), syntaktische (Parser) und semantische Analyse des Quellendes mit Ergebnis Syntaxbaum
2. Ggf. Übersetzung des Syntaxbaums in Zwischencode und Optimierung des Zwischencodes
3. Übersetzung des Zwischencodes/Syntaxbaums in Maschinencode.
4. Ggf. Linken des Maschinencodes mit Bibliotheken
5. Ergebnis: Lauffähiger Binärcode

Cross-Compiler (und -Linker)

- Cross-Compiler (und -Linker)
 - läuft auf Host-System (z.B. Windows, Linux, OS X),
 - baut Programm/Kompilat für Target-(Embedded-)System (z.B. Atmega328p, ARM-Cortex-M4 etc.).
- Vorteile
 - schnelleres Kompilieren durch mehr Ressourcen (Leistung, Speicher) auf dem Host-System als auf dem Target-System

Cross-Compiler: Beispiel gcc arm

1. Präprozessing

Eingabe `main.c`, Ausgabe: `main.i` (Quellcode mit Includes)

2. Kompilierung

Eingabe: `main.i`, Ausgabe: `main.s` (Assemblersprache)

3. Assemblierung

Eingabe: `main.s`, Ausgabe: `main.o` (Maschinensprache)

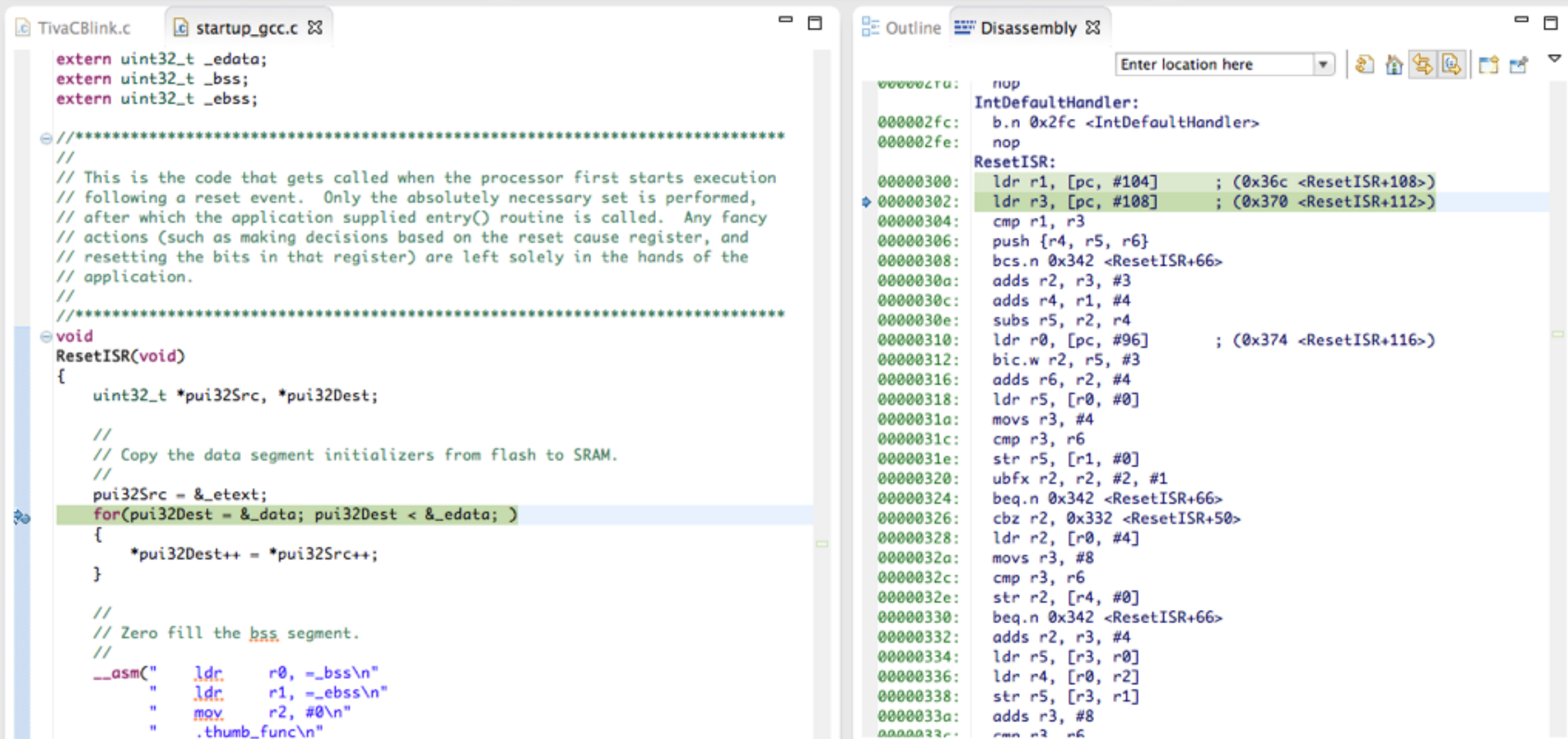
4. Linken

Eingabe: `main.o`, Ausgabe: `main.elf` (Exe-/Linkable Format)

5. Konvertierung

Eingabe: `main.elf`, Ausgabe: `main.bin` (Binärdatei zum Flashen)

Cross-Compiler: C- und Assemblercode



The image shows a cross-compiler IDE with two main panes. The left pane displays C code for a startup routine, and the right pane displays the corresponding assembly code. The C code defines external variables, a comment block, and a `ResetISR` function that copies data from flash to SRAM and zero-fills the BSS segment. The assembly code shows the implementation of these operations using ARM instructions, with memory addresses and offsets visible on the left of each line.

```
extern uint32_t _edata;
extern uint32_t _bss;
extern uint32_t _ebss;

//*****
//
// This is the code that gets called when the processor first starts execution
// following a reset event. Only the absolutely necessary set is performed,
// after which the application supplied entry() routine is called. Any fancy
// actions (such as making decisions based on the reset cause register, and
// resetting the bits in that register) are left solely in the hands of the
// application.
//
//*****

void
ResetISR(void)
{
    uint32_t *pui32Src, *pui32Dest;

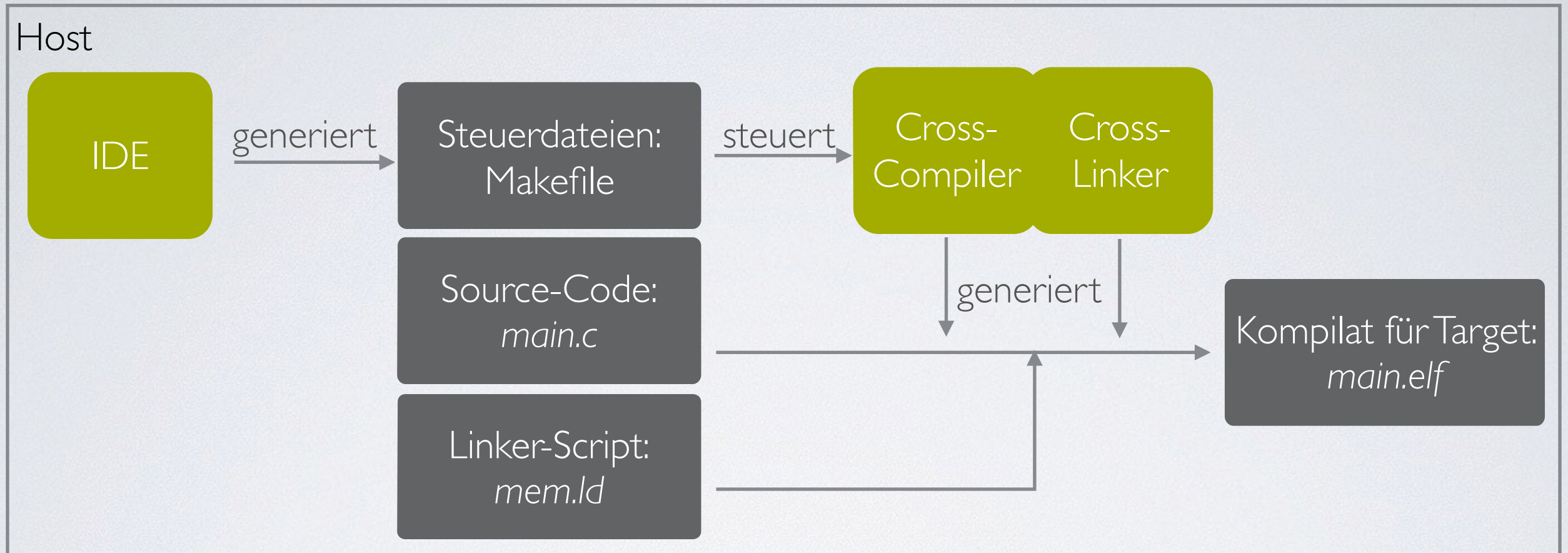
    //
    // Copy the data segment initializers from flash to SRAM.
    //
    pui32Src = &_etext;
    for(pui32Dest = &_data; pui32Dest < &_edata; )
    {
        *pui32Dest++ = *pui32Src++;
    }

    //
    // Zero fill the bss segment.
    //
    __asm("    ldr    r0, =_bss\n"
        "    ldr    r1, =_ebss\n"
        "    mov    r2, #0\n"
        "    .thumb_func\n")
}
```

Disassembly view:

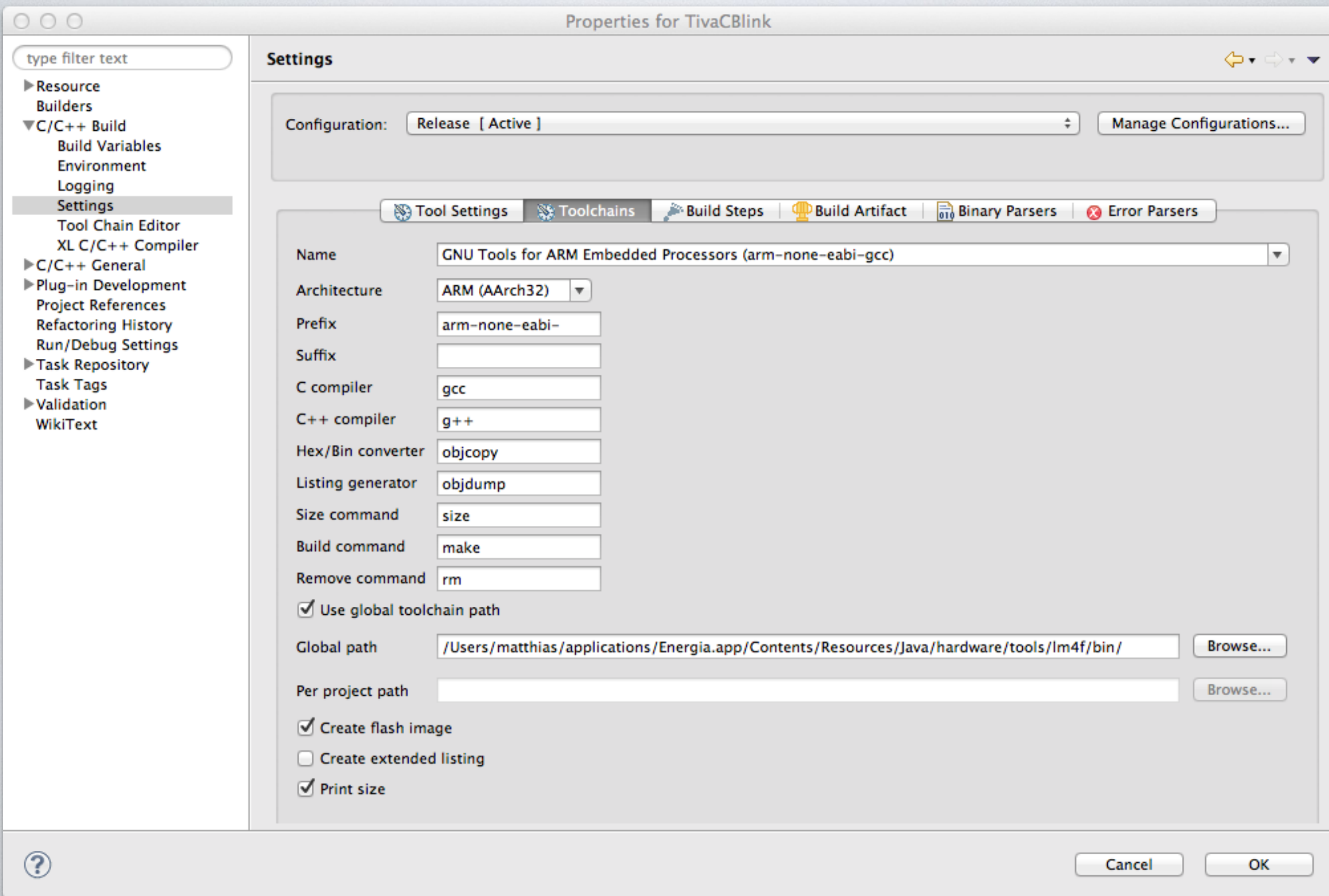
```
000002fc: nop
000002fe: b.n 0x2fc <IntDefaultHandler>
00000300: nop
00000302: ldr r1, [pc, #104] ; (0x36c <ResetISR+108>)
00000304: ldr r3, [pc, #108] ; (0x370 <ResetISR+112>)
00000306: cmp r1, r3
00000308: push {r4, r5, r6}
0000030a: bcs.n 0x342 <ResetISR+66>
0000030c: adds r2, r3, #3
0000030e: adds r4, r1, #4
00000310: subs r5, r2, r4
00000312: ldr r0, [pc, #96] ; (0x374 <ResetISR+116>)
00000314: bic.w r2, r5, #3
00000316: adds r6, r2, #4
00000318: ldr r5, [r0, #0]
0000031a: movs r3, #4
0000031c: cmp r3, r6
0000031e: str r5, [r1, #0]
00000320: ubfx r2, r2, #2, #1
00000322: beq.n 0x342 <ResetISR+66>
00000324: cbz r2, 0x332 <ResetISR+50>
00000326: ldr r2, [r0, #4]
00000328: movs r3, #8
0000032a: cmp r3, r6
0000032c: str r2, [r4, #0]
0000032e: beq.n 0x342 <ResetISR+66>
00000330: adds r2, r3, #4
00000332: ldr r5, [r3, r0]
00000334: ldr r4, [r0, r2]
00000336: str r5, [r3, r1]
00000338: adds r3, #8
0000033a: cmp r3, r6
```


Cross-Compiler

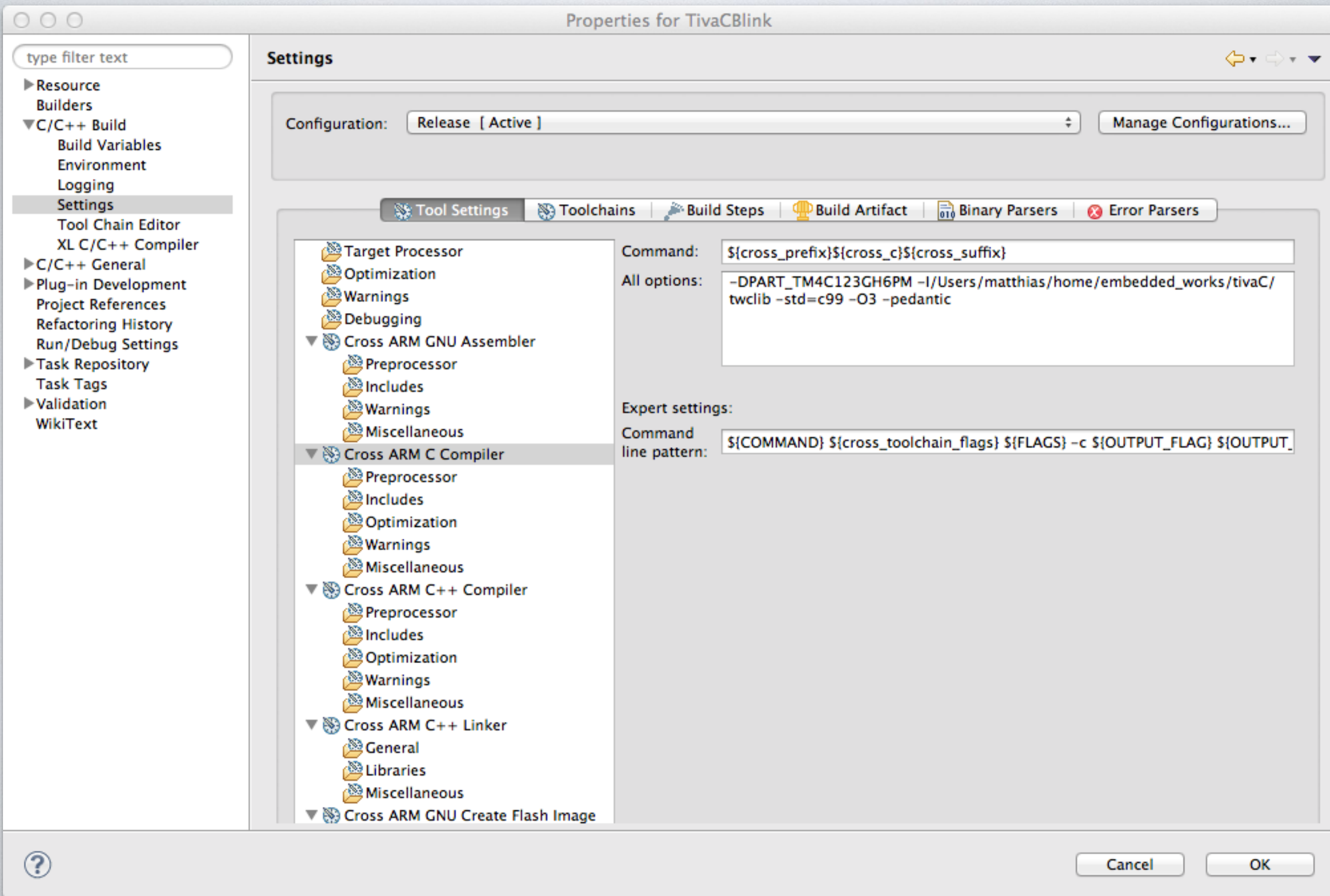


Hinweis: Normalerweise sind Werkzeuge der Toolchain einzelne Programme, die auch über ein Terminal/Shell manuell ausführbar sind.

Beispiel: Einstellungen für gcc arm in Eclipse



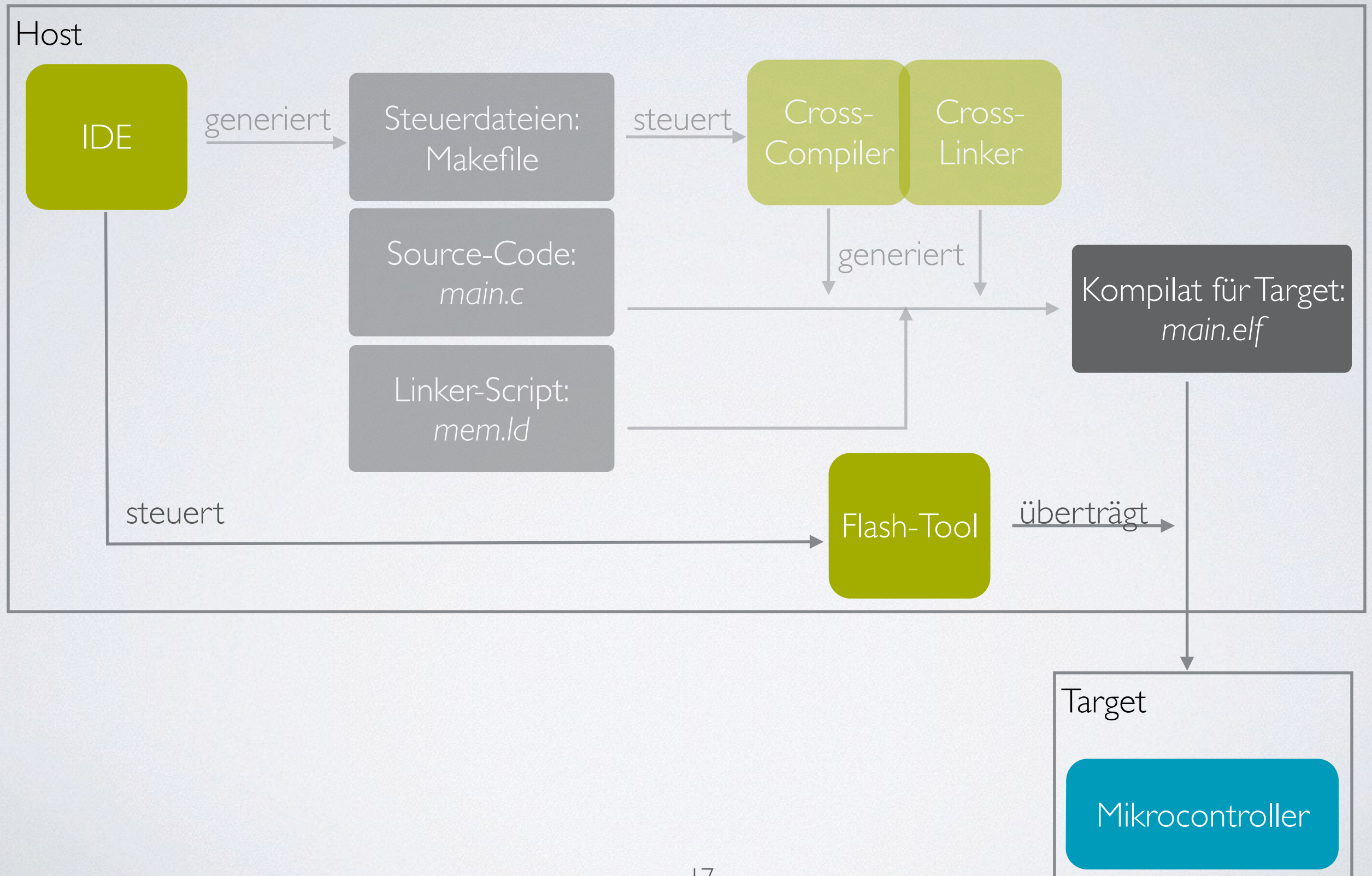
Beispiel: Einstellungen für gcc arm in Eclipse



Flash-Werkzeug

- Übertragen des Kompilats von dem Host-System auf das Target-System
 - mittels vordefinierter Schnittstelle (Soft- und Hardware)
 - abhängig von der Entwicklungshardware
 - auch möglich mittels Board circuit debug interface

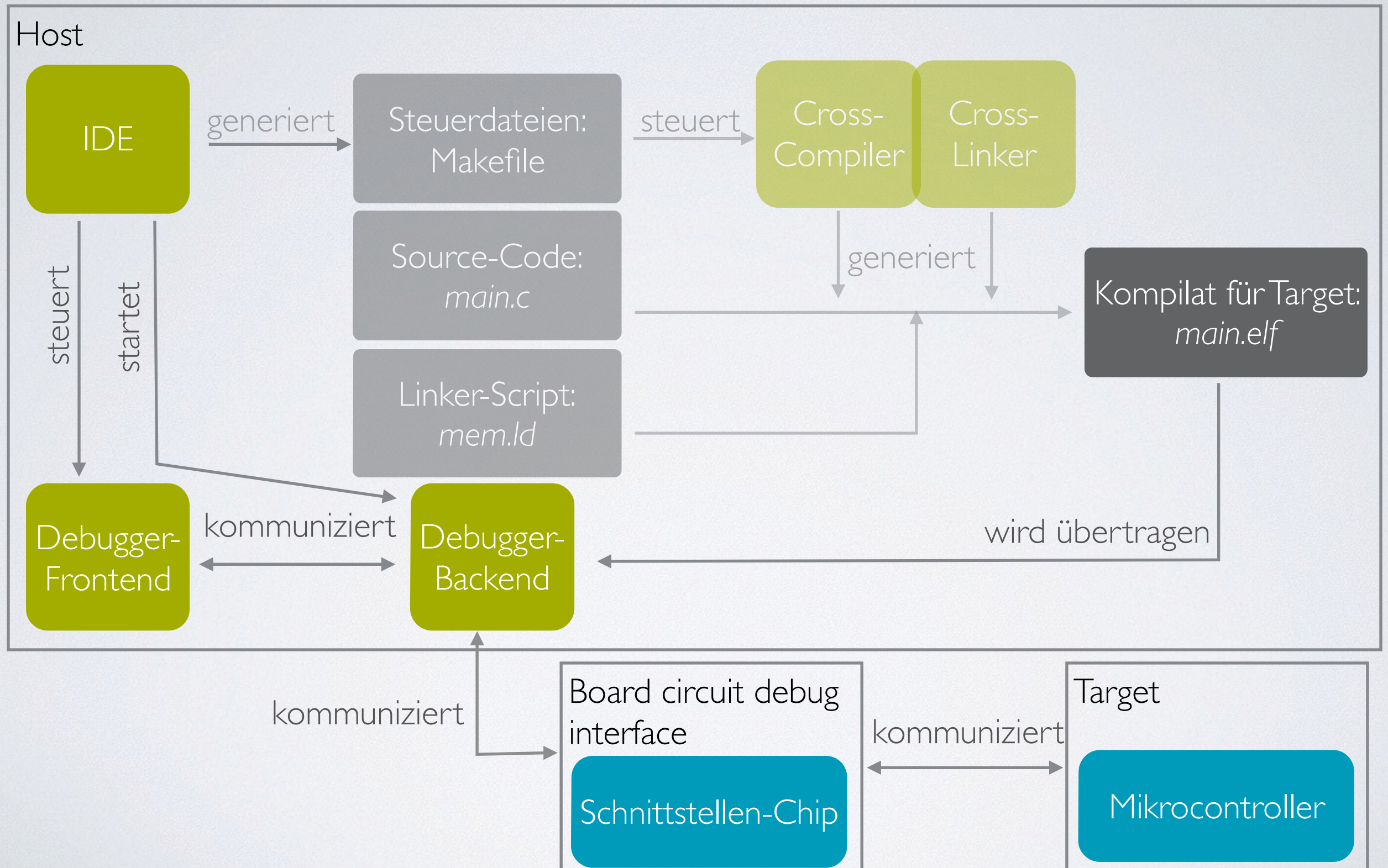
Cross-Compiler und Flashen



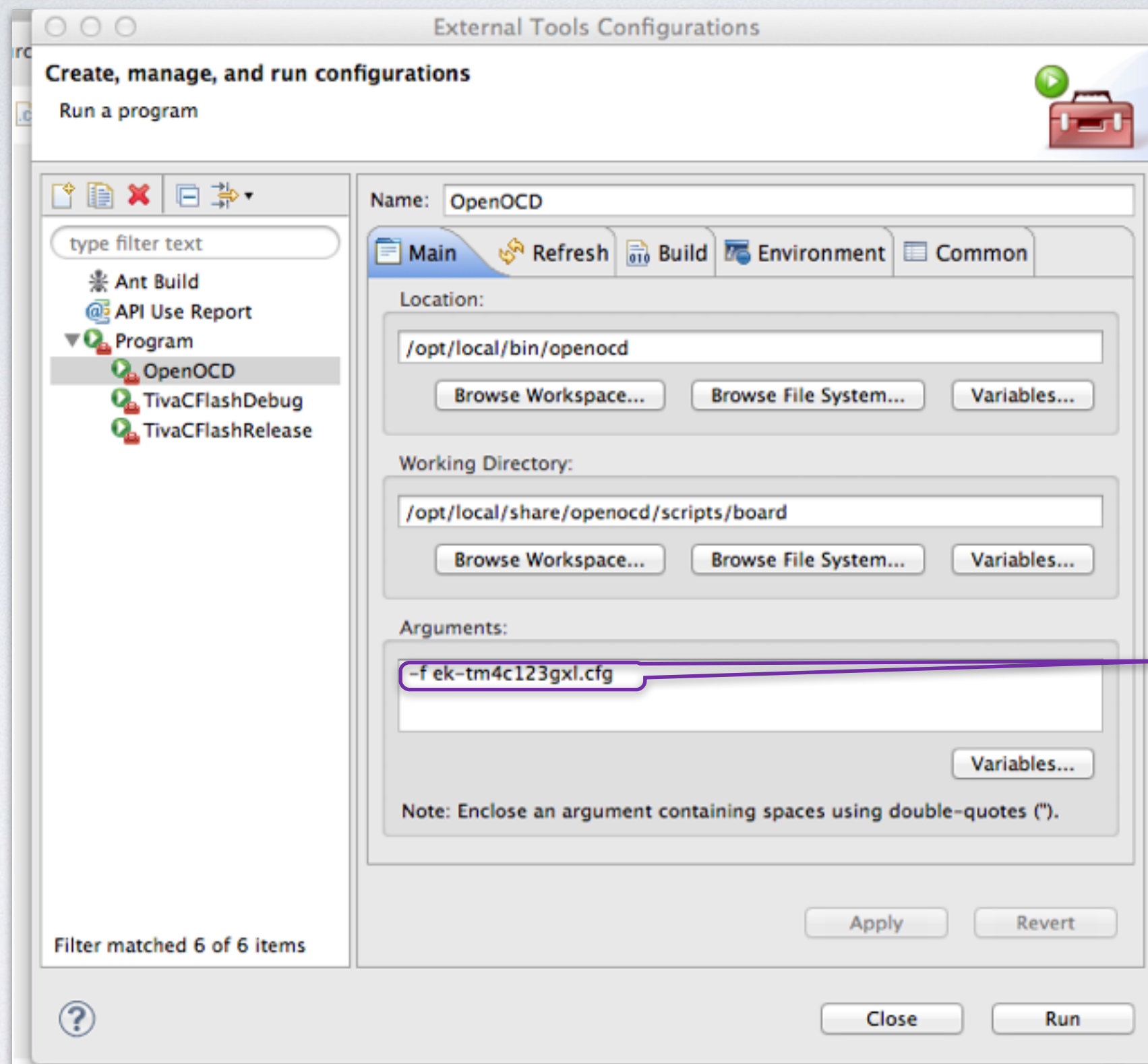
Debugger und Board circuit debug interface

- Debugger (Frontend)
 - läuft auf dem Host-System und
 - kommuniziert mit dem Target-System
 - über vordefinierte Schnittstelle.
- Board circuit debug interface (Backend)
 - wird angesprochen von Applikation auf dem Host-System,
 - welche mit dem Debugger kommuniziert.

Target-Debugging



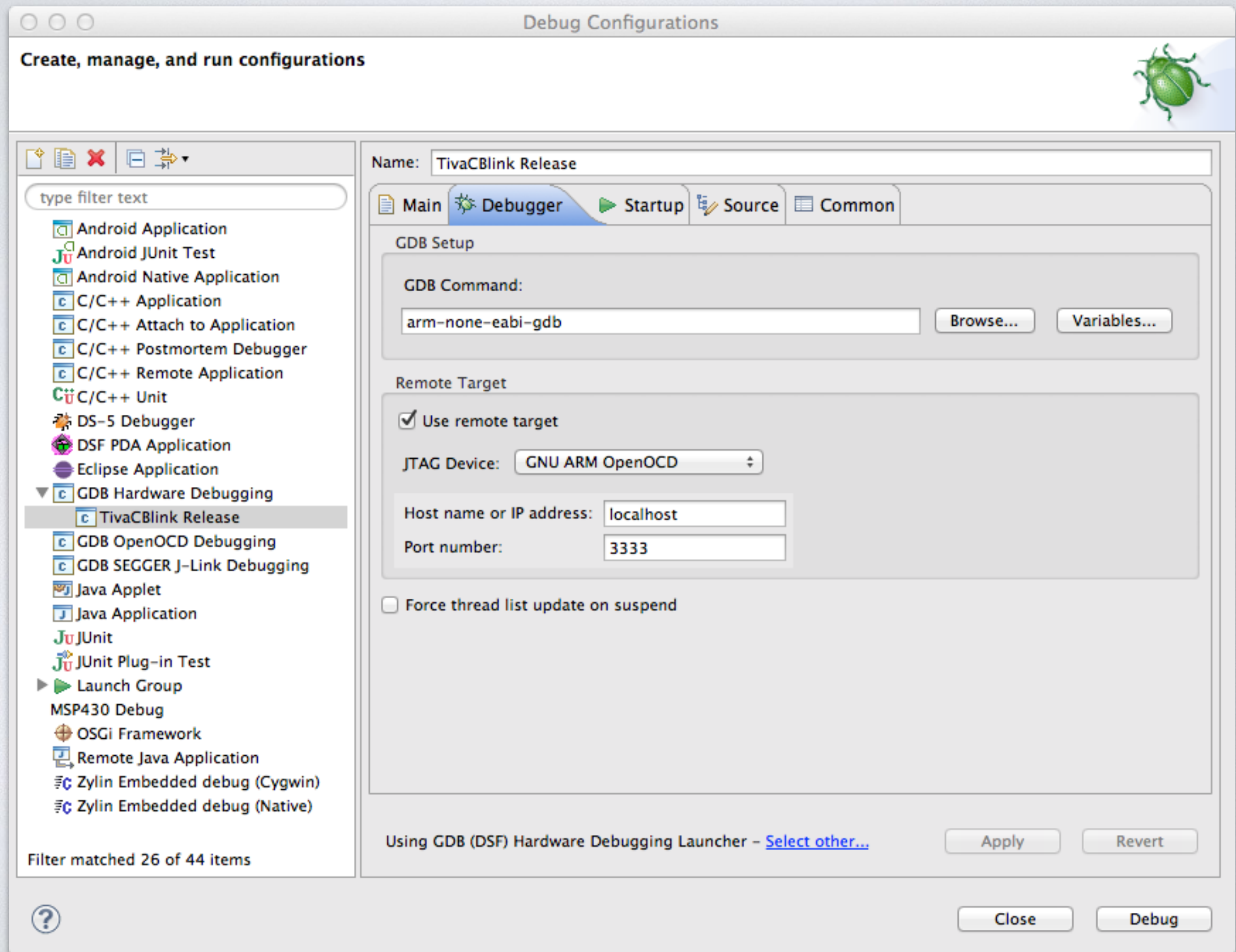
Beispiel: Einstellungen für openocd (Backend) in Eclipse



Konfigurationsdatei
für Target-System

Problems Tasks Console Properties Problem Details
OpenOCD [Program] /opt/local/bin/openocd
Open On-Chip Debugger 0.7.0 (2014-02-08-21:29)
Licensed under GNU GPL v2

Beispiel: Einstellungen für gdb arm (Frontend) in Eclipse



Target-Debugging in Eclipse

The screenshot shows the Eclipse IDE interface for target debugging. The top toolbar contains various icons for file operations, debugging, and execution. Below the toolbar, the 'Debug' tab is active, showing a tree view of the debug configuration. The tree view includes 'OpenOCD [Program]' and 'TivaCblink Release [GDB Hardware Debugging]'. Under 'TivaCblink Release', there is a sub-entry 'TivaCblink.elf' which is expanded to show 'Thread [1] (Suspended : Step)'. The current execution point is 'ResetISR() at startup_gcc.c:252 0x302'. The 'gdb' icon is visible at the bottom of the tree view.

On the right side, the 'Variables' tab is active, displaying a table of general registers. The table has two columns: 'Name' and 'Value'. The registers listed are r0, r1, r2, r3, and r4. The values for r0, r2, r3, and r4 are 0, while r1 has a value of 536870912. Below the table, the details for register r1 are shown, including its hex, decimal, octal, binary, and default values.

At the bottom, the 'startup_gcc.c' file is open, showing the following code:

```
extern uint32_t _edata;  
extern uint32_t _bss;  
extern uint32_t _ebss;  
  
//*****
```

Backend

Frontend

Target-Debugging in Eclipse

Debug - TivaCBLINK/src/startup_gcc.c - Eclipse - /Users/matthias/Documents/workspace

Debug | C/C++ | Resource | Quick Access

Debug | OpenOCD [Program] | /opt/local/bin/openocd | TivaCBLINK Release [GDB Hardware Debugging] | TivaCBLINK.elf | Thread [1] (Suspended : Step) | ResetISR() at startup_gcc.c:252 0x302 | gdb

Variables | Breakpoints | **Registers** | Modules

Name	Value	Description
General Registers		
r0	0	
r1	536870912	
r2	0	
r3	0	
r4	0	

Name : r1
Hex: 0x20000000
Decimal: 536870912
Octal: 04000000000
Binary: 10000000000000000000000000000000
Default: 536870912

Project Explorer

- TivaCBLINK
 - Binaries
 - Includes
 - src
 - startup_gcc.c
 - TivaCBLINK.c
 - Debug
 - Release
 - META-INF
 - build.properties
 - mem.ld
 - spec.d

TivaCBLINK.c | startup_gcc.c

```
extern uint32_t _edata;
extern uint32_t _bss;
extern uint32_t _ebss;

//*****
//
// This is the code that gets called when the processor first starts execution
// following a reset event. Only the absolutely necessary set is performed,
// after which the application supplied entry() routine is called. Any fancy
// actions (such as making decisions based on the reset cause register, and
// resetting the bits in that register) are left solely in the hands of the
// application.
//
//*****
void
ResetISR(void)
{
    uint32_t *pui32Src, *pui32Dest;

    //
    // Copy the data segment initializers from flash to SRAM.
    //
    pui32Src = &_etext;
    for(pui32Dest = &_edata; pui32Dest < &_edata; )
    {
        *pui32Dest++ = *pui32Src++;
    }
}
```

Outline | Disassembly

Enter location here

```
000002fc: nop
000002fe: b.n 0x2fc <IntDefaultHandler>
00000300: nop
00000302: ResetISR:
00000304: ldr r1, [pc, #104] ; (0x36c <ResetISR+108>)
00000306: ldr r3, [pc, #108] ; (0x370 <ResetISR+112>)
00000308: cmp r1, r3
0000030a: push {r4, r5, r6}
0000030c: bcs.n 0x342 <ResetISR+66>
0000030e: adds r2, r3, #3
00000310: adds r4, r1, #4
00000312: subs r5, r2, r4
00000314: ldr r0, [pc, #96] ; (0x374 <ResetISR+116>)
00000316: bic.w r2, r5, #3
00000318: adds r6, r2, #4
0000031a: ldr r5, [r0, #0]
0000031c: movs r3, #4
0000031e: cmp r3, r6
00000320: str r5, [r1, #0]
00000322: ubfx r2, r2, #2, #1
00000324: beq.n 0x342 <ResetISR+66>
00000326: cbz r2, 0x332 <ResetISR+50>
00000328: ldr r2, [r0, #4]
```

Aktueller Befehl

Konfiguration des Target-Systems

- Bei selbst erstellter Toolchain weiterer notwendiger Schritt:
 - Konfiguration des Target-Boards hinsichtlich
 - Memory map und Interrupt-Vektor-Tabelle
 - ggf. Bootloader

Memory Map

- Ein Linker Script beschreibt die Abbildung der Segmente (Memory Map) des Kompilats in den Speicher des Target-Systems
 - für Programmcode in ROM und RAM,
 - für Variablen im RAM,
 - für die Vektor-Interrupt-Tabelle.
- Selbst Erstellen eines Script nur mit genauen Wissen über das Target-System möglich (aus Dokumentation).

Interrupt-Vektor-Tabelle

- Teil der Memory Map
- Beinhaltet die Interrupt-Handler:
 - Einsprungadressen für ausgelöste Interrupts (später mehr).
- Oft detailliert zu initialisieren.

Beispiel: Memory-Map und Interrupt-Vektor-Tabelle

Memory-Map Interrupt-Vektor-Tabelle

Debug - TivaCBLINK/src/startup_gcc.c - Eclipse - /User/matthias/Documents/workspace

Debug Console: OpenOCD [Program] /opt/local/bin/openocd TivaCBLINK Release [GDB Hardware Debugging] TivaCBLINK.elf Thread [1] (Suspended : Step) ResetISR() at startup_gcc.c:252 0x302 gdb

Registers Window:

Name	Value	Description
General Registers		
r0	0	
r1	536870912	
r2	0	
r3	0	
r4	0	

Name : r1
Hex: 0x20000000
Decimal: 536870912
Octal: 04000000000
Binary: 10000000000000000000000000000000
Default: 536870912

Project Explorer: TivaCBLINK Binaries Includes src startup_gcc.c TivaCBLINK.c Debug Release META-INF build.properties mem.ld spec.d

Disassembly Window:

```
000002fc: nop
000002fe: IntDefaultHandler:
000002fe: b.n 0x2fc <IntDefaultHandler>
000002fe: nop
00000300: ResetISR:
00000300: ldr r1, [pc, #104] ; (0x36c <ResetISR+108>)
00000302: ldr r3, [pc, #108] ; (0x370 <ResetISR+112>)
00000304: cmp r1, r3
00000306: push {r4, r5, r6}
00000308: bcs.n 0x342 <ResetISR+66>
0000030a: adds r2, r3, #3
0000030c: adds r4, r1, #4
0000030e: subs r5, r2, r4
00000310: ldr r0, [pc, #96] ; (0x374 <ResetISR+116>)
00000312: bic.w r2, r5, #3
```


Beispiel: Auszug aus Datenblatt mit Memory Map

Table 2-4. Memory Map

Start	End	Description	For details, see page ...
Memory			
0x0000.0000	0x0003.FFFF	On-chip Flash	540
0x0004.0000	0x1FFF.FFFF	Reserved	-
0x2000.0000	0x2000.7FFF	Bit-banded on-chip SRAM	524
0x2000.8000	0x21FF.FFFF	Reserved	-
0x2200.0000	0x220F.FFFF	Bit-band alias of bit-banded on-chip SRAM starting at 0x2000.0000	524
0x2210.0000	0x3FFF.FFFF	Reserved	-
Peripherals			
0x4000.0000	0x4000.0FFF	Watchdog timer 0	775
0x4000.1000	0x4000.1FFF	Watchdog timer 1	775
0x4000.2000	0x4000.3FFF	Reserved	-
0x4000.4000	0x4000.4FFF	GPIO Port A	659
0x4000.5000	0x4000.5FFF	GPIO Port B	659

November 15, 2013

Texas Instruments-Production Data

91

Quelle: Texas Instruments, TivaTM TM4C123GH6PM Microcontroller, <http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf>

Ausschnitt Linker-Script

MEMORY

```
{  
    FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 0x00040000  
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 0x00008000  
}
```

SECTIONS

```
{  
    .text :  
    {  
        _text = .;  
        KEEP(*(.isr_vector))  
    }  
}
```

...

Beispiel: Auszug aus Datenblatt mit Interrupt-Vektor-Tabelle

Table 2-8. Exception Types

Exception Type	Vector Number	Priority ^a	Vector Address or Offset ^b	Activation
-	0	-	0x0000.0000	Stack top is loaded from the first entry of the vector table on reset.
Reset	1	-3 (highest)	0x0000.0004	Asynchronous
Non-Maskable Interrupt (NMI)	2	-2	0x0000.0008	Asynchronous
Hard Fault	3	-1	0x0000.000C	-
Memory Management	4	programmable ^c	0x0000.0010	Synchronous
Bus Fault	5	programmable ^c	0x0000.0014	Synchronous when precise and asynchronous when imprecise
Usage Fault	6	programmable ^c	0x0000.0018	Synchronous
-	7-10	-	-	Reserved
SVCall	11	programmable ^c	0x0000.002C	Synchronous
Debug Monitor	12	programmable ^c	0x0000.0030	Synchronous
-	13	-	-	Reserved

Ausschnitt Interrupt-Vektor-Tabelle

```
__attribute__((section(".isr_vector")))
void (* const g_pfnVectors[])(void) =
{
    (void (*)(void))((uint32_t)pui32Stack + sizeof(pui32Stack),
    // The initial stack pointer
    ResetISR,
    // The reset handler
    NmiSR,
    // The NMI handler
    FaultISR,
    // The hard fault handler
    IntDefaultHandler,
    // The MPU fault handler
    IntDefaultHandler,
    // The bus fault handler
    IntDefaultHandler,
    // The usage fault handler
    ...
}
```


Bootloader

- Auf Target-System installierter Programmcode
- Zur Unterstützung des Überspielen (Flashen) zum Target-System
- Änderung des Bootloaders mittels speziellen Schritten (meist mit Setzen von sogenannten Fuse-Bits bei dem Flashen)

“HELLO WORLD” IN DER
EMBEDDED WELT

“Hello World”

- Eingebettete Systeme haben oft wenig Ein-/Ausgabegeräte
- Erstes Programm ist dann Blinken einer LED (entspricht dem “Hello World” auf Desktop-Systemen)

LED-Blinken

- Genereller Ablauf bei zusammengebauter Hardware:
 - Initialisierung des Systems (eines sogenannten Ports)
 - Schleife mit
 - LED an (Signal “an” an Port)
 - Warten
 - LED aus (Signal “off” an Port)
 - Warten

Launchpad Tiva C Beispiel: main.c

```
#include <stdint.h>
#include "inc/tm4c123gh6pm.h"
int main(void)
{
    // Initialisierung des GPIO Port auf dem Board.
    SYSCTL_RCGC2_R = SYSCTL_RCGC2_GPIOF;

    // Verbrauch von Taktzyklen (Wartet auf erste Init.)
    volatile uint32_t ui32Loop = 0;

    // Initialisierung des GPIO als digitalen Ausgang.
    GPIO_PORTF_DIR_R = 0x08;
    GPIO_PORTF_DEN_R = 0x08;

    // Schleife
    while(1)
    {
        // LED an
        GPIO_PORTF_DATA_R |= 0x08;

        // Warten
        for(ui32Loop = 0; ui32Loop < 320000; ui32Loop++);

        // LED aus
        GPIO_PORTF_DATA_R &= ~(0x08);

        // Warten
        for(ui32Loop = 0; ui32Loop < 320000; ui32Loop++);
    }
}
```


Launchpad Tiva C Beispiel: main.i

...

Hinweise auf Include-Dateien und typedefs

...

```
int main(void)
{
```

```
    (*((volatile uint32_t *)0x400FE108)) = 0x00000020;
```

```
    volatile uint32_t ui32Loop = 0;
```

```
    (*((volatile uint32_t *)0x40025400)) = 0x08;
```

```
    (*((volatile uint32_t *)0x4002551C)) = 0x08;
```

```
    while(1)
```

```
    {
```

```
        (*((volatile uint32_t *)0x400253FC)) |= 0x08;
```

```
        for(ui32Loop = 0; ui32Loop < 320000; ui32Loop++);
```

```
        (*((volatile uint32_t *)0x400253FC)) &= ~(0x08);
```

```
        for(ui32Loop = 0; ui32Loop < 320000; ui32Loop++);
```

```
    }
```

```
}
```


Launchpad Tiva C Beispiel: main.s

...Anweisungen an den Assembler...

main:

.LFB0:

.file 1 "../src/main.c"

.loc 1 44 0

.cfi_startproc

@ args = 0, pretend = 0, frame = 8

@ frame_needed = 0, uses_anonymous_args = 0

@ link register save eliminated.

push{r4, r5, r6, r7}

.LCFI0:

.cfi_def_cfa_offset 16

.cfi_offset 4, -16

.cfi_offset 5, -12

.cfi_offset 6, -8

.cfi_offset 7, -4

.loc 1 53 0

movw r2, #21788

.loc 1 46 0

movw r6, #57608

.loc 1 52 0

mov r5, #21504

.loc 1 44 0

sub sp, sp, #8

.LCFI1:

.cfi_def_cfa_offset 24

.loc 1 46 0

movt r6, 16399

.loc 1 52 0

movt r5, 16386

...weitere Assemblerbefehle...

Die weiteren Dateien
main.o, main.elf,
main.bin enthalten
Binärcode.

Launchpad Tiva C Beispiel: Disassembly

... vorher weiterer Code

// Schleife

while(1)

{

// LED an

GPIO_PORTF_DATA_R |= 0x08;

2a6: 680e ldr r6, [r1, #0]

2a8: f046 0508 orr.w r5, r6, #8

2ac: 600d str r5, [r1, #0]

// Warten

for(ui32Loop = 0; ui32Loop < 320000; ui32Loop++);

2ae: 9001 str r0, [sp, #4]

2b0: 9a01 ldr r2, [sp, #4]

2b2: 429a cmp r2, r3

2b4: d805 bhi.n 2c2 <main+0x56>

2b6: 9c01 ldr r4, [sp, #4]

2b8: 1c67 adds r7, r4, #1

2ba: 9701 str r7, [sp, #4]

2bc: 9e01 ldr r6, [sp, #4]

2be: 429e cmp r6, r3

2c0: d9f9 bls.n 2b6 <main+0x4a>

// LED aus

GPIO_PORTF_DATA_R &= ~(0x08);

2c2: 680d ldr r5, [r1, #0]

2c4: f025 0208 bic.w r2, r5, #8

2c8: 600a str r2, [r1, #0]

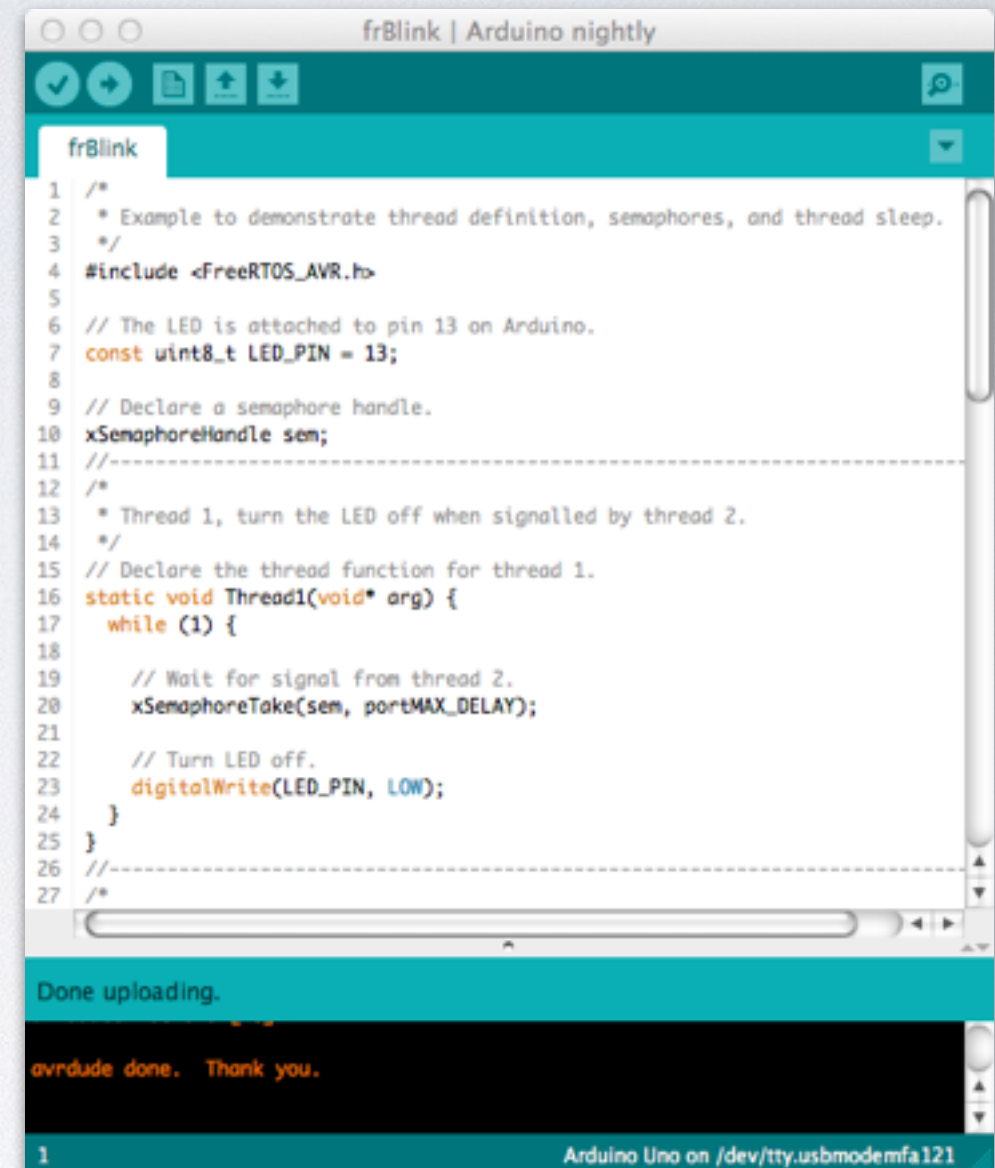
... folgend weiterer Code

ARDUINO

ENTWICKLUNGSWERKZEUGE

Arduino Entwicklungswerkzeuge

- Arduino mit Atmel Chips IDEs
 - Eigene Arduino IDE
 - Plugin für Eclipse
 - Visual Micro für MS Visual Studio
 - Atmel Studio
 - und weitere

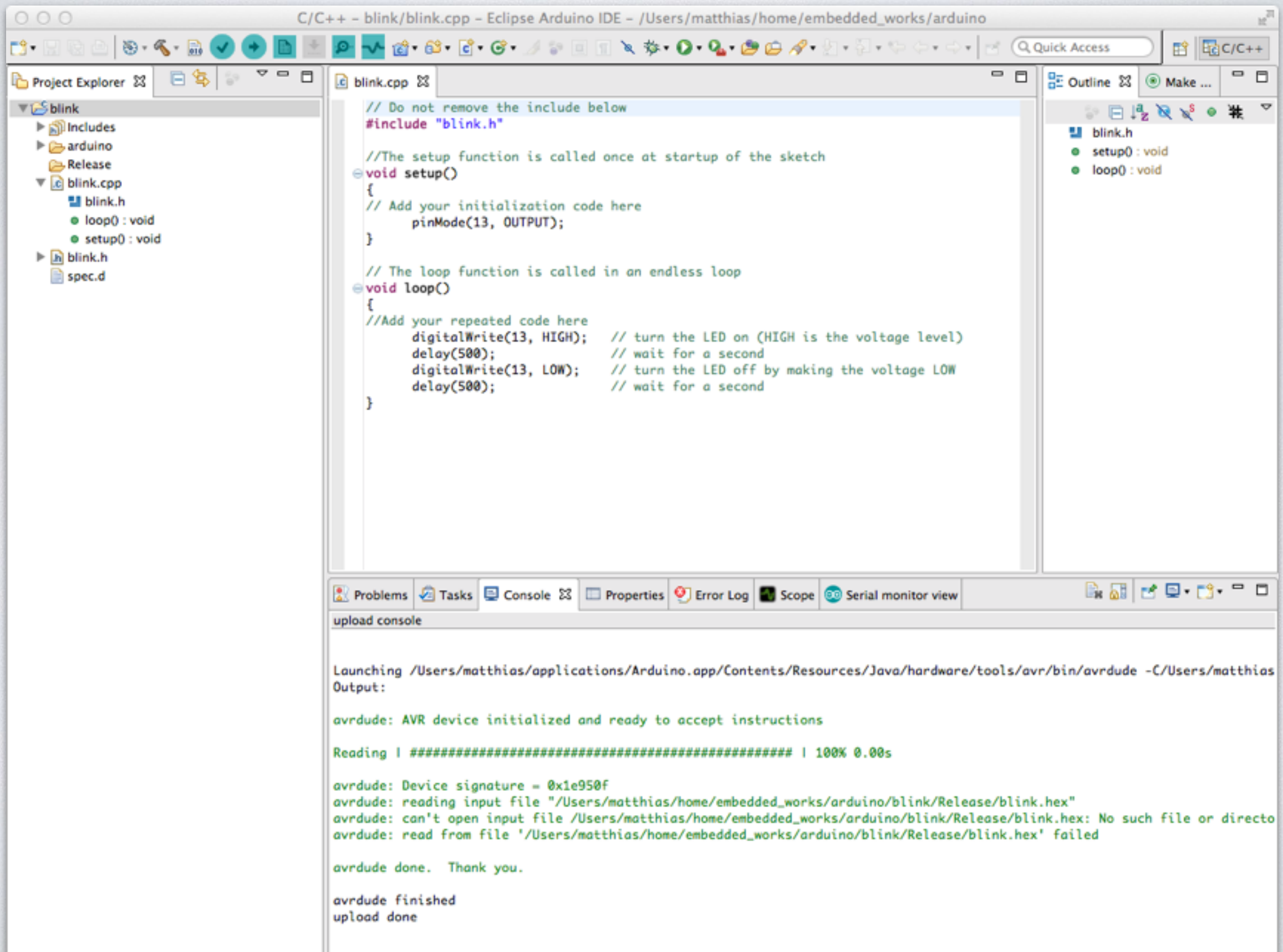
A screenshot of the Arduino IDE interface. The title bar reads "frBlink | Arduino nightly". The main editor window shows a C++ program named "frBlink" with the following code:

```
1  /*
2   * Example to demonstrate thread definition, semaphores, and thread sleep.
3   */
4  #include <FreeRTOS_AVR.h>
5
6  // The LED is attached to pin 13 on Arduino.
7  const uint8_t LED_PIN = 13;
8
9  // Declare a semaphore handle.
10 xSemaphoreHandle sem;
11 //-----
12 /*
13  * Thread 1, turn the LED off when signalled by thread 2.
14  */
15 // Declare the thread function for thread 1.
16 static void Thread1(void* arg) {
17     while (1) {
18
19         // Wait for signal from thread 2.
20         xSemaphoreTake(sem, portMAX_DELAY);
21
22         // Turn LED off.
23         digitalWrite(LED_PIN, LOW);
24     }
25 }
26 //-----
27 /*
```

The bottom status bar shows "Done uploading." and "avrduide done. Thank you." The bottom right corner indicates "Arduino Uno on /dev/tty.usbmodemfa121".

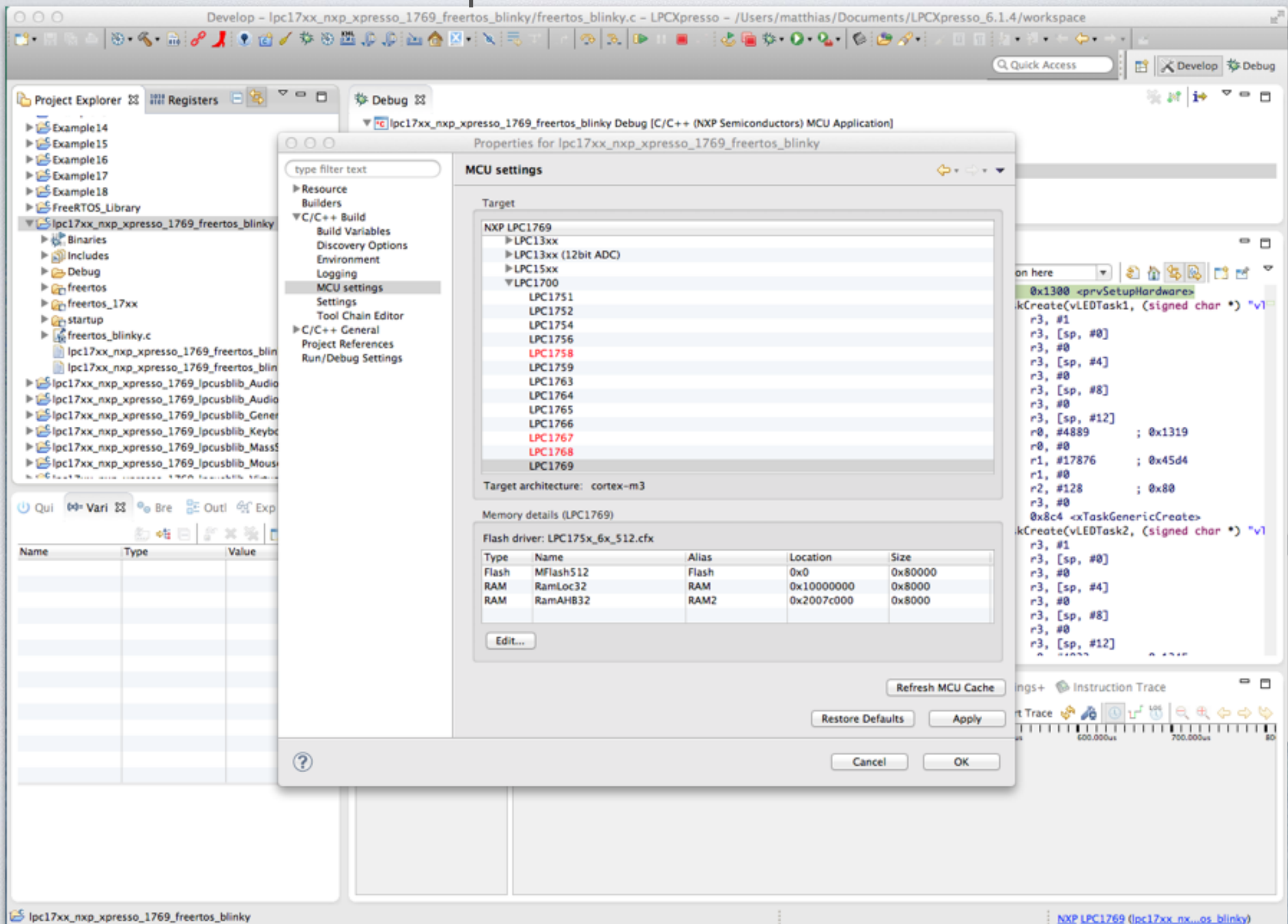
Standard Arduino IDE

Eclipse mit Arduino Plugin



KOMMERZIELLE ENTWICKLUNGSWERKZEUGE

LPCXpresso von NXP



Toolchain: Anbieter

- Konfigurationen werden von einem kommerziellen Werkzeug normalerweise übernommen (z.B. Wizard).
- Toolchains von Embedded-Hardware Herstellern nur für deren Hardware.
- Toolchains von “unabhängigen” Anbieter für Vielzahl von Hardware.
- Kommerzielle Lizenz (ein Entwickler) ab ca. 1500 Euro.
- Zur Evaluierung oft eingeschränkte Lizenzen erhältlich.

Beispiele einiger Anbieter

- Preise für einen kommerziellen Entwickler (nicht für Hochschulen), Stand 03.2014:
 - Keil MDK-ARM, ca. 3500-4000€,
 - IAR Embedded Workbench (ARM Baseline): ca. 2500-2800€
 - Rowley Crossworks (z.B. für ARM), 1200€
 - TI Code Composer (nur TI Boards), ca. 600-650€
 - Green Hills MULTI, ca. 6000 US\$

Zusammenfassung

- Im Kontext der Programmierung eingebetteter Systeme
 - Elemente einer Toolchain
 - “Hello World“-Äquivalent

Literatur / Quellen

- Arduino, URL: <http://www.arduino.cc>
- Eclipse, URL: <http://www.eclipse.org>
- Texas Instruments, URL: <http://www.ti.com>
- **Stand aller Internetquellen: 01.03.2014**