

# **Keil $\mu$ Vision 5.23 Tutorial**

Andreas Mieke  
andreas@1750studios.com

Hollabrunn, 10. September 2017

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>3</b>
1.1	Warum der Umstieg zu $\mu$ Vision 5? . . . . .	3
1.2	Mindestsystemanforderungen . . . . .	3
<b>2</b>	<b>Das erste <math>\mu</math>Vision 5 Projekt</b>	<b>4</b>
2.1	Die Installation . . . . .	4
2.2	Der Pack Installer . . . . .	11
2.3	Installation des HTBL Packs . . . . .	15
2.4	Die Projekterstellung . . . . .	17
<b>3</b>	<b>Debugging</b>	<b>28</b>
3.1	Keil ULINK2/ME Debugger . . . . .	35
3.2	ST-Link Debugger . . . . .	36
<b>4</b>	<b>Assembler-Programmierung</b>	<b>37</b>
<b>5</b>	<b>C-Programmierung</b>	<b>40</b>

# 1 Einführung

## 1.1 Warum der Umstieg zu $\mu$ Vision 5?

In der HTBL Hollabrunn wurde in den letzten Jahren laufend die Version 4 der Programmierungsumgebung  $\mu$ Vision verwendet, ein Umstieg auf die neuere Version 5 war weder nötig noch wirklich sinnvoll.


Allerdings hat die  $\mu$ Vision in der Version 4 einen großen Nachteil, welcher die Verwendbarkeit in der Zukunft stark einschränkt. Denn ein kompilieren von Programmen für Cortex M4 oder höher ist bei dieser Version nicht möglich, und Version 5 wird zur zwingenden Voraussetzung. Da die Entwicklung weiter voran schreitet, ist es für die HTBL Hollabrunn nicht mehr praktikabel die veraltete Version 4 einzusetzen.

Dieses Dokument wird kurz auf die Mindestanforderungen der neuen Software, und des weiteren auf die Inbetriebnahme mittels einfachem Beispielprogramm demonstrieren. Alte  $\mu$ Vision 4 Projekte können auf diese Weise in die neue Umgebung übertragen werden.

## 1.2 Mindestsystemanforderungen

	Minimum	Empfohlen
Prozessor	1 GHz (32/64 bit)	2 GHz (64 bit) oder mehr
RAM	1 GB	4 GB oder mehr
Festplattenspeicher	2 GB	5 GB oder mehr
Internet		2 Mb/s oder mehr (für Pack Installer)

Alle Windows Versionen ab Windows Vista (32/64 bit) werden unterstützt.

 Achtung: Im Gegensatz zu  $\mu$ Vision in Version 4, wird von dieser Version das Betriebssystem Microsoft Windows XP nicht mehr unterstützt!

## 2 Das erste $\mu$ Vision 5 Projekt

### 2.1 Die Installation

Bevor mit der eigentlichen Installation der IDE<sup>1</sup> begonnen werden kann, muss diese von der offiziellen Keil Webseite heruntergeladen werden. Dies kann unter diesem Link getan werden: <https://www.keil.com/demo/eval/arm.htm>

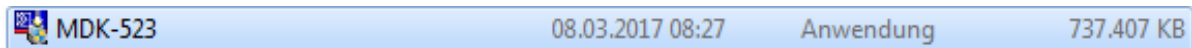


Abbildung 1: Der heruntergeladene  $\mu$ Vision 5 Installer

Nachdem der Installer heruntergeladen wurde, sollte eine ausführbare Datei wie in Abbildung 1 zu sehen ist vorhanden sein, eventuell sollte die Größe dieser Datei überprüft werden, um auszuschließen, dass es beim Download zu einem Fehler kam.

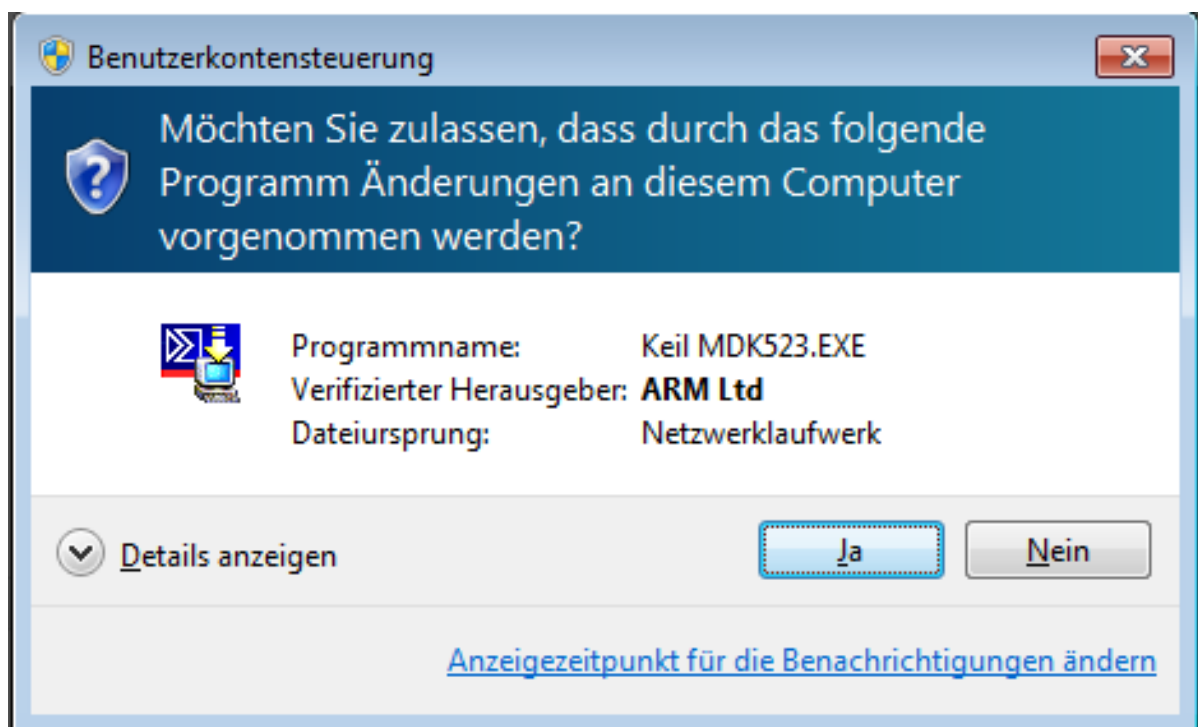


Abbildung 2: Dialog von Windows User Account Control

<sup>1</sup>Integrated Development Environment

Nachdem der Installer erfolgreich heruntergeladen wurde, muss dieser mit einem Doppelklick gestartet werden. Eventuell zeigt Windows einen Bestätigungsdialog (Abbildung 2 auf der vorherigen Seite) an, dieser ist mit einem Klick auf den Button **Ja** zu bestätigen.

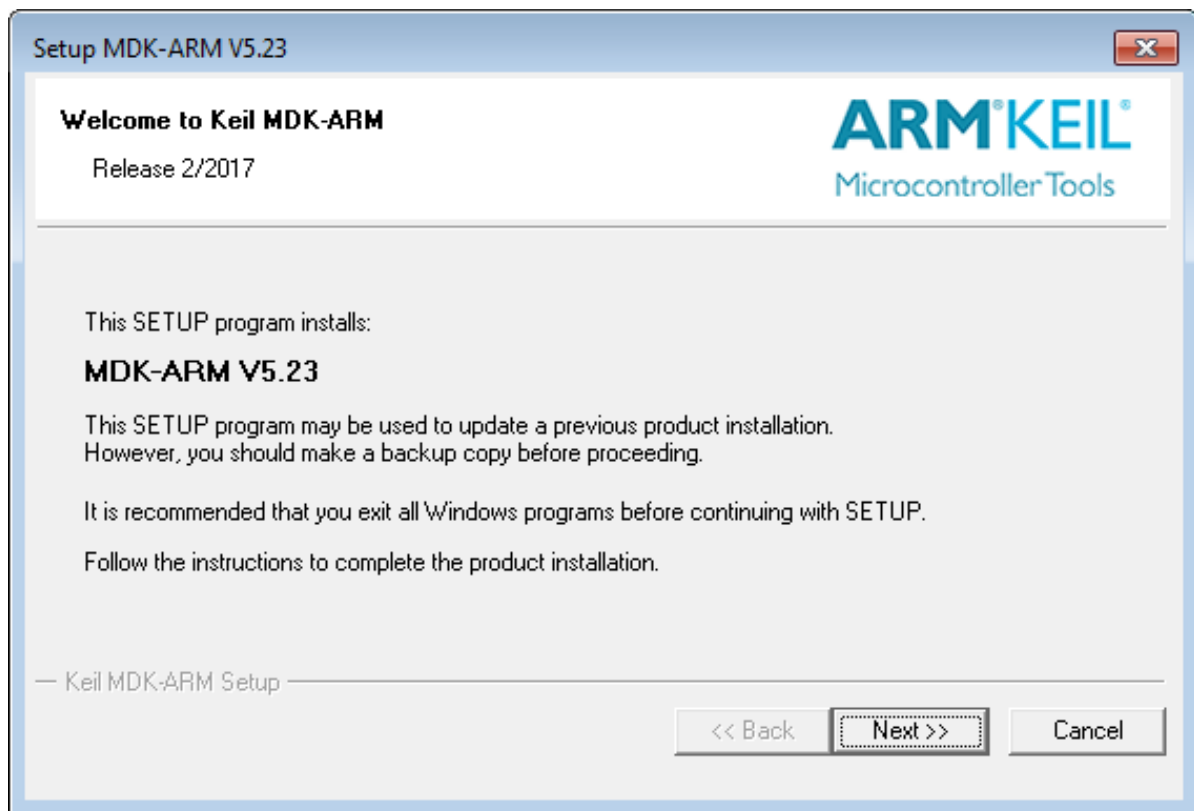


Abbildung 3: Begrüßungsbildschirm des Installers

Wenn der Dialog bestätigt wurde, startet der eigentliche Installationsprozess. Im Begrüßungsbildschirm (Abbildung 3) wird nochmals erläutert welche Version der Software zur Zeit installiert wird (in diesem Fall  $\mu$ Vision in Version **5.23**). Die Richtigkeit dieser Angaben wird mit einem Klick auf **Next** bestätigt.

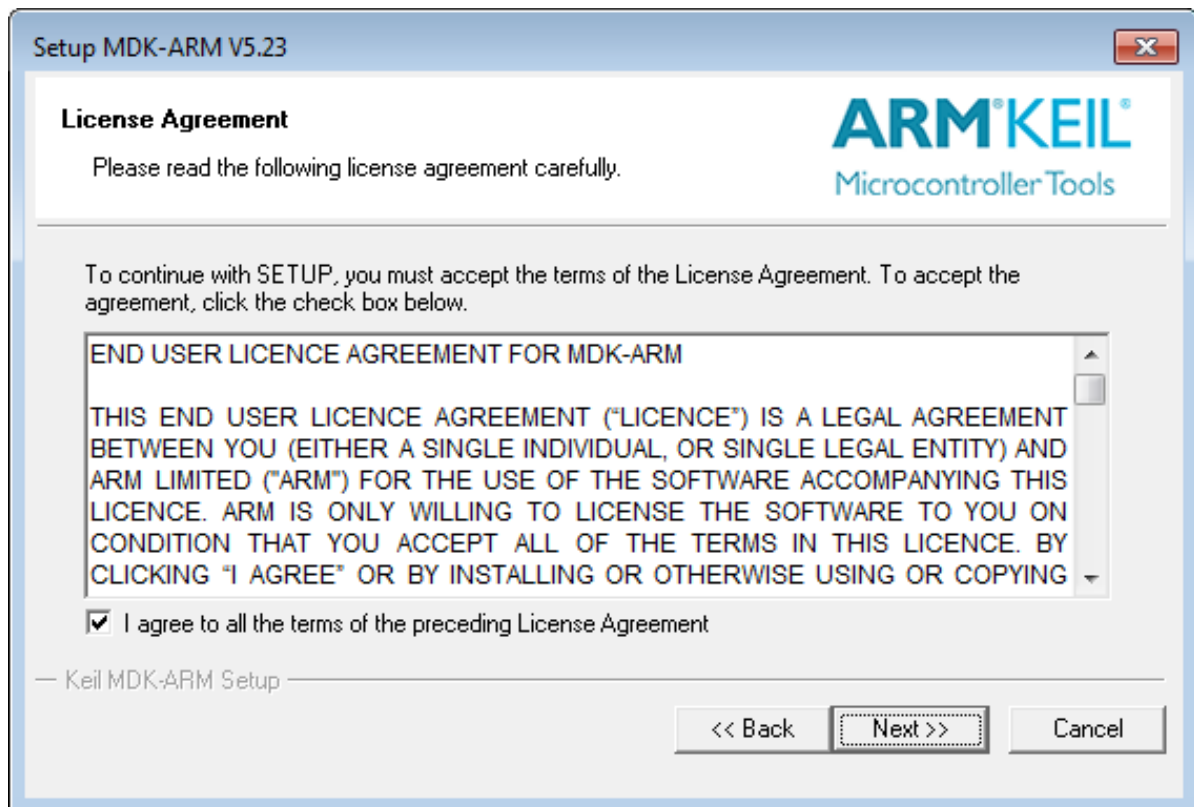


Abbildung 4: Lizenzbedingungen

Danach müssen die Lizenzbedingungen akzeptiert werden (Abbildung 4), hierzu muss der Hacken in der Checkbox gesetzt werden und wieder mit einem Klick auf **Next** bestätigt werden.

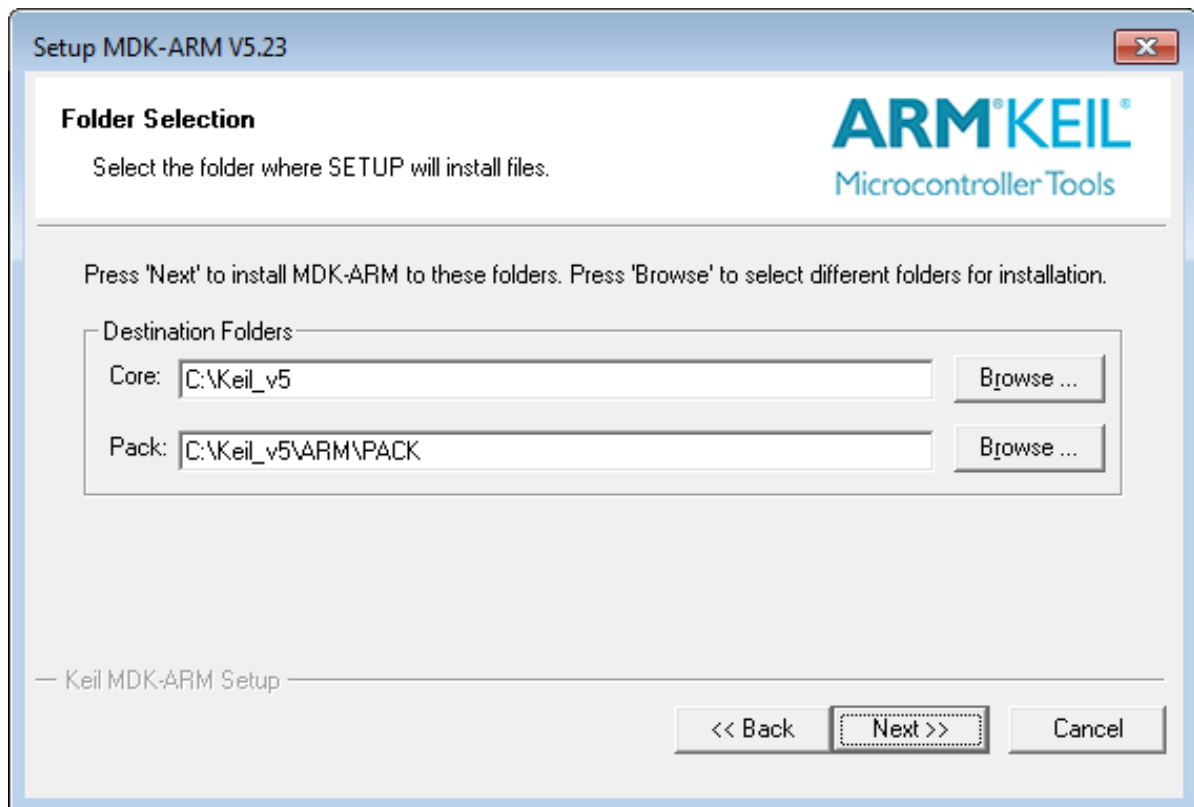


Abbildung 5: Auswahl der Installationspfade

Im nächsten Bildschirm (Abbildung 5) wird der Installationsort für den Compiler und die IDE, sowie der Pfad für die Installation von Packs (Abbildung 12 auf Seite 13) abgefragt. Grundsätzlich können beide Felder auf beliebige Pfade gesetzt werden, aufgrund der Kompatibilität und der vereinfachten Fehlersuche wird aber empfohlen den Standard beizubehalten.

**Setup MDK-ARM V5.23**

**Customer Information**

Please enter your information.

ARM<sup>®</sup> KEIL<sup>®</sup>  
Microcontroller Tools

Please enter your name, the name of the company for whom you work and your E-mail address.

First Name:

Last Name:

Company Name:

E-mail:

— Keil MDK-ARM Setup —

<< Back   **Next >>**   Cancel

Abbildung 6: Eingabe der Benutzerdaten

Danach werden Daten zum Benutzer abgefragt (Abbildung 6). Diese Felder können entweder mit erfundenen Daten, oder – wenn man später ggf. eine Lizenz hinzufügen will – mit den realen Daten des Benutzers gefüllt werden.



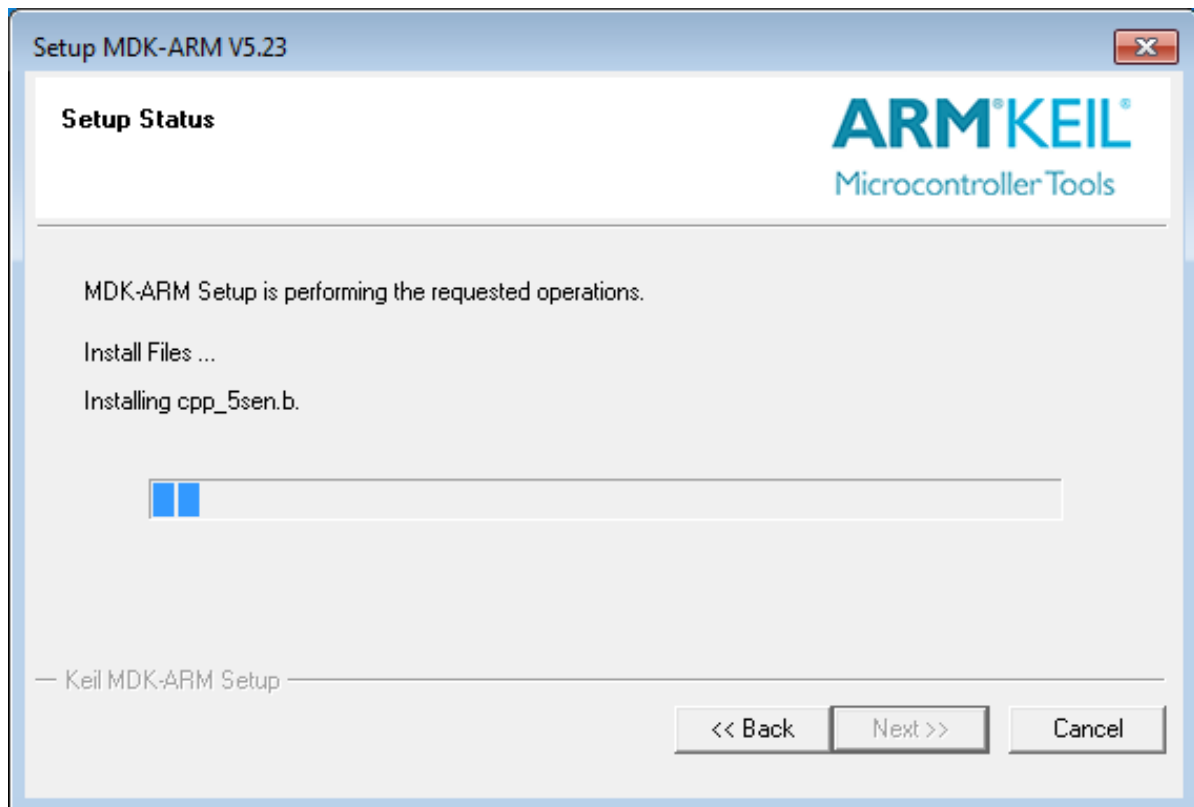


Abbildung 7: Installationsfortschritt

Jetzt wird mit der eigentlichen Installation der Dateien begonnen. Der Fortschritt wird in einem eigenen Bildschirm (Abbildung 7) angezeigt. Nun muss gewartet werden, bis der Balken komplett durchgelaufen ist.

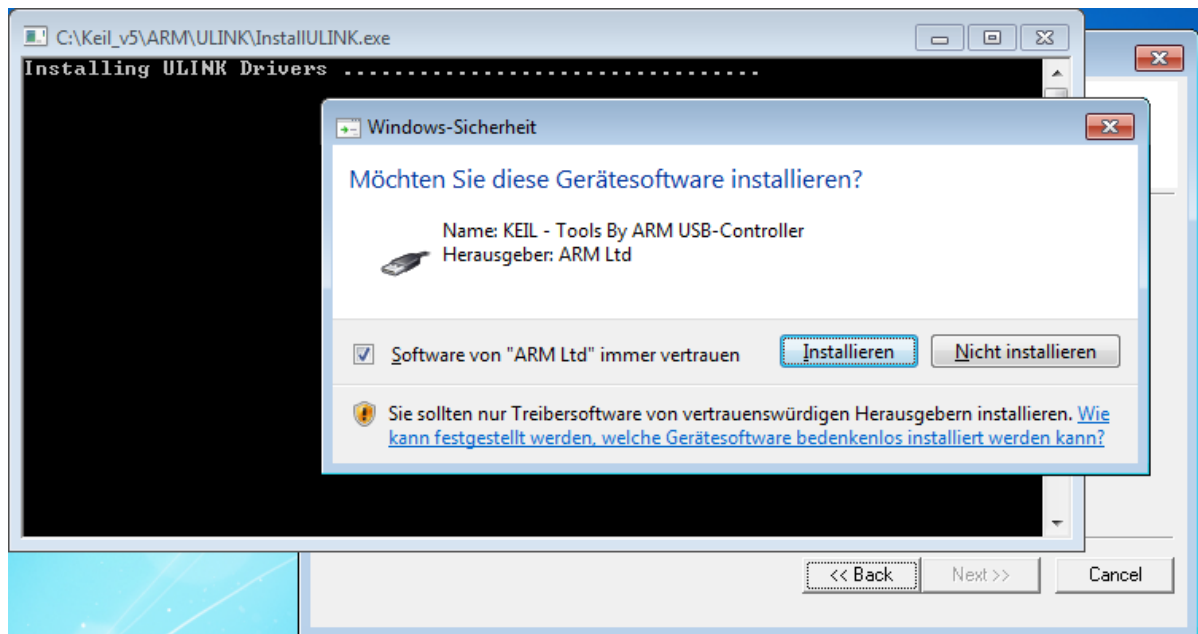


Abbildung 8: Warnung der Windows-Sicherheit

Je nach Windows Version kann ein Konsolenfenster auf gehen, welches einfach ignoriert werden kann. Gegebenenfalls wird auch eine Warnung der Windows-Sicherheit bezüglich der Installation von Treibern angezeigt (Abbildung 8). Diese ist mit einem Klick auf **Installieren** zu bestätigen.

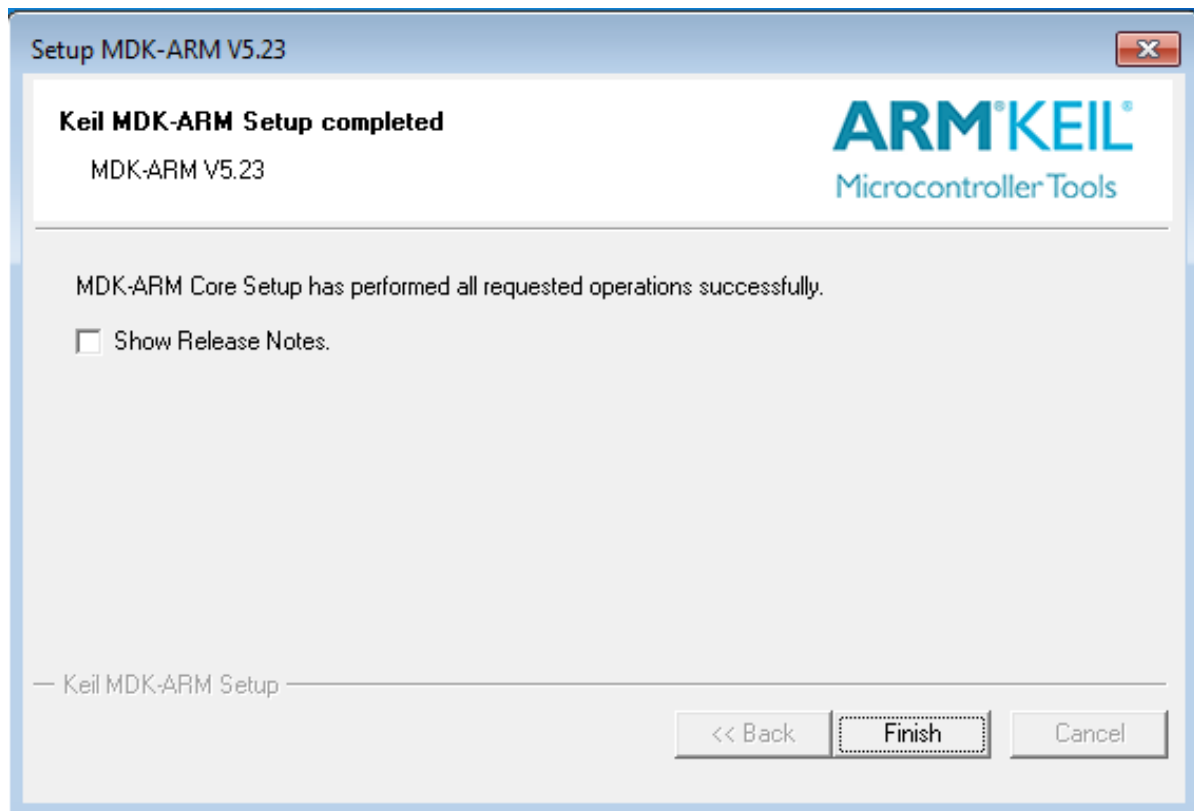


Abbildung 9: Installation erfolgreich

Am letzten Bildschirm (Abbildung 9) wird nachgefragt ob die Release Notes angezeigt werden sollen, dies ist nicht nötig und somit sollte in der Checkbox auch kein Hacken sein. Mit einem klick auf den **Finish** Button wird die Installation beendet. Ein Neustart des Rechners sollte nicht nötig sein.

## 2.2 Der Pack Installer

Nachdem die Installation erfolgreich beendet wurde, startet der **Pack Installer** der  $\mu$ Vision. Dieses Programm verwaltet alle CMSIS<sup>2</sup>-Pakete welche im laufe der Verwendung der IDE heruntergeladen und installiert werden. Der Installer hat im groben zwei Spalten, auf der linken Seite kann man einen Prozessor heraus suchen, welchen man verwenden will, und auf der rechten Seite werden dann die für diesen Prozessor verfügbaren Pakete angezeigt.

---

<sup>2</sup>Cortex Microcontroller Software Interface Standard

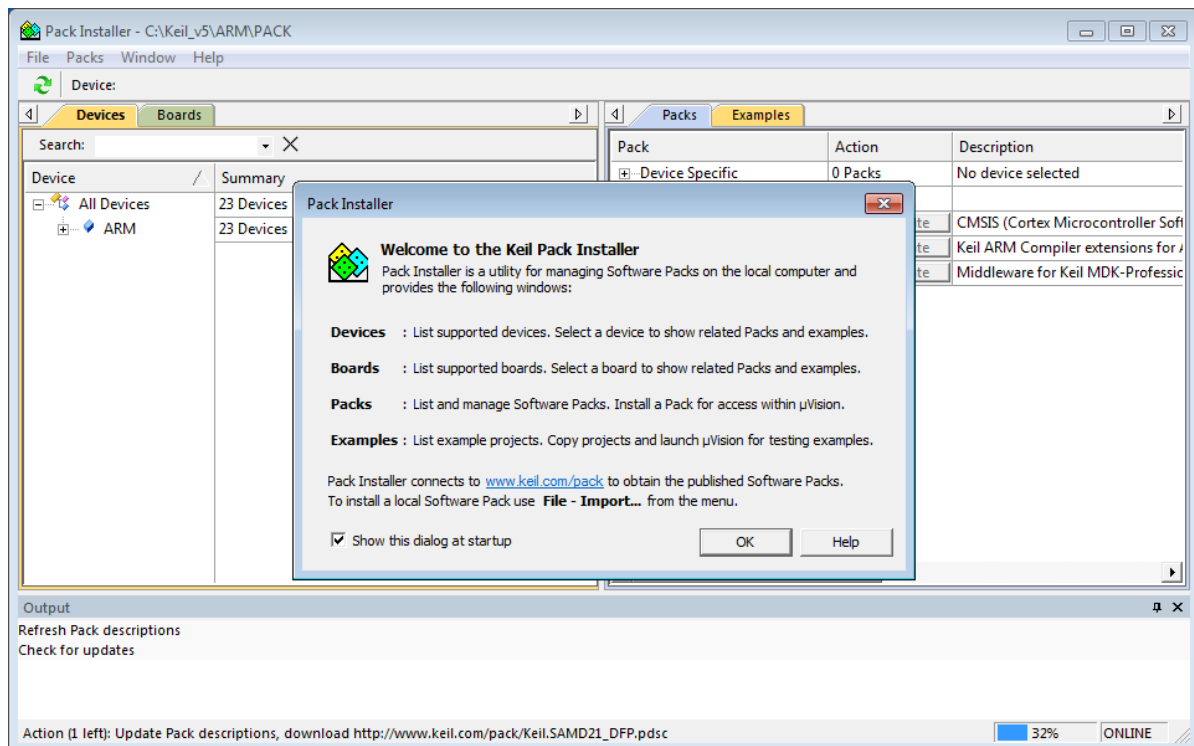


Abbildung 10: Der Pack Installer nach dem ersten Start

Beim ersten Start des Pack Installers, in weiterer Folge nur Installer genannt, wird eine Willkommensnachricht, welche diesen kurz beschreibt, angezeigt (Abbildung 10). Diese kann entweder durch deaktivieren der entsprechenden Checkbox für immer versteckt, oder mittels Button geschlossen werden.

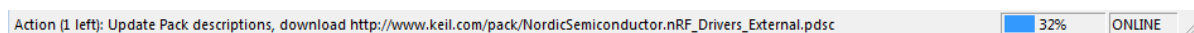


Abbildung 11: Downloadfortschritt

Beim ersten Start sind nur die direkt von ARM ausgelieferten Prozessorkerne im Installer verfügbar (Abbildung 10), allerdings wird gleichzeitig auch ein Updateprozess gestartet, welcher alle nötigen Paket-Infos herunterlädt und gegebenenfalls bereits installierte Pakete updated. Um den Installer richtig verwenden zu können, müssen wir dieses erste Update abwarten. In der Statusleiste des Programms (Abbildung 11) sieht man den entsprechenden Fortschritt. Wenn dieser auf 100% steigt, beziehungsweise der Text komplett verschwindet ist das Update beendet und wir können fortfahren.

☞ Zu beachten: Es kann vorkommen, dass die Fortschrittsanzeige beim installieren der einzelnen Pakete kurz verschwindet, es sollte also zur Sicherheit einige Sekunden gewartet werden, wenn der Text verschwindet um sicher zu gehen, dass das Update auch wirklich fertig eingespielt wurde.

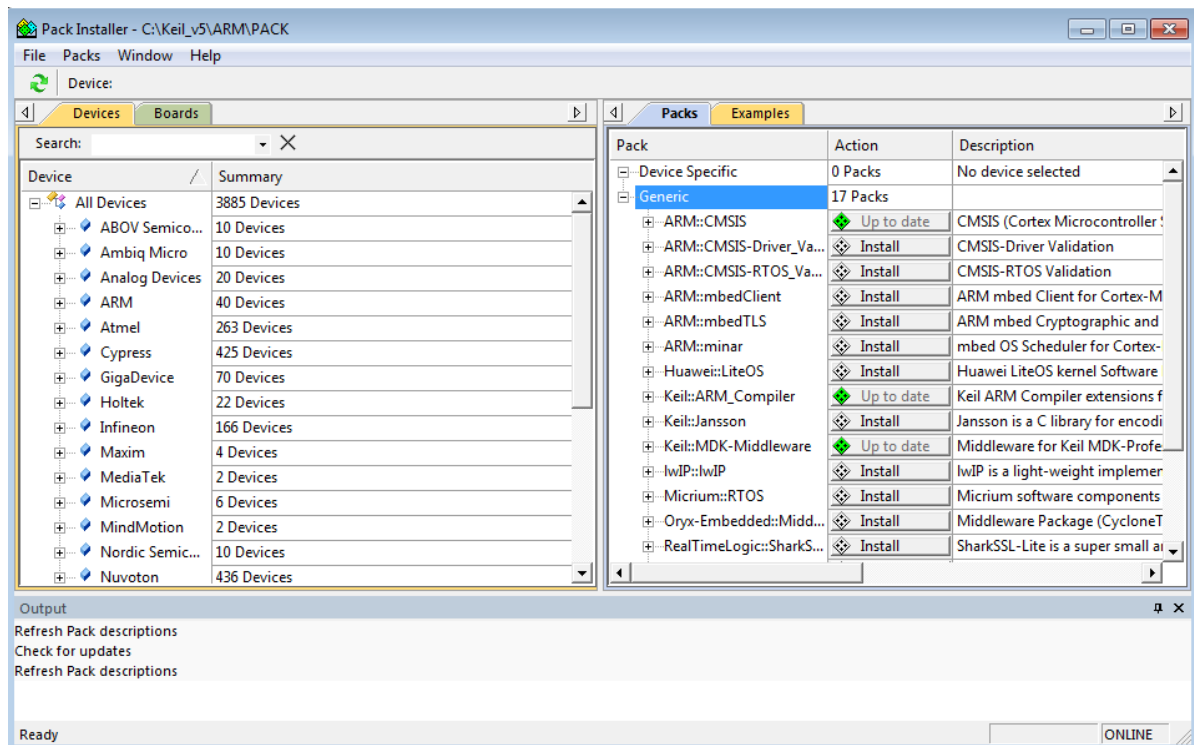


Abbildung 12: Verfügbare Pakete

Nachdem die Updates erfolgreich eingespielt wurden sollte sich die gerade noch leere Liste (Abbildung 12) mit knapp 4000 verfügbaren Prozessoren gefüllt haben.



Abbildung 13: Prozessor des Minimalsystems

Danach muss der passende Prozessor ausgewählt werden (Abbildung 13). Für den Schulgebrauch ist dies zur Zeit der **STM32F103RB**<sup>3</sup>. Man kann den richtigen Prozessor

<sup>3</sup>Cortex M3 Prozessor der Firma STMicroelectronics

entweder über die Liste auswählen, oder einfach das Suchfeld oben links verwenden um direkt den Richtigen angezeigt zu bekommen.




[-] Device Specific	3 Packs	STM32F103RB selected
+ Hitec::CMSIS_RTOS_Tutorial	 Install	An Introduction to using CMSIS RTOS for Cortex-M Microcontrollers
+ Keil::STM32F1xx_DFP	 Install	STMicroelectronics STM32F1 Series Device Support, Drivers and Examples
+ Keil::STM32NUCLEO_BSP	 Install	STMicroelectronics Nucleo Boards Support and Examples

Abbildung 14: Verfügbare Pakete

Auf der rechten Seite (Abbildung 14) scheinen, sobald der richtige Prozessor ausgewählt ist, die für diesen Prozessor verfügbaren CMSIS Pakete auf, diese können nun mittels Klick auf den entsprechenden Button installiert und danach in der IDE verwendet werden. Für den Schulgebrauch ist das Paket **Keil::STM32F1xx\_DFP** nötig und ausreichend. Dieses beinhaltet alle verwendeten Libraries und auch Beispielprogramme. Während der Installation des Pakets ist wieder auf die Statusleiste und den Fortschritt in dieser zu achten, wenn alles erfolgreich installiert wurde, sollte unser verwendeter Prozessor auf der linken Übersichtsseite grün hinterlegt sein, siehe dazu Abbildung 15 auf der nächsten Seite. Des weiteren sollten sämtliche Packs, bei welchen **Update** steht geupdated werden.

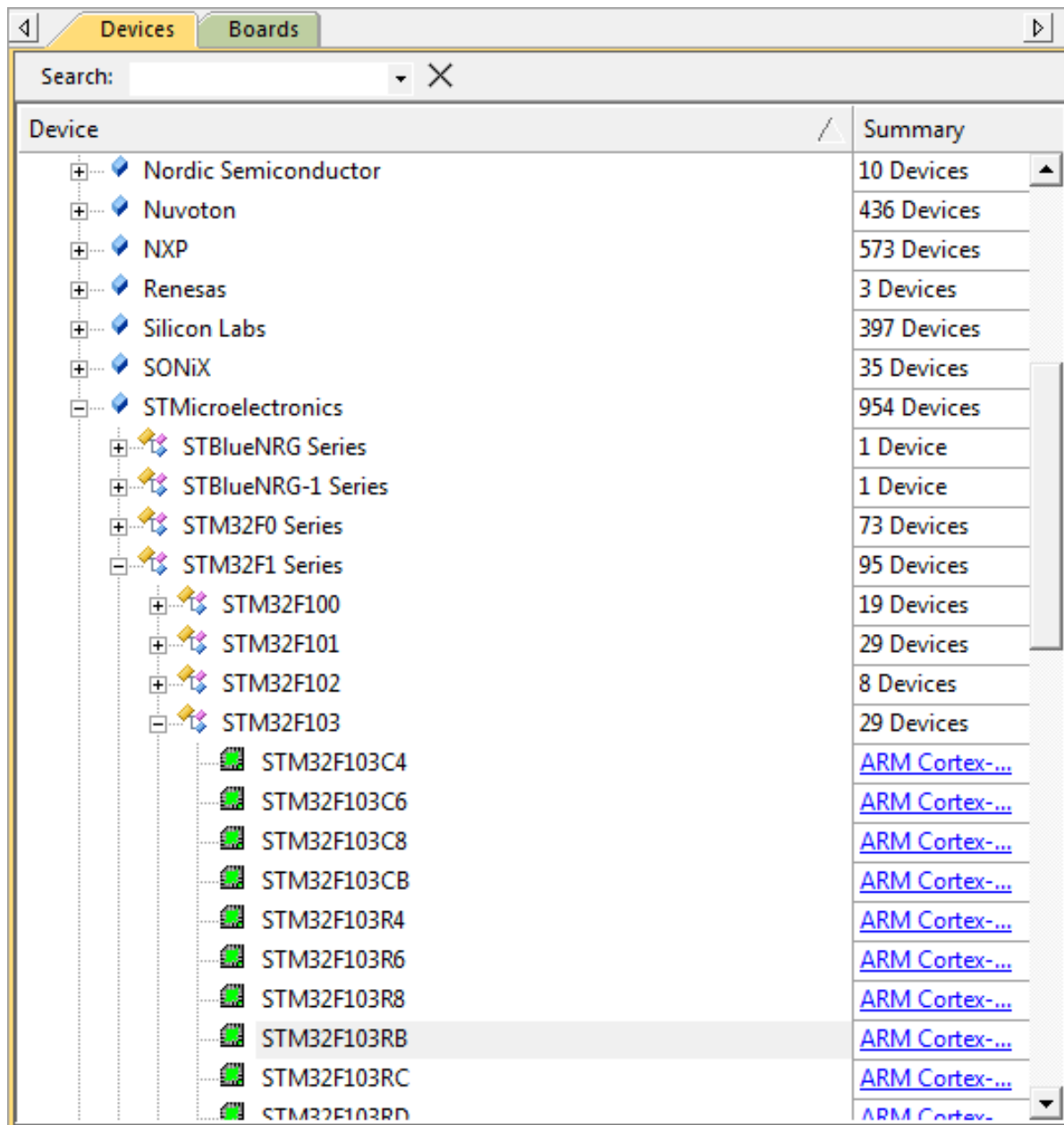


Abbildung 15: Erfolgreich installierter Prozessor

## 2.3 Installation des HTBL Packs

Um die spezifischen Libraries und Header Files der HTBL Hollabrunn einfach zur Verfügung zu stellen, gibt es für eben diese ein eigenes CMSIS-Pack. Dieses wird aber anders als die anderen Packs (noch) nicht über das Internet vertrieben, sondern vom Lehrer auf

einem Medium wie USB-Stick oder CD ausgeteilt. Dementsprechend muss dieses Pack manuell in den Pack-Manager hinzugefügt werden.

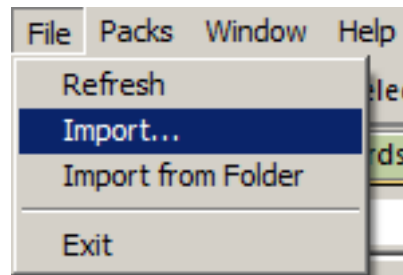


Abbildung 16: Menüpunkt zum manuellen Import von Packs

Hierzu muss auf **File**, und dann auf **Import...** geklickt werden, wie in Abbildung 16 dargestellt. Danach öffnet sich ein Explorer Fenster, in welchem man die entsprechende .pack Datei auswählen muss. Wenn dies geschehen ist wird die Datei eingelesen und automatisch installiert. Sollte aus irgendeinem Grund die Zielfeile schon existieren, so ist das überschreiben mittels Klick auf **Ja** im entsprechenden Dialogfenster zu bestätigen. Nach erfolgter Installation sollte das Übersichtsfenster wie in Abbildung 17 aussehen.

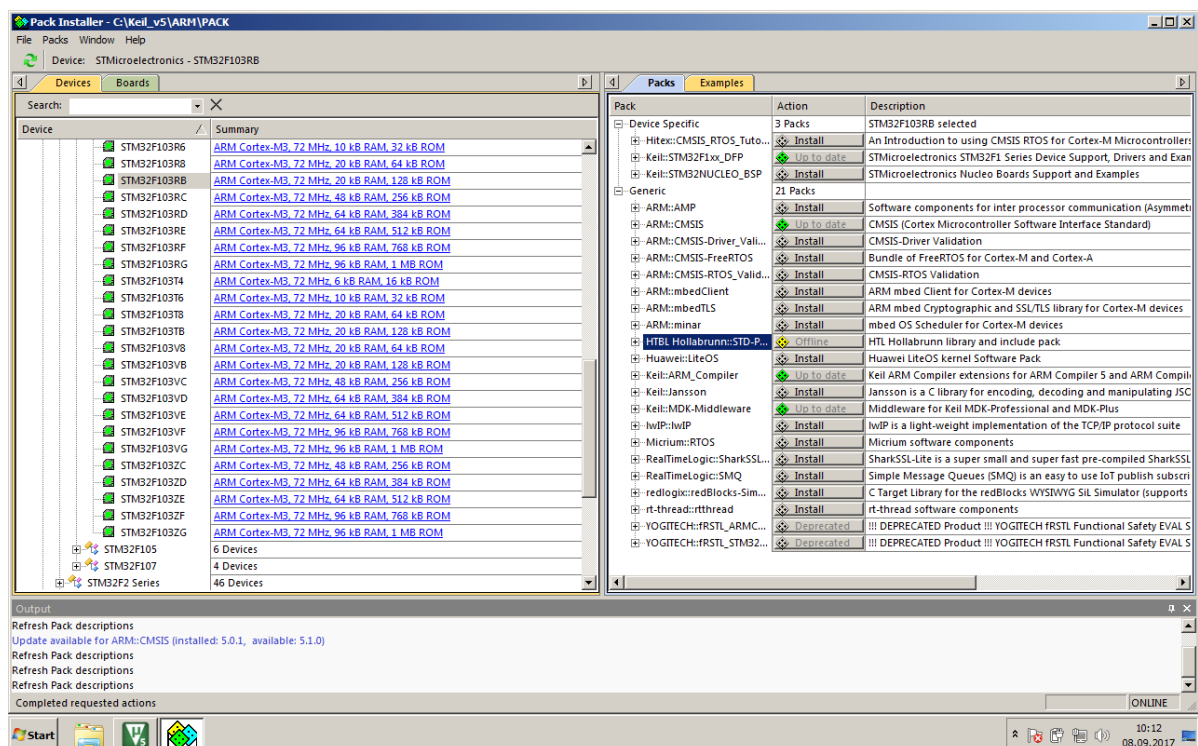


Abbildung 17: Hauptfenster nach erfolgreicher Installation des HTBL Packs



## 2.4 Die Projekterstellung

Als nächstes muss ein  $\mu$ Vision Projekt erstellt werden, die Projekte dienen zur Organisation der Source-Files und der verwendeten Bibliotheken, sowohl CMSIS als auch eigene Bibliotheken. Das Projekt kann mittels grafischem Interface einfach konfiguriert und mit CMSIS Libraries versehen werden.

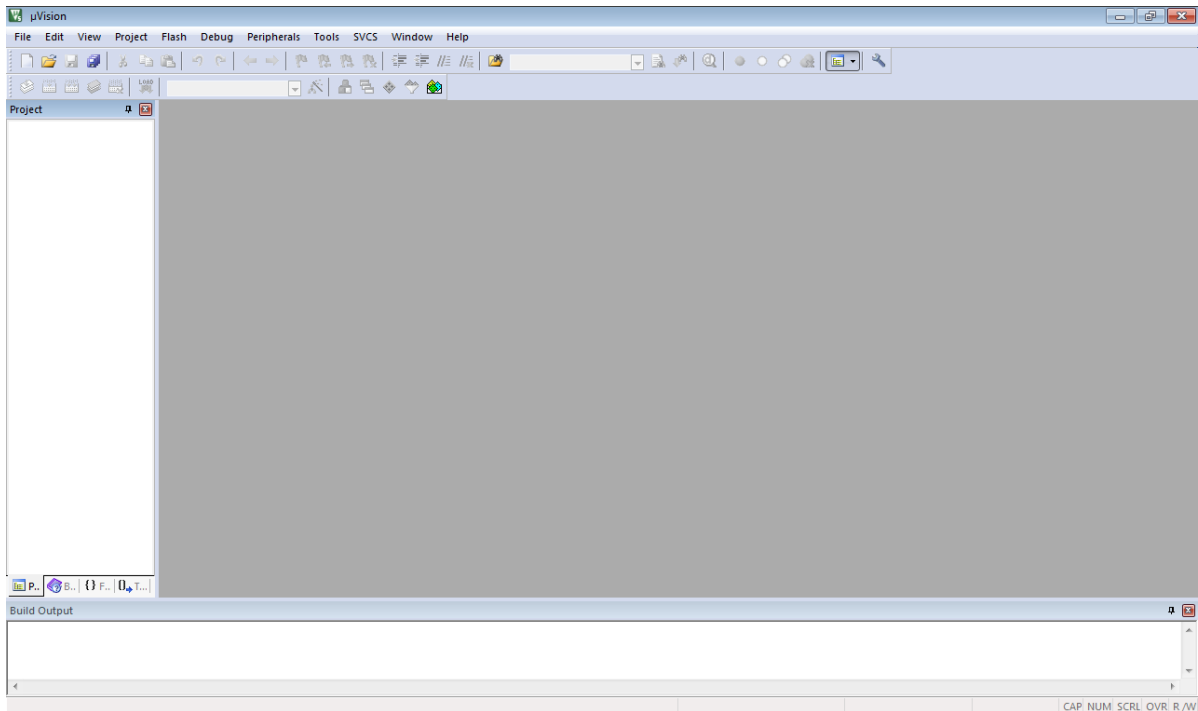


Abbildung 18: Hauptfenster der IDE

Nach dem Start der IDE ist zunächst ein leeres Fenster zu sehen (Abbildung 18). Dieses besteht aus einer Menüleiste ganz oben und direkt darunter die Schnellzugriffsschaltflächen. Auf der linken Seite ist ein noch leerer Projektbaum zu finden und rechts im großen grauen Feld der eigentliche Texteditor für die Source-Files. Darunter befindet sich noch ein Fenster für Log und Compiler Ausgaben, unter diesem die Statusleiste.

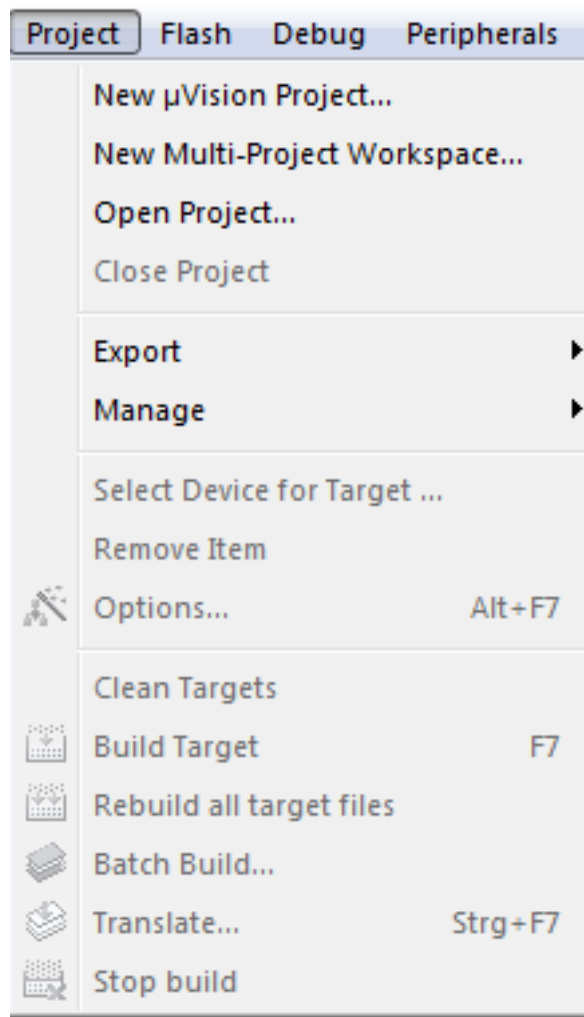


Abbildung 19: Projekt-Menü

Mit einem Klick auf den entsprechenden Menüpunkt (Abbildung 19) kann ein neues  $\mu$ Vision Projekt erstellt werden.

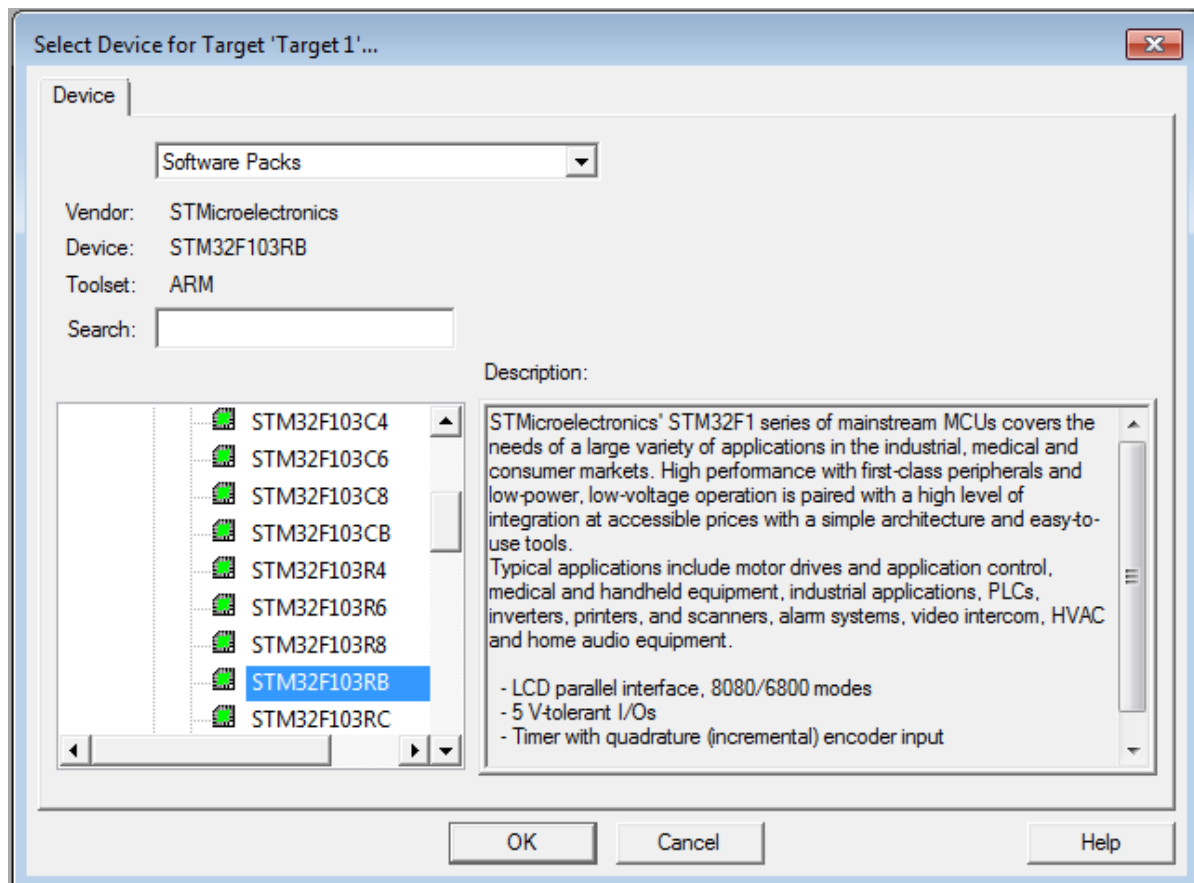


Abbildung 20: Prozessorauswahldialog

Im nächsten Bildschirm (Abbildung 20) wird abgefragt welcher Prozessor verwendet werden soll, in unserem Fall ist dies wieder der **STM32F103RB**. Man kann den richtigen Prozessor entweder in der Liste heraus suchen, oder den Namen direkt in das Suchfeld eingeben. Auf der rechten Seite wird ein Beschreibungstext mit den Features des gewählten Prozessors angezeigt.

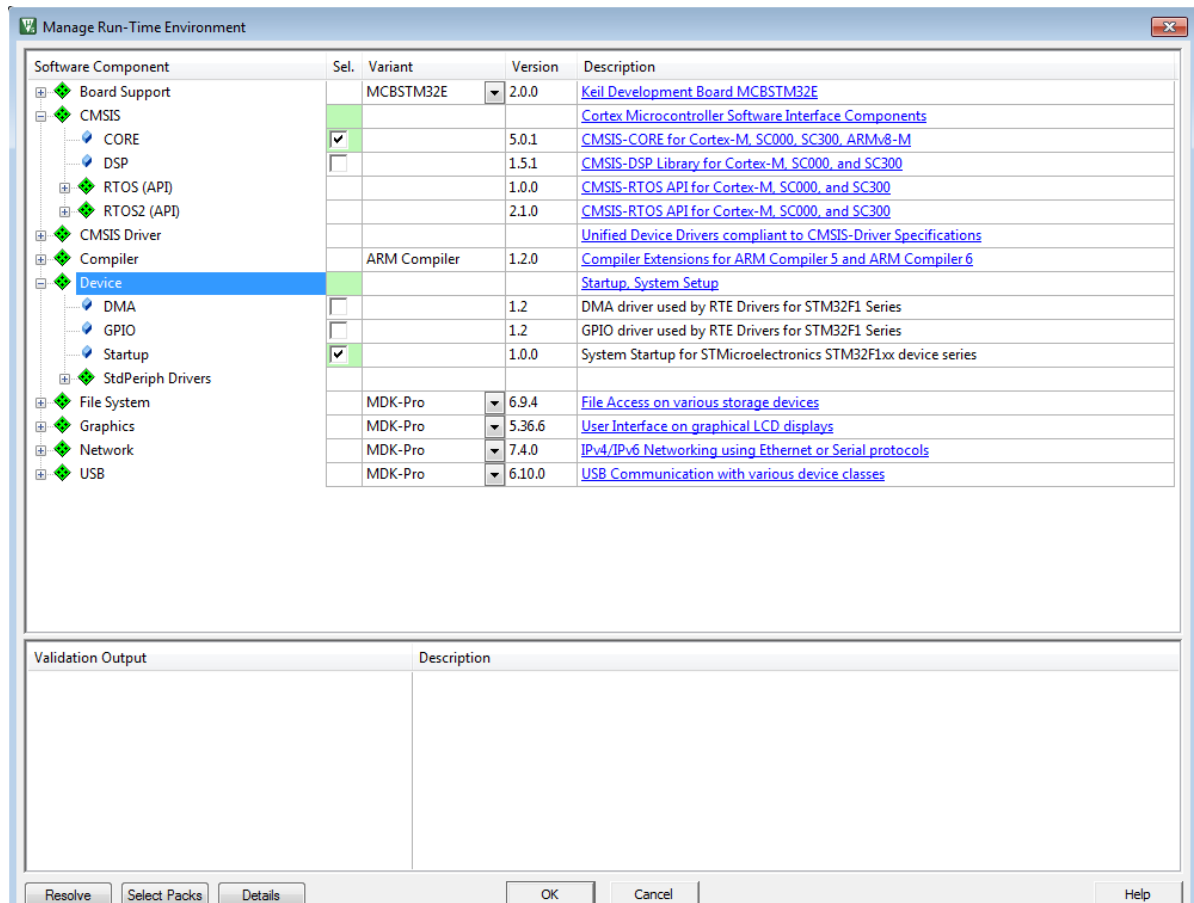



Abbildung 21: Laufzeitumgebungskonfigurationsfenster

Danach wird gefragt welche CMSIS-Libraries eingebunden werden sollen (Abbildung 21), hier ist mindestens **CORE** im Reiter **CMSIS** und **Startup** im Reiter **Device** zu wählen. Werden mehr Features benötigt, können diese in den entsprechenden Reitern aktiviert werden. Ein späteres Ändern der Laufzeitumgebung ist über einen Klick auf die entsprechende Schnelzugriffsschaltfläche ohne größere Umstände möglich. Die Konfiguration ist mittels Klick auf **OK** zu bestätigen.

 Zu beachten: Während die meisten CMSIS-Libraries frei verfügbar sind, gibt es auch einige wenige (File System, Graphics, Network, USB) welche nur mit einer Pro Version von  $\mu$ Vision verwendbar sind. Ist dies der Fall muss auf freie Libraries ausgewichen, oder eine Pro Version erworben werden.

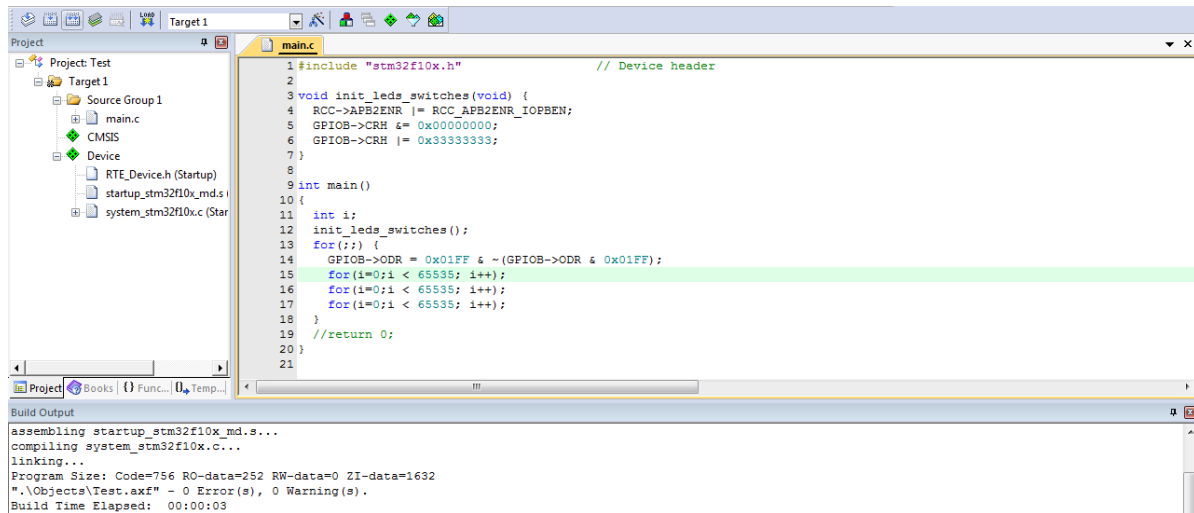


Abbildung 22: Beispielprogramm

In Abbildung 22 ist ein kleines Beispielprogramm zu sehen, welches die LEDs der LED-/Schalterplatine blinken lässt. Links zu sehen ist der Projektbaum. C-Files werden über den entsprechenden Menüpunkt (Rechtsklick auf die **Source Group 1** und dann **Add Item to Group...**) angelegt und direkt in das Projekt eingebunden. Der passende Dateiname und Typ ist im folgenden Fenster (Abbildung 23 auf der nächsten Seite) entsprechend zu wählen. Das Listing 1 auf Seite 22 kann einfach in das C-File kopiert werden.

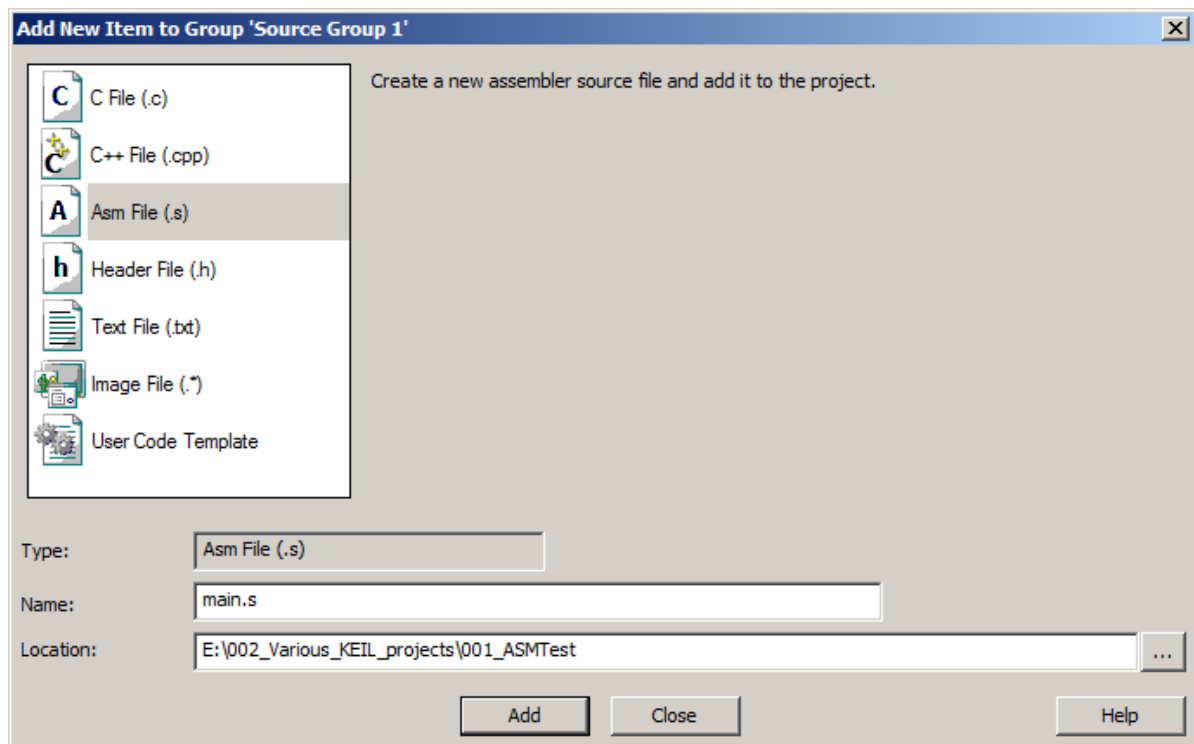


Abbildung 23: Dateierstellungsdialog

Listing 1: LED Blinklicht

```
#include "stm32f10x.h"

void init_leds_switches(void) {
    RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;
    GPIOB->CRH &= 0x00000000;
    GPIOB->CRH |= 0x00000003;
}

int main()
{
    int i;
    init_leds_switches();

    for (;;) {
        GPIOB->ODR = 0x01FF & ~(GPIOB->ODR & 0x01FF);
        for (i=0; i < 65535; i++);
        for (i=0; i < 65535; i++);
        for (i=0; i < 65535; i++);
    }
}
```

```
}
```

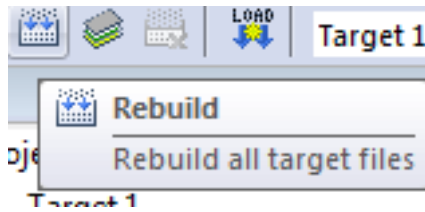


Abbildung 24: Schaltfläche zum kompilieren

Als nächstes muss das geschriebene Programm kompiliert werden, dazu ist einfach auf die Schaltfläche (Abbildung 24) zu klicken. Nach einigen wenigen Sekunden sollte dieser Vorgang erfolgreich abgeschlossen sein (Ausgabe im Logfenster<sup>4</sup> beachten).

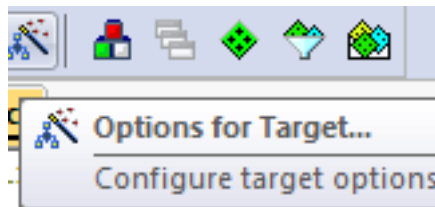


Abbildung 25: Optionsschaltfläche

Bevor nun aber das fertig kompilierte Programm auf das Übungssystem geflasht werden kann, muss der Debugging Adapter<sup>5</sup> angeschlossen (Abbildung 26 auf der nächsten Seite) und dessen Software geupdated werden. Dies kann mittels Klick auf den Zauberstab (Abbildung 25) gestartet werden werden.

<sup>4</sup>Beschreibung der Fenster siehe Abbildung 18 auf Seite 17

<sup>5</sup>In unserem Fall ein Keil ULINK-ME





Abbildung 26: Aufbau des Minimalsystems



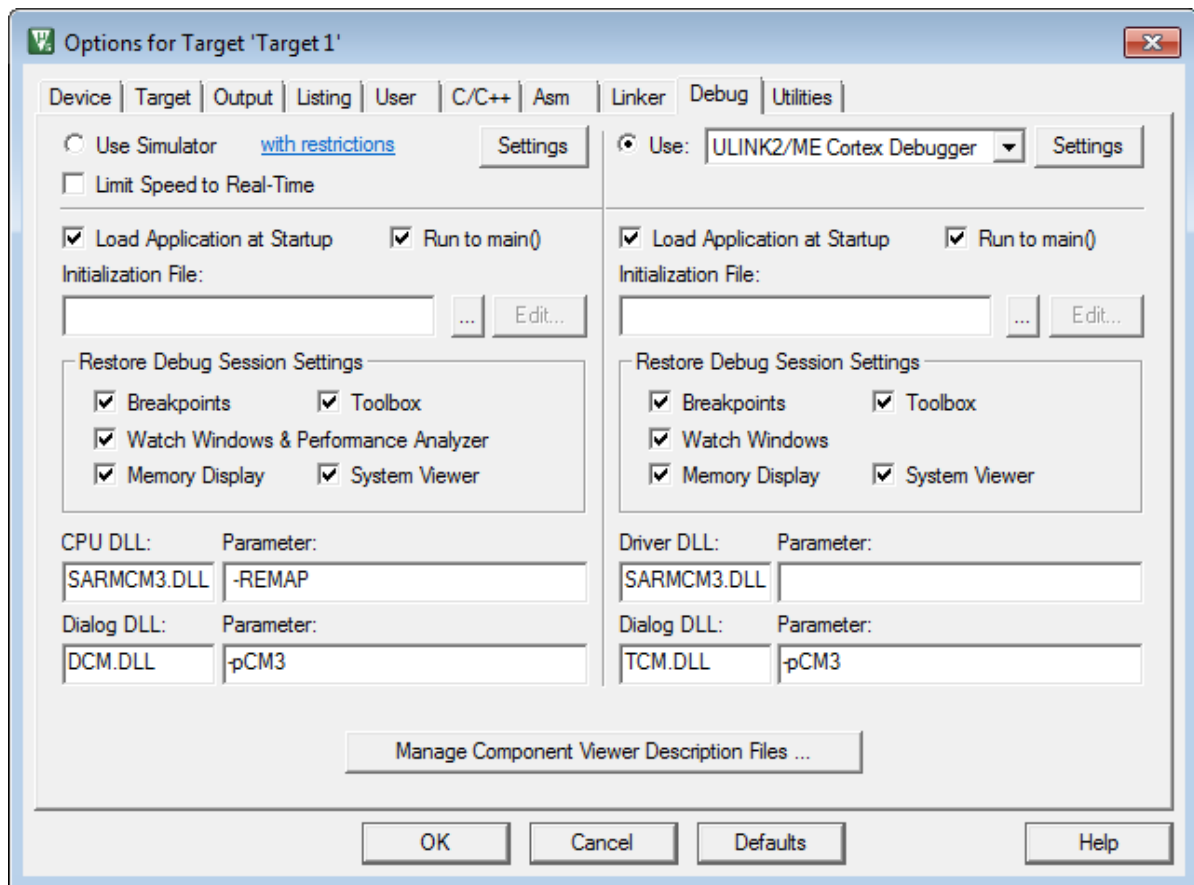


Abbildung 27: Optionsfenster

Danach geht das in Abbildung 27 abgebildete Fenster auf. Eventuell steht die Auswahl nicht auf dem Debug Reiter oben, dann ist dies manuell mittels Mausklick durchzuführen. Auf der rechten Seite muss der Radio Button bei **Use** ausgewählt werden und im Dropdown daneben **ULINK2/ME Cortex Debugger**. Wenn die Updateaufforderung (Abbildung 28 auf der nächsten Seite) nicht automatisch auftaucht, muss der Button **Settings** geklickt werden.

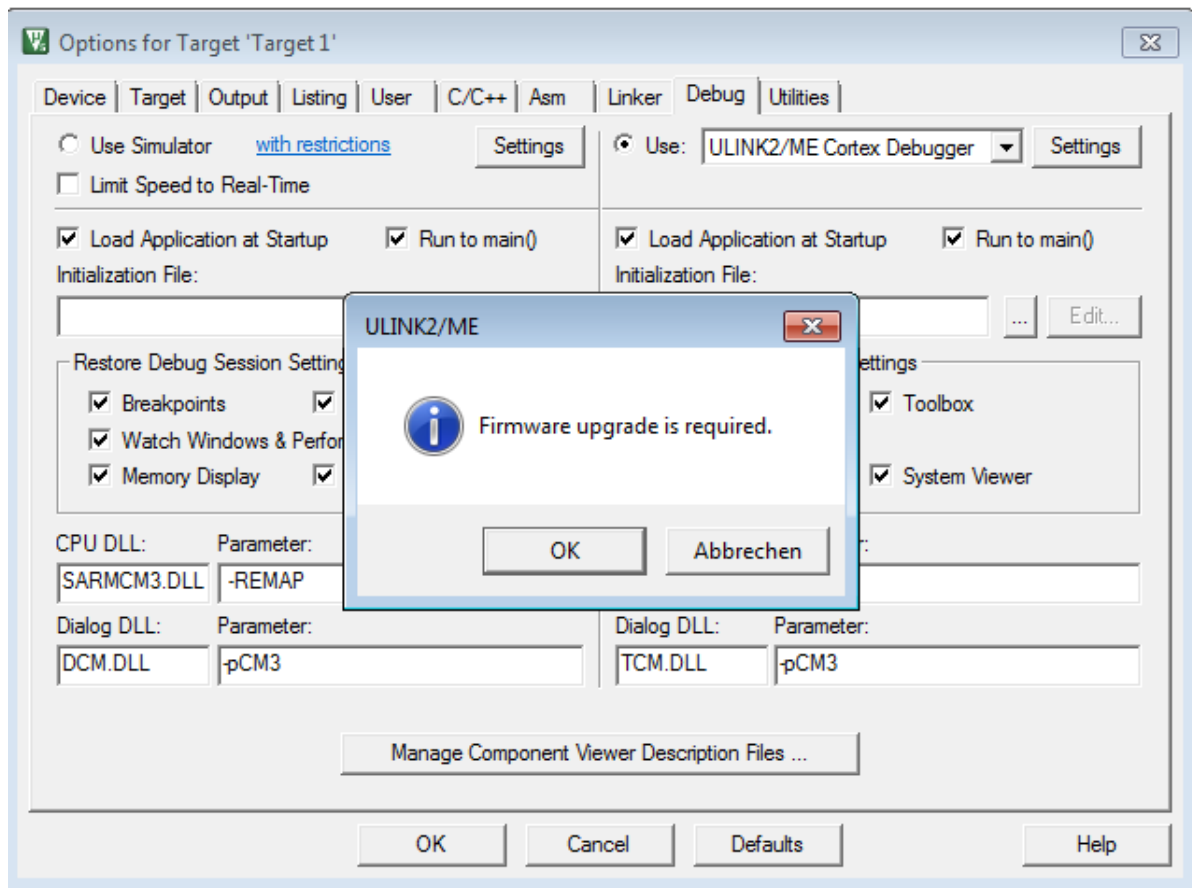




Abbildung 28: Firmwareupdate

 **Achtung:** Das Firmware Update ist notwendig um den Debugger zusammen mit  $\mu$ Vision 5 zu verwenden, allerdings ist dieser dann nicht mehr mit Version 4 kompatibel, hierfür muss die Firmware wieder mittels externem Programm auf die Alte geändert werden.

Die Meldung zum Update der Firmware muss mittels Klick auf den Button **OK** bestätigt werden. Danach muss gewartet werden, bis das Update durchgelaufen ist.

 **Achtung:** Trennen Sie in dieser Zeit nicht das USB Kabel des Debuggers, da dieser sonst gegebenenfalls irreperabel beschädigt werden könnte!

Wenn das Update fertig durchgelaufen ist, und wieder das Optionsfenster (Abbildung 27 auf der vorherigen Seite) zu sehen ist, stellen Sie sicher, dass die dort gezeigten Einstellungen übereinstimmen, und klicken Sie dann auf **OK**.

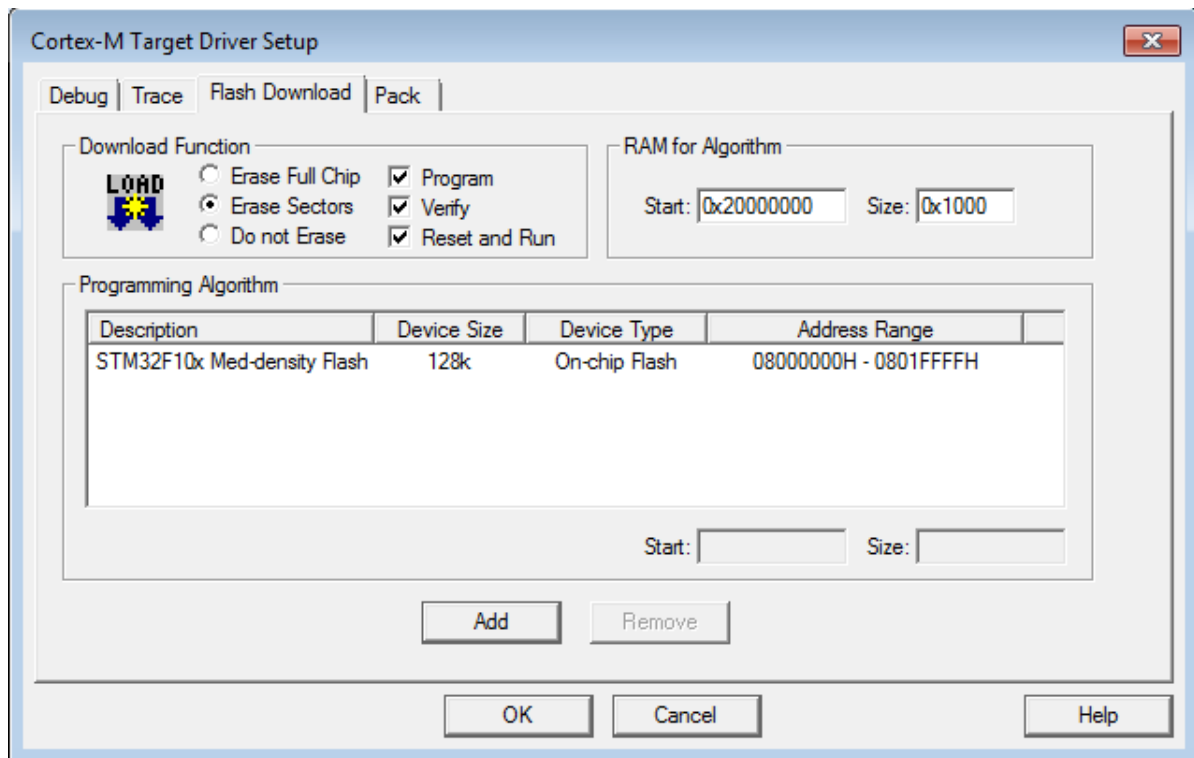


Abbildung 29: Debugger Einstellungen

Im nun auf gegangenen Fenster (Abbildung 29) sollte die Checkbox **Reset and Run** aktiviert werden. Dies ist zwar für den korrekten Betrieb nicht nötig, erleichtert aber das Arbeiten sehr, da nicht nach jedem mal neu flashen der Reset Knopf des Prozessors gedrückt werden muss.

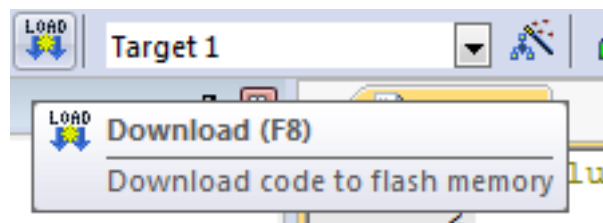


Abbildung 30: Load Button

Nun kann das kompilierte Programm mittels des Load Buttons (Abbildung 30) auf den Zielprozessor geladen werden. Wenn die Einstellungen, welche bei Abbildung 29 zu sehen sind richtig angewendet wurden, sollte nun die LEDs der LED-/Schalterplatine

anfangen zu blinken. Damit ist die  $\mu$ Vision IDE in der Version 5 fertig eingerichtet und bereit verwendet zu werden!

### 3 Debugging

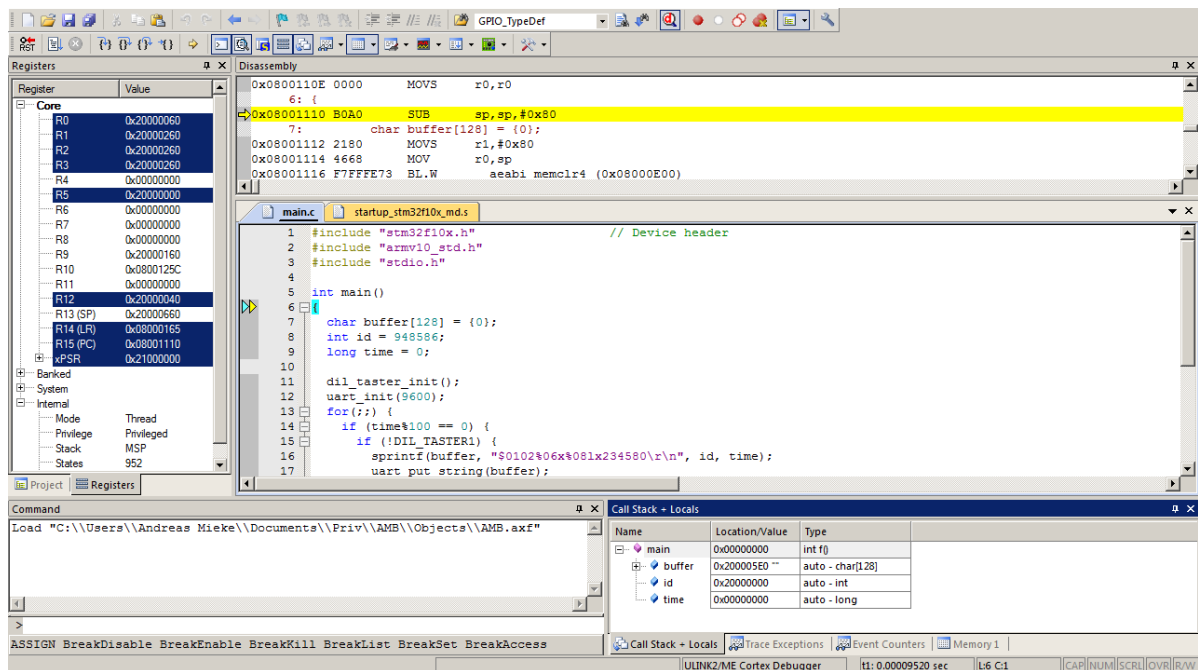
Im nächsten Teil dieses Tutorials wird auf den Debugger der Keil  $\mu$ Vision 5 eingegangen werden, und des weiteren auch erläutert, wie sich eben dieser vom Debugger der  $\mu$ Vision 4 unterscheidet.

In der Symbolleiste (Abbildung 31) befinden sich entsprechende Icons zum starten des Debuggers und um Breakpoints zu verwalten. Mit Hilfe dieser Buttons kann der Debugger gestartet und gestoppt werden (Lupe ganz links) und Breakpoints hinzugefügt (roter Punkt), aktiviert und deaktiviert (weißer Punkt) werden. Des weiteren können alle Breakpoints gleichzeitig mittels Klick auf das Icon mit den zwei weißen Kreisen mit rotem Rand deaktiviert werden. Der Button rechts daneben löscht alle Breakpoints komplett.

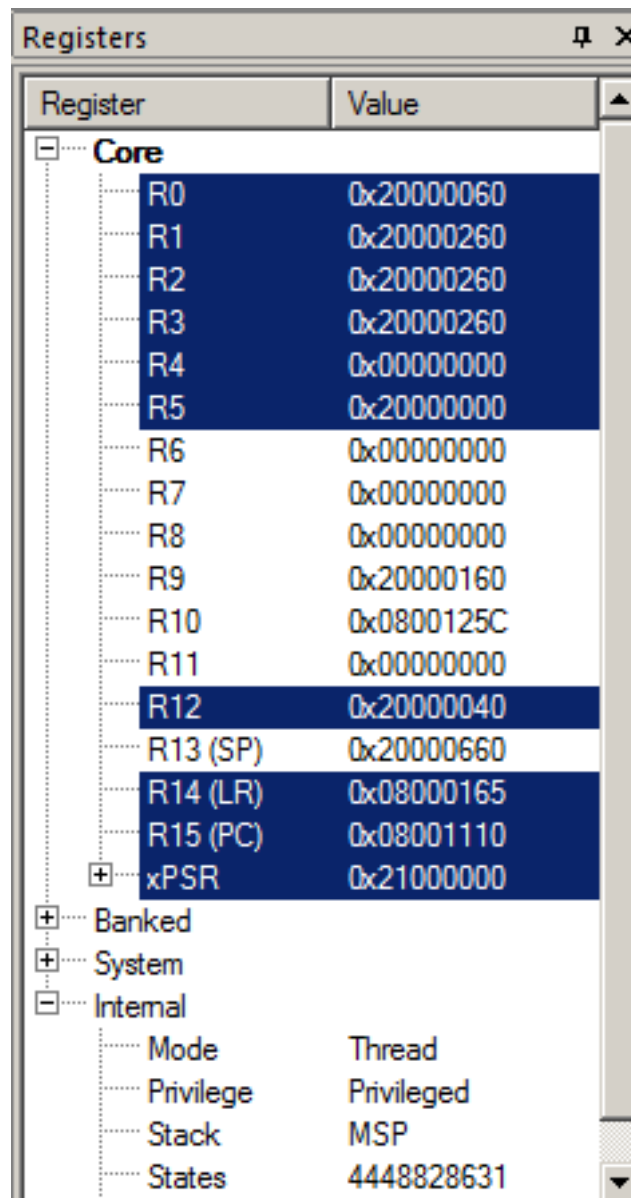


Abbildung 31: Debugging-Symbolleiste

Wird der Debugger nun mittels Klick auf die Lupe in Abbildung 31 aktiviert, so wird das aktuelle Programm neu auf den Prozessor geflasht und gestartet, gleichzeitig erweitert sich auch das User Interface der IDE (Abbildung 32 auf der nächsten Seite) um für das Debugging relevante Fenster. Weiters wird, wenn kein Breakpoint gesetzt ist, das Programm bis zum main-Aufruf ausgeführt und dort angehalten.

Abbildung 32: User Interface der Keil  $\mu$ Vision 5 im Debugging Modus

Im Register Fenster (Abbildung 33 auf der nächsten Seite) sind alle Register des Mikrocontrollers zu sehen, zusätzlich dazu sind jene, welche sich seit dem letzten Step verändert haben, blau eingefärbt. Das Register Fenster ist für das Debugging von C/C++ Programmen nicht wirklich sinnvoll, für Assembler-Programme aber eine große Hilfe.



Register	Value
<b>Core</b>	
R0	0x20000060
R1	0x20000260
R2	0x20000260
R3	0x20000260
R4	0x00000000
R5	0x20000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x20000160
R10	0x0800125C
R11	0x00000000
R12	0x20000040
R13 (SP)	0x20000660
R14 (LR)	0x08000165
R15 (PC)	0x08001110
xPSR	0x21000000
<b>Banked</b>	
<b>System</b>	
<b>Internal</b>	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	4448828631

Abbildung 33: Register Fenster

Ein weiteres Fenster ist das Disassembly-Fenster, zu sehen in Abbildung 34 auf der nächsten Seite, welches den zur Zeit ausgeführten Binärcode in Form von Assemblerbefehlen zeigt. Dies ist sinnvoll, wenn eine binäre Library analysiert und auf Fehler geprüft werden muss. Des weiteren kann man hier gut sehen bei welchem Teil einer Libraryfunktion das Programm zum Beispiel abstürzt oder hängen bleibt.

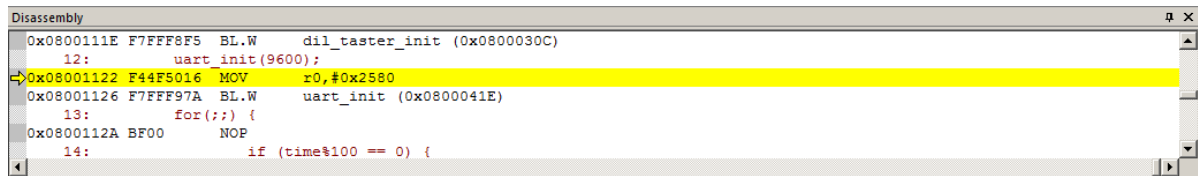


Abbildung 34: Disassembly Fenster

Im Hauptfenster (Abbildung 35) wird der zur Zeit ausgeführte Programmcode angezeigt, mit zwei Pfeilen auf der Seite, welche die aktuelle Position und die Cursor-Position anzeigen. Der Cursor-Pfeil macht Sinn, wenn man die Funktion "Run to Line" benutzen will. Dies ist der zweite Button von rechts in Abbildung 36.

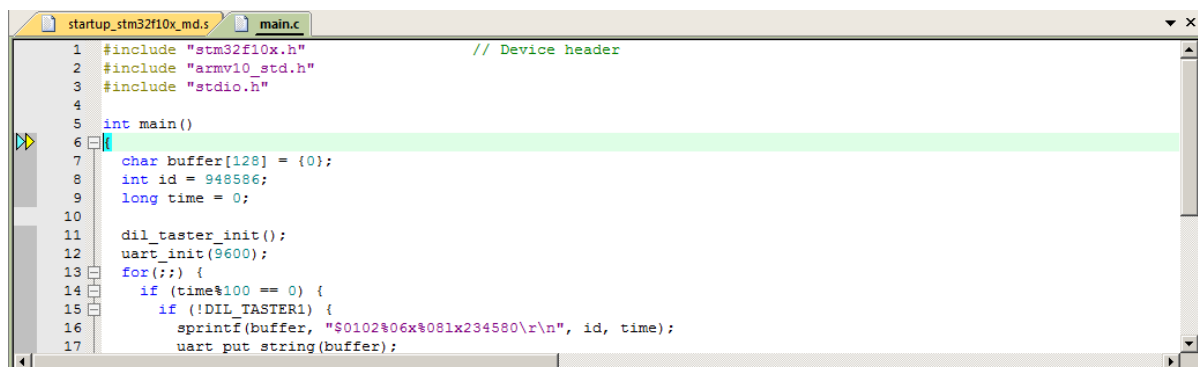


Abbildung 35: Hauptfenster



Abbildung 36: Stepping-Buttons

Des weiteren kann durch einen Klick auf die graue Fläche neben den Zeilennummern für die entsprechende Zeile ein Breakpoint aktiviert (und deaktiviert) werden. Siehe dazu Abbildung 37 auf der nächsten Seite. Ein Breakpoint hält die Ausführung der laufenden Anwendung an dieser Stelle an und setzt diese erst nach einem Klick auf Run (Abbildung 36, zweiter Button von links) oder auf einen der Stepping-Buttons (zweite Buttongruppe von rechts) wieder im entsprechenden Modus fort. Dies ist sehr sinnvoll um sich zum Beispiel den Inhalt von Registern anzusehen und so Fehler zu suchen.

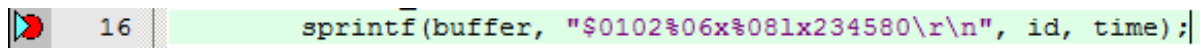


Abbildung 37: Programmzeile mit Breakpoint und Ausführungs-Pfeil

Neben den Hauptregistern selbst, Abbildung 33 auf Seite 30, kann im Debugger auch die Konfiguration von einzelnen Peripherieeinheiten angesehen werden. Hierfür gibt es jeweils Extra Fenster, welche über das Hauptmenü geöffnet werden kann, zu sehen in Abbildung 38, in diesem Beispiel sehen wir uns die Konfiguration von GPIOB an.

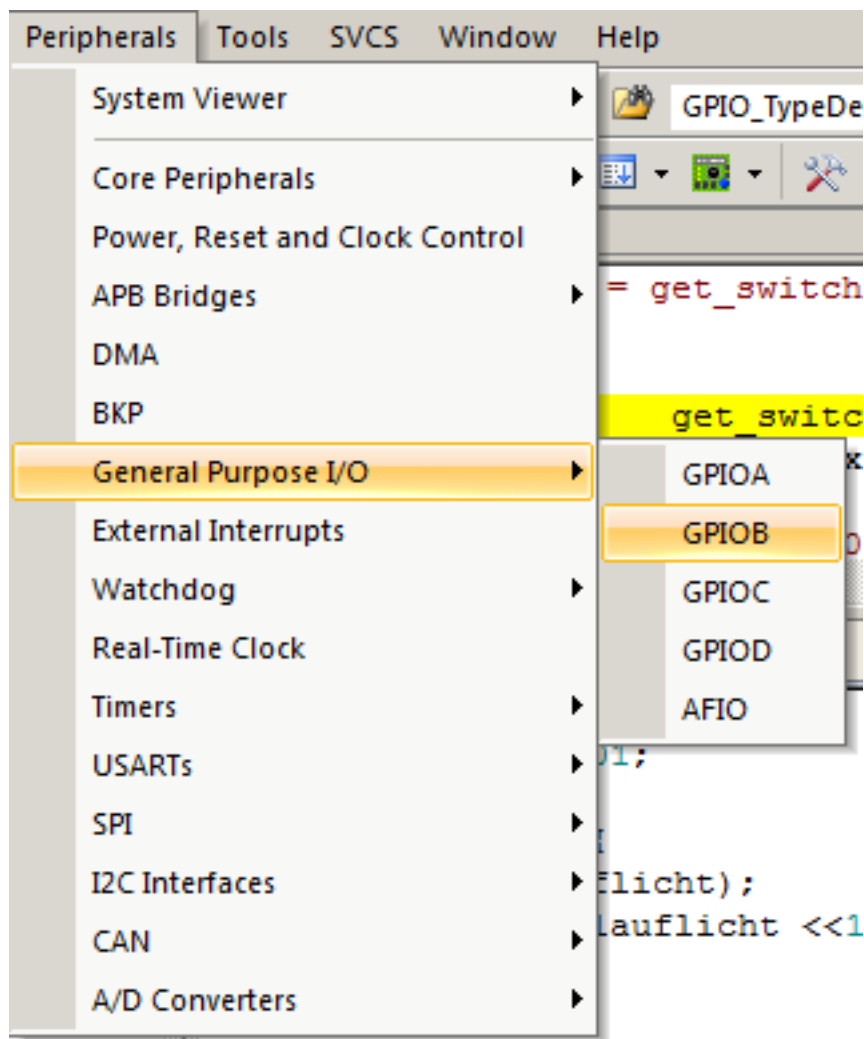


Abbildung 38: Peripherie-Hauptmenü

Nach dem Klick auf den entsprechenden Menüeintrag öffnet sich ein weiteres Fenster,



Abbildung 39. Mit Hilfe dieses Fensters kann nun sowohl die Konfiguration der entsprechenden Peripherieeinheit als auch die aktuellen Werte der Input/Output Register angesehen werden.

**General Purpose I/O B (GPIOB)**

Pin	CNF
PB.4	Input with pull-up/down
PB.5	Floating Input
PB.6	Floating Input
PB.7	Floating Input
<b>PB.8</b>	<b>GP output push-pull</b>
PB.9	GP output push-pull
PB.10	GP output push-pull
PB.11	GP output push-pull

Selected Port Pin Configuration  
 MODE: 3: Output (50 MHz) CNF: 0: GP output push-pull

Configuration & Mode Settings  
 GPIOB\_CRH: 0x33333333 GPIOB\_CRL: 0x44484444

GPIOB  
 GPIOB\_IDR: 0x000002D1  
 GPIOB\_ODR: 0x00000210  
 GPIOB\_LCKR: 0x00000000

Settings: Clock Enabled

Abbildung 39: GPIOB Registerfenster

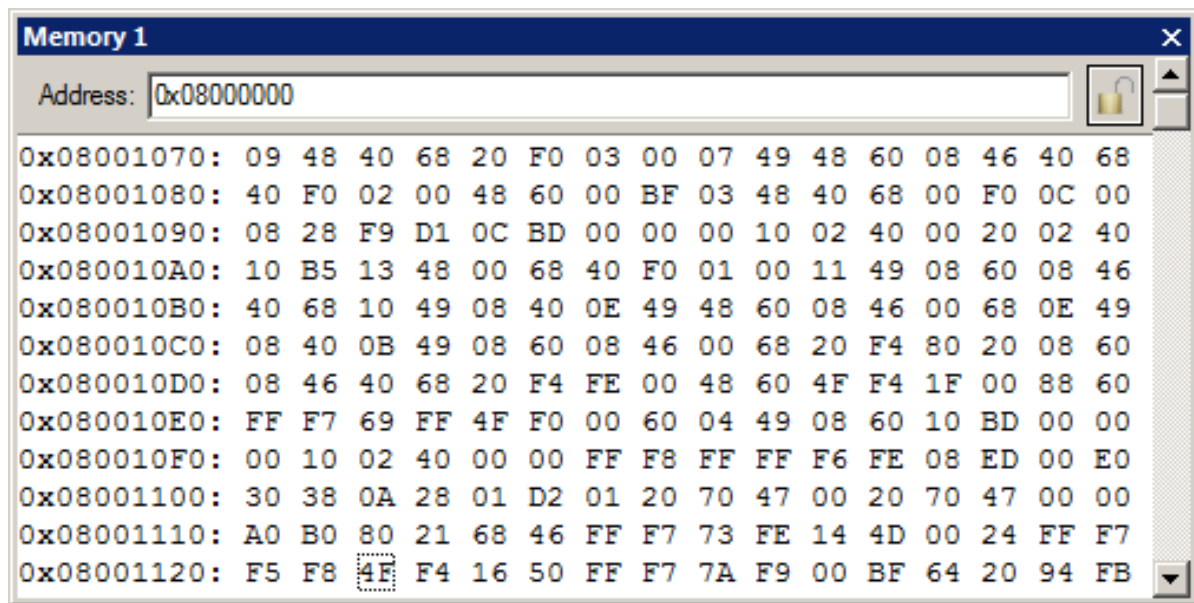


Abbildung 40: Memory Fenster

Eine weitere Möglichkeit des Debuggings ist es, das Memory Fenster zu benutzen. Dieses Fenster (Abbildung 40) zeigt den Inhalt des Speichers ab einer eingegebenen Adresse an. Hier zu sehen ist die Instruktion, welche gerade in Abbildung 34 auf Seite 31 ausgeführt wird. Weiters ist es Möglich mittels Rechtsklick den Inhalt von Speicherbereichen zu ändern. Dies ist insbesondere dann sinnvoll, wenn man den Op-Code einer Instruktion oder ähnliches während das Programm läuft ändern möchte.

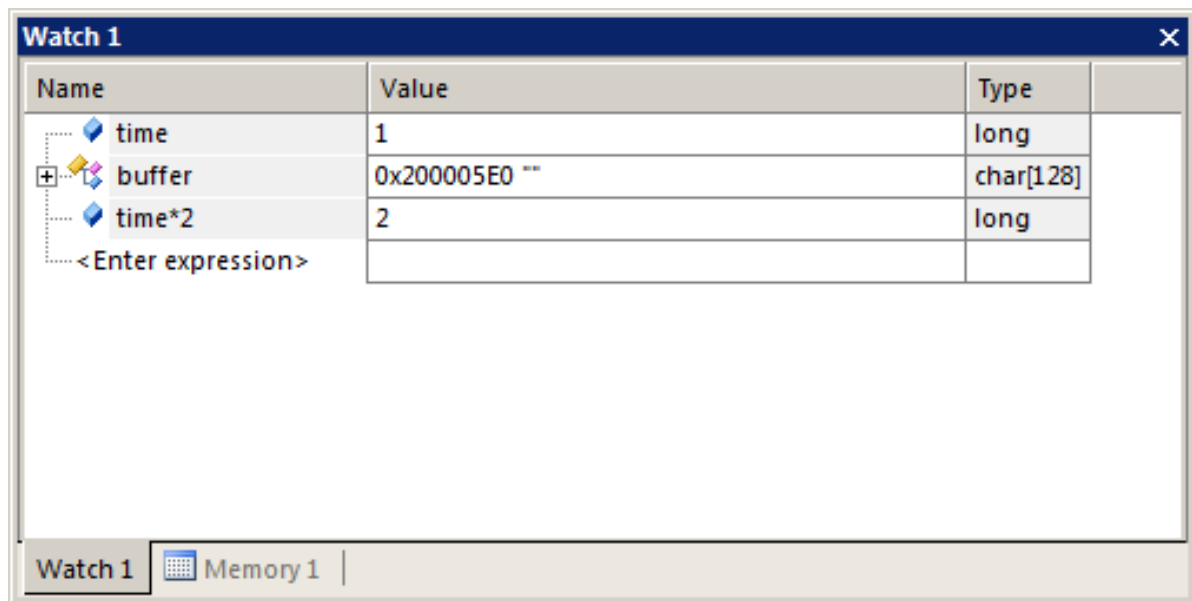


Abbildung 41: Watches Fenster

Mit Hilfe des Watches Fenster (Abbildung 41) können Variablen im C Programm "Beobachtet" werden. Das Fenster zeigt, sofern auf die Variable an der entsprechenden Programmstelle zugegriffen werden kann, immer den aktuellen Wert eben dieser an. Auch hier ist es möglich den Wert in Echtzeit zu ändern, außerdem können einfache Berechnungen durchgeführt werden.

### 3.1 Keil ULINK2/ME Debugger

Der ULINK/ME<sup>6</sup> ist der Standard-Debug-Adapter, welcher bis 2017 verwendet wurde und bei den alten Lehrsystemen immer noch verwendet wird. Die entsprechende Konfiguration und das nötige Firmware-Upgrade wurden hierfür schon in Abschnitt 2.4 auf Seite 17 erläutert. Hierbei sind besonders Abbildung 27 auf Seite 25 und Abbildung 29 auf Seite 27 zu beachten.

<sup>6</sup>Diese Bezeichnung bezeichnet die alte Firmware, nach dem Upgrade der Firmware für  $\mu$ Vision 5 wird dieser Adapter als ULINK2/ME bezeichnet und ist inkompatibel zu  $\mu$ Vision 4.

## 3.2 ST-Link Debugger

Das neue Übungssystem ist dafür vorgesehen mit einem ST-Link Debugger im SWD<sup>7</sup>-Modus verwendet zu werden. Dieser braucht im Gegensatz zu einem auf JTAG<sup>8</sup> basierenden Debugger weniger Pins und somit weniger Platz auf dem Zielsystem. Die IDE muss lediglich auf diesen neuen Debugger umgestellt werden, siehe dazu Abbildung 42. Nähere Infos dazu, wie man zu diesem Fenster kommt kann in Abschnitt 2.4 auf Seite 17 gefunden werden.

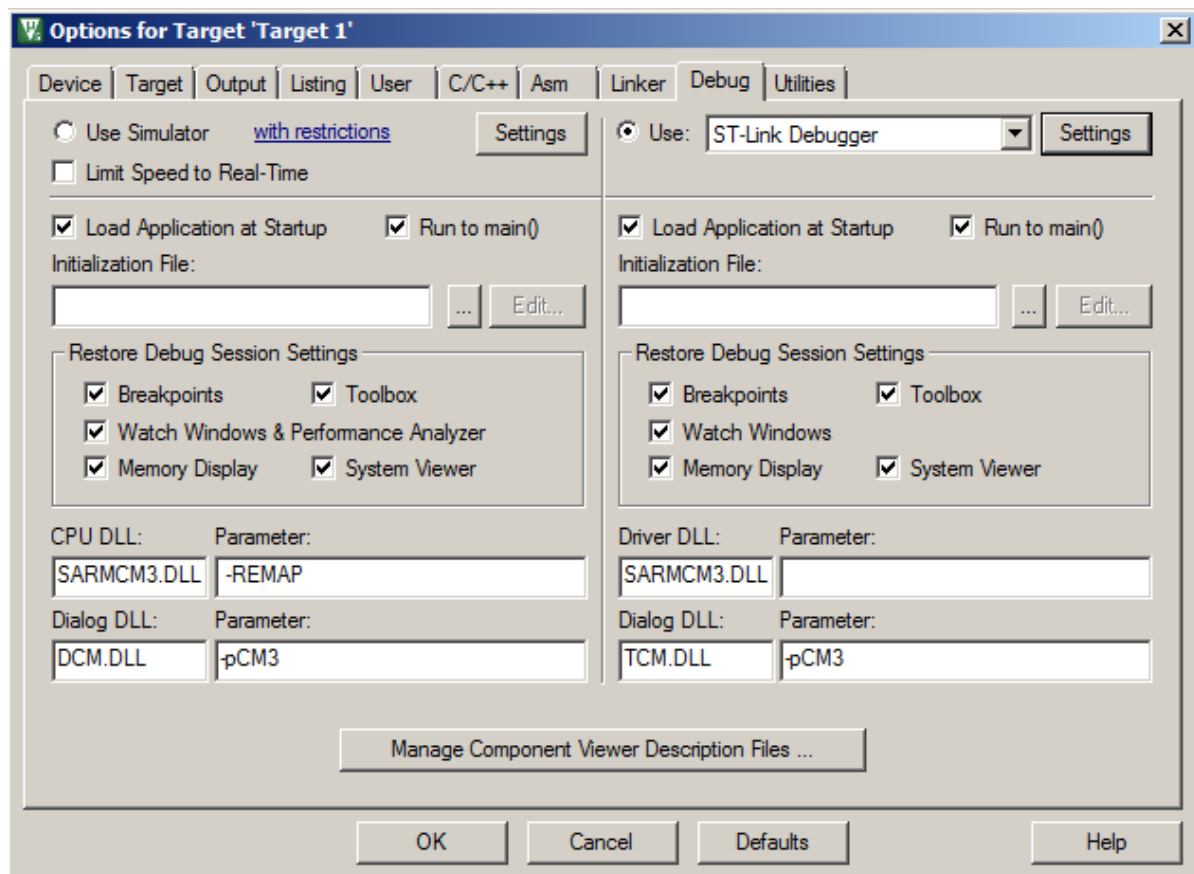


Abbildung 42: Debug Fenster mit ST-Link ausgewählt

<sup>7</sup>Serial Wire Debug

<sup>8</sup>Joint Test Action Group

## 4 Assembler-Programmierung

Zu Beginn wird die Programmierung vom Cortex M3 mittels Assembler unterrichtet, dementsprechend befasst sich auch dieses Tutorial zuerst mit der Assembler-Programmierung. Hierzu wird ein Beispielprogramm her genommen und erläutert wie die IDE eingestellt werden muss, und was alles zu tun ist um dieses Programm erfolgreich zu assemblieren und auszuprobieren.

Als erstes muss ein neues Projekt angelegt werden wie in Abschnitt 2.4 auf Seite 17 erklärt. Wichtig hierbei ist es die richtigen CMSIS-Packs zu wählen, da das Ziel ein Assembler-Projekt ist, ist hierbei das HTL Assembler-Pack (Abbildung 43) zu wählen.

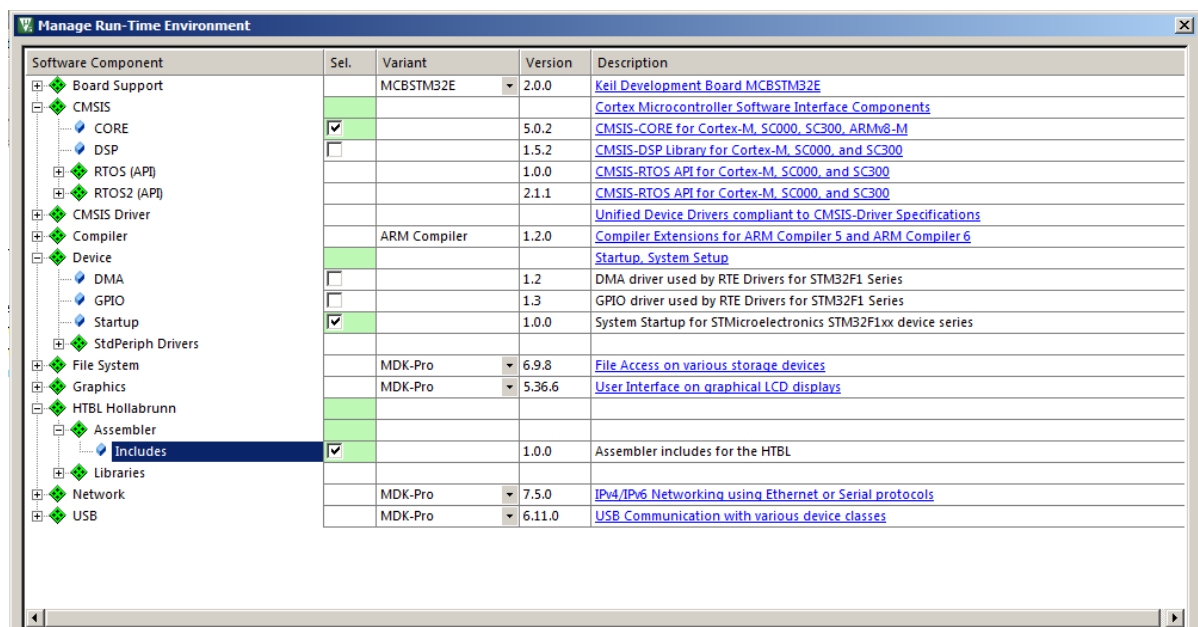


Abbildung 43: Paketwahldialog mit HTL Pack

Ist dies erledigt, kann eine neue Assembler Source-Datei erstellt werden, dazu klickt man mit der rechten Maustaste auf die **Source Group 1** und wählt im Kontextmenü **Add New Item to Group...**, im folgenden Fenster (Abbildung 44 auf der nächsten Seite) wählt man nun **Asm File** und vergibt einen passenden Namen (zum Beispiel **main.s**).

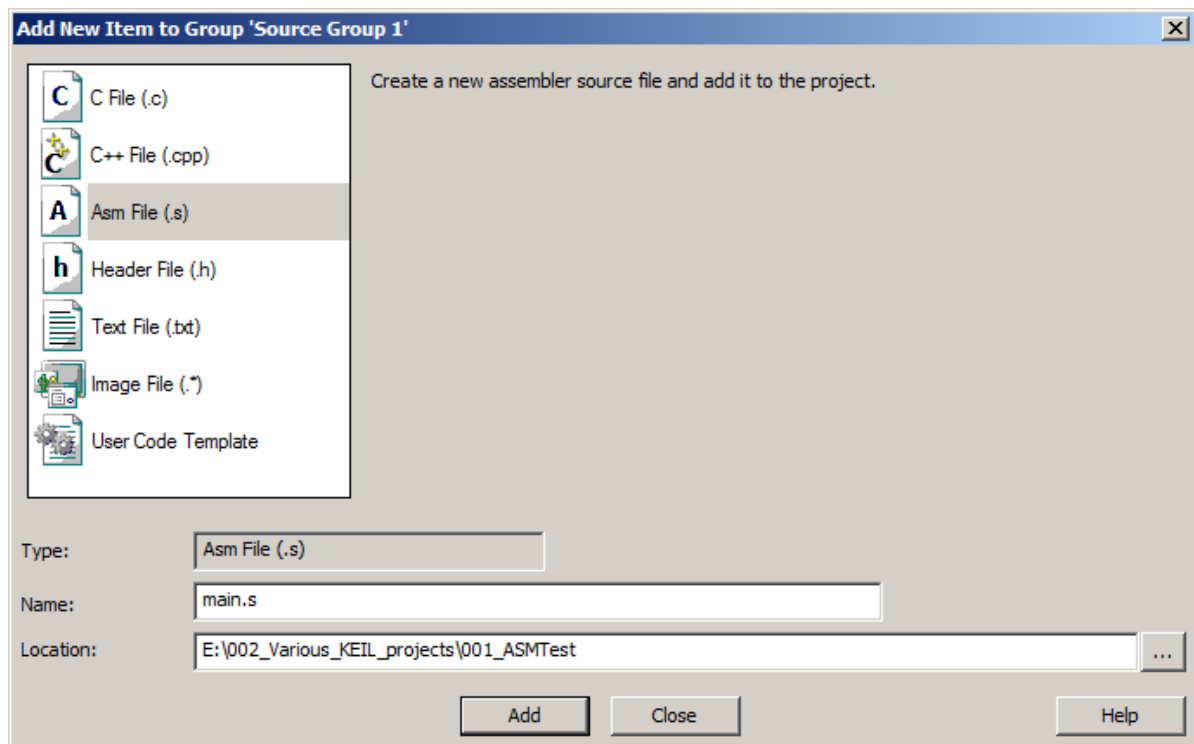


Abbildung 44: Dateierstellungsdialog

Durch einen Klick auf **Add** wird die Datei hinzugefügt und kann anschließend bearbeitet werden. In diesem Beispiel wird das erste Beispielprogramm (eine blinkende LED auf der LED-/Schalterplatine) programmiert. Hierzu ist der Source Code aus Listing 2 auf Seite 38 zu verwenden. In der zweiten Zeile wird die HTBL Memory Map inkludiert, welche durch das CMSIS Pack zur Verfügung gestellt wird.

Listing 2: LED Blinklicht

```

AREA BLINKEN, CODE, READONLY
INCLUDE STM32_F103RB_MEM_MAP.INC
EXPORT __main

__main          PROC
                BL      init_port
                LDR      R1, =GPIOB_ODR
__main_again    LDR      R0, [R1]
                EOR      R0, R0, #0x100
                STR      R0, [R1]
                BL      wait_500ms
                B        __main_again
                ENDP

```

```

init_port      PROC
                PUSH    {R0–R2, LR}

                MOV     R2, #0x8
                LDR     R1, =RCC_APB2ENR
                LDR     R0, [R1]
                ORR     R0, R0, R2
                STR     R0, [R1]

                LDR     R1, =GPIOB_CRH
                LDR     R0, [R1]
                LDR     R2, =0xFFFFFFFF0
                AND     R0, R0, R2
                MOV     R2, #0x03
                ORR     R0, R0, R2
                STR     R0, [R1]

                POP     {R0–R2, PC}
                ENDP

wait_500ms     PROC
                PUSH    {R0–R2, LR}
                MOV     R0, #0x1F4
                MOV     R1, #0
__wait_ms_loop MOV     R2, #0x63B
__wait_ms_loop1 SUB     R2, R2, #1
                CMP     R2, R1
                BNE     __wait_ms_loop1
                SUB     R0, R0, #1
                CMP     R0, R1
                BNE     __wait_ms_loop
                POP     {R0–R2, PC}
                ENDP

                END

```

Nun kann das Programm assembliert und anschließend auf den Cortex geladen werden. Dieses vorgehen ist näher in Abschnitt 2.4 auf Seite 17 beschrieben. Nun sollte, nach einem Reset des Microcontrollers, die erste LED von rechts (PB8) auf der LED-/Schalterplatine blinken. Nähere Informationen zum Debugging können in Abschnitt 3 auf Seite 28 gefunden werden.

## 5 C-Programmierung

Nach dem Einstieg mit Assembler, wird auch die Programmierung mit der Sprache C unterrichtet, die Projekterstellung für diese Sprache weicht nicht sonderlich von der für Assembler (Abschnitt 4 auf Seite 37) und dem vorgehen im Allgemeinen (Abschnitt 2.4 auf Seite 17) ab. Allerdings ist auch hier zu beachten das richtige CMSIS Pack der HTL zu verwenden. Dies ist im Anfang noch nicht so wichtig, da die ersten Programme ohne die HTL **STDLib**<sup>9</sup> auskommen, allerdings ist diese in den späteren Programmen unbedingt nötig. Um die Library einzufügen, ist diese im Pack Manager (Abbildung 45) auszuwählen.

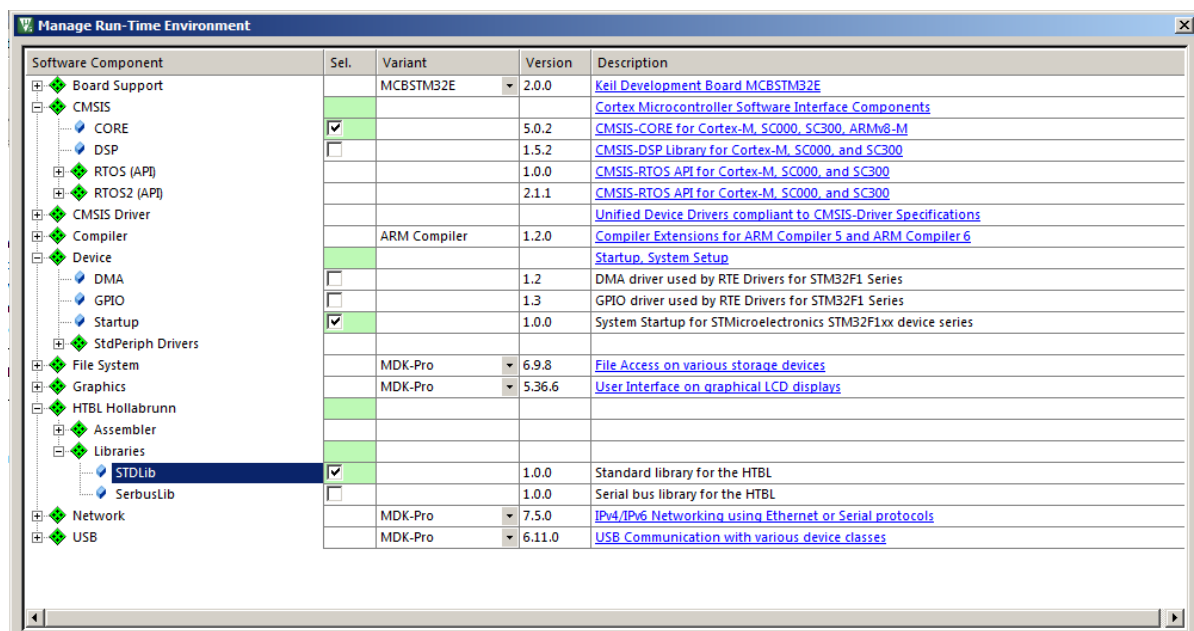


Abbildung 45: Paketwahldialog mit HTL STDLib Pack

Als erstes Demoprogramm ist hier ein Lauflicht, welches in Listing 3 auf Seite 40 zu sehen ist, zu realisieren.

Listing 3: LED Lauflicht

```
#include <stm32f10x.h>

void wait(void);
void init_leds_switches(void);
void set_leds(char value);
```

<sup>9</sup>Eine Library mit Helferfunktionen, welche speziell auf das in der HTL verwendete Minimalsystem zugeschnitten ist



```
char get_switches(void);
void wait_ms(int ms);

void init_leds_switches() {
    int temp;

    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;
    RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;

    temp = GPIOA->CRL;
    temp &= 0x00000000;
    temp |= 0x88888888;
    GPIOA->CRL = temp;
    GPIOA->BSRR = 0x00FF;

    temp = GPIOB->CRH;
    temp &= 0x00000000;
    temp |= 0x33333333;
    GPIOB->CRH = temp;
}

void set_leds(char value) {
    GPIOB->ODR = (GPIOB->ODR & 0xFFFF00FF) |
                ((value & 0x000000FF) <<8);
}

char get_switches() {
    return (GPIOA->IDR & 0x000000FF);
}

void wait_ms(int ms) {
    int i, j;

    for (i = 0; i < ms; i++) {
        for (j = 0; j < 1595; j++);
    }
}

void wait() {
    unsigned char speed;
    unsigned char i;

    speed = get_switches();
    speed = ~speed;
```

```
    speed = speed & 0x7f;

    for (i = 0; i <= speed; i++) {
        wait_ms(100);
    }
}

int main () {
    char i;
    char hs;
    char lauflicht;

    init_leds_switches();
    hs = get_switches() & 0x80;
    while (hs) {
        lauflicht = 0x01;
        i = 0;
        while (i < 8) {
            set_leds(lauflicht);
            lauflicht = lauflicht << 1;
            i++;
            wait();
        }
        hs = get_switches() & 0x80;
    }

    while (1);
}
```

Auch hier verhält sich das Debugging so wie in Abschnitt 3 auf Seite 28 beschrieben.



Dieses Werk ist lizenziert unter einer Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz.

Dieses Dokument wird laufend aktualisiert, die aktuelle Version dieses Dokuments kann stets unter folgender Adresse abgerufen werden:

[https://git.1750studios.com/diploma-thesis/001\\_Keil\\_MDK5\\_Tutorial](https://git.1750studios.com/diploma-thesis/001_Keil_MDK5_Tutorial)