Breaking the 32kB Limit

0r2

Verwendung der ARM GCC Toolchain unter der Keil MDK-Lite IDE

Using the ARM GCC Toolchain with Keil MDK-Lite IDE

Compilieren/Linken von Baseflight bzw. Harakiri

Compiling/Linking of Baseflight and Harakiri

© by Joerg Quinten
aBUGSworstnightmare

Revisions:

0r1: initial release

0r2: added english translations, made some modifications/additions

1. Einleitung / Introduction

Die FunFly 'Naze32' ist ein Flightcontroller für Multicopter.

Als Firmware kommen Baseflight (Original des Naze32 Hardwareentwicklers Timecop) bzw. Harakiri (Portierung/Erweiterung von Baseflight von Roberto - aka crashpilot1000) zum Einsatz.

Um die Firmware selber editieren, compilieren bzw. debuggen zu können benötigt man ein IDE (Integrated Development Environment) welche keine Codegrößenbeschränkung hat. Im Falle der Keil MDK-Lite, der kostenlosen Version der Keil IDE, liegt die maximale Codegröße bei 32kB was für die Naze32 Firmware deutlich zu wenig ist. Was liegt also näher als die Keil IDE so einzurichten dass der GCC ARM Toolchain verwendet wird um diese Beschränkung aufzuheben.

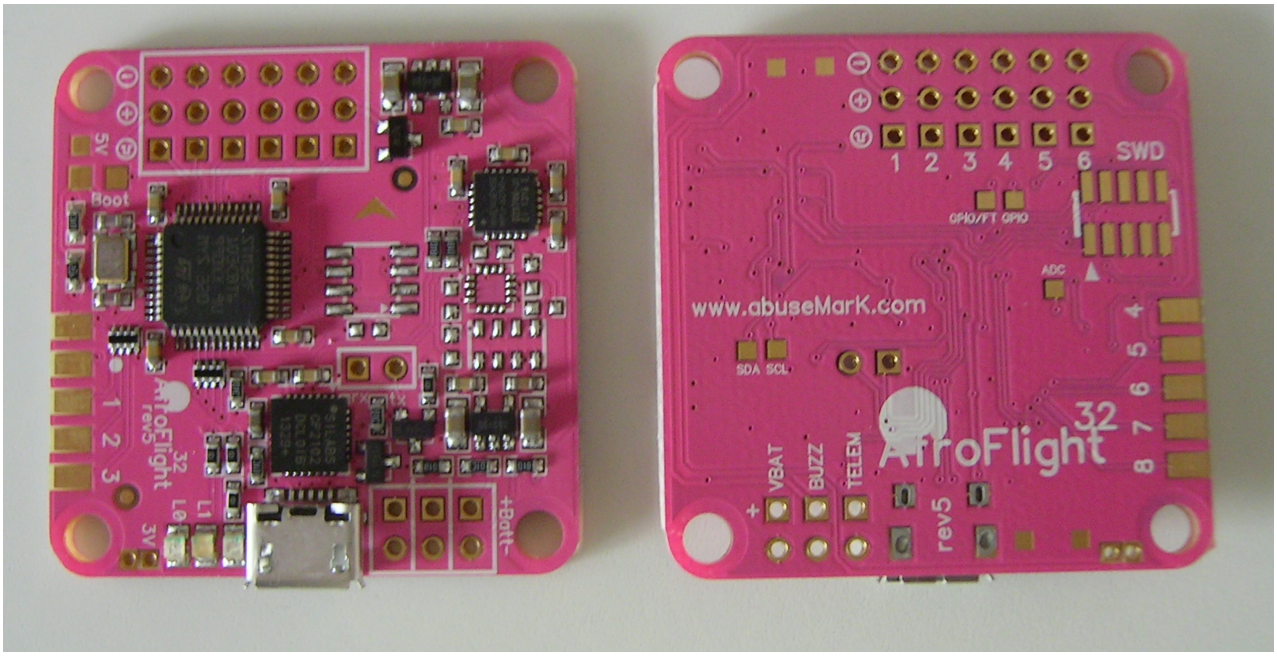
Dieses Tutorial soll zeigen welche Schritte dafür notwendig sind. Am Ende verfügt man dann über eine kostenlose, voll funktionale Entwicklungsumgebung für ARM Mikrocontroller und kann sich entweder mit Baseflight/Harakiri oder eigenen Projekten austoben.

The FunFly 'Naze32' is a multicopter flight controller.

Naze32 is shipped with the Baseflight firmware (original from Naze 32 Hardware developer Timecop) but can also be used with Harakiri (port/expansion of Baseflight, by Roberto – aka crashpilot1000).

If you intend to edit/compile/debug the firmware of the Naze32 by yourself you will need an IDE (Integrated Development Environment) which has no codesize limitation. The free-of-cost MDK-Lite is limited to 32kB which is insufficient for the Naze32 firmware. So why not configure the Keil IDE to use the ARM GCC toolchain to overcome this limitation?

This tutorial will give you a step-by-step guide what to do. When you're done you will have a free-of-charge, full-featured toolchain for ARM microcontrollers. You can use it for working with Baseflight/Harakiri or for your own projects.



AcroNaze32 – Hot Pink 'limited edition'

2. Vorbereitung / Preparation

Zuerst muss die benötigte Software geladen werden. Es handelt sich dabei um:

- Keil MDK-Lite (32KB) Edition

You need to download the required software:

- Keil MDK-Lite (32kB) Edition

http://www.keil.com/arm/mdk.asp

(zum Zeitpunkt der Erstellung dieses Tutorials in der Version V5.00)

(at the time of writing this tutorial the current revision is V5.00)

- GNU Tools für ARM Embedded Processors

- GNU Tools for ARM Embedded Processors

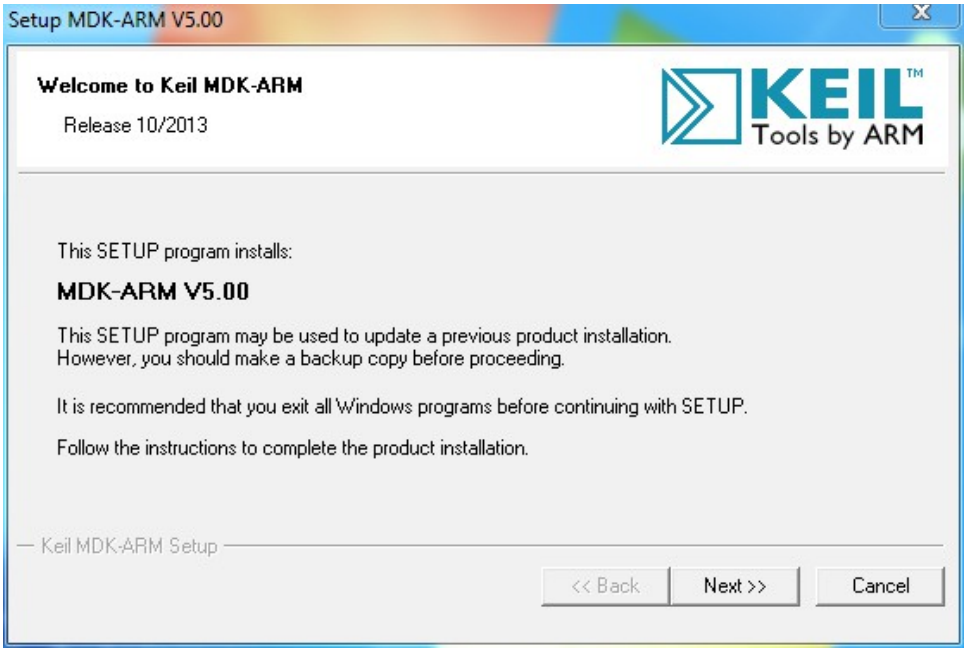
https://launchpad.net/gcc-arm-embedded/

(zum Zeitpunkt der Erstellung dieses Tutorials in der Version 4.7-2013-q3-update)

(at the time of writing this tutorial the current version is 4.7-2013-q3-update)

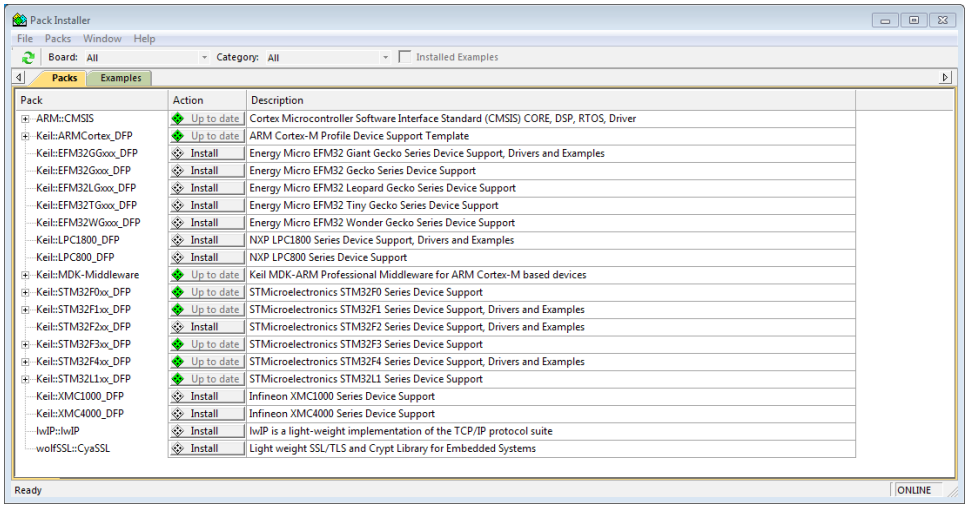
Beide Software-Pakete werden nun installiert. Gestartet wird mit Keil MDK-Lite.

Both software-pakets need to be installed. The Keil MDK-Lite will make the start.



Die Installation ist problemlos und bedarf keiner weiteren Erläuterung. Am Ende der Installation startet die Toolchain mit dem Pack Installer. Hier werden die Pakete für ARM:CMSIS sowie für die STM32-Serien installiert.

The installation is straight forward and needs no additional explanation. At the end of the installation the Pack Installer get's started. The ARM:CMSIS and the packets for the STM32-series needs to be installed.



Jetzt ist die ARM GCC Toolchain an der Reihe. Hier sollte darauf geachtet werden dass das Installationsverzeichnis im ROOT-Directory liegt. Dadurch wird die Pfadangabe vereinfacht um später keine Probleme mit rekursiven Pfadangaben zu haben.

Now you need to install the ARM GCC toolchain. Install it to your harddisks ROOT-directory. This will keep the path simple and prevent problems with recursive paths.

Für das Tutorial wird folgender Pfad verwendet:

This tutorial uses the installation path shown below:

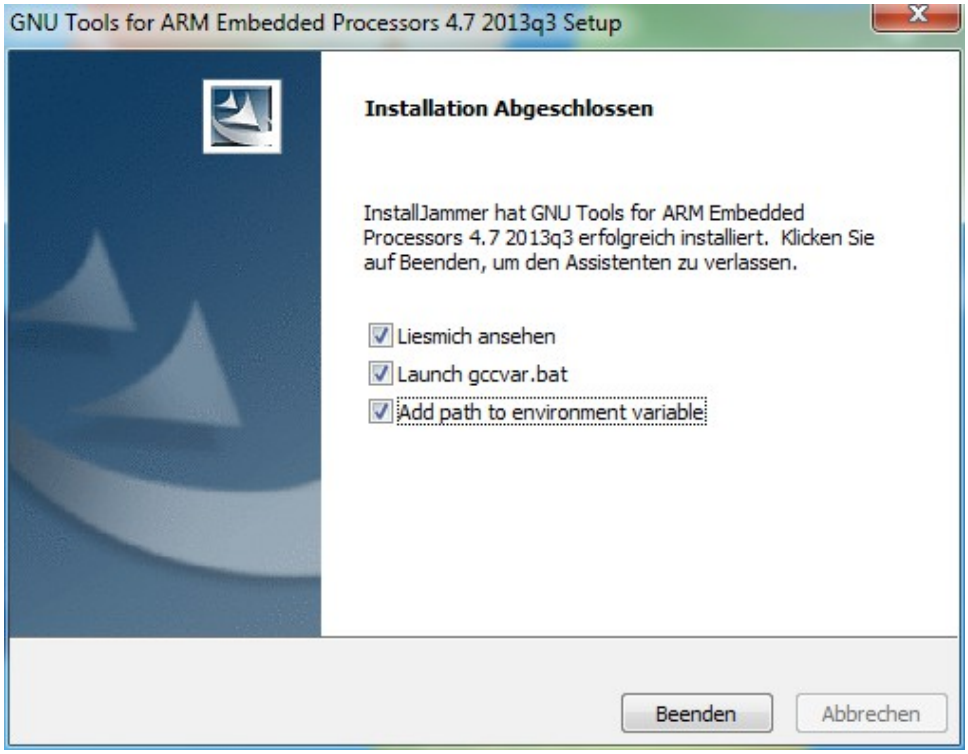
```
C:\GNUARM\4_7_2013q3
```

Es empfiehlt sich die Versionsnummer beizubehalten. Dadurch können später weitere Toolchain-Versionen installiert werden ohne die Settings bei bestehenden Projekten ändern zu müssen.

Keep the version number of the ARM GCC toolchain. This allows you to install different compiler revisions without the need of changing each projects settings.

Am Ende der Installation wird noch die Checkbox für 'Add path to environment variable' gesetzt. Wer will kann sich auch die Readme.txt ansehen sowie den Compiler starten.

Check the 'ADD PATH' checkbox and finish the installation. If you want to, you can have a look at the readme.txt and 'dry-start' the compiler for the first time.


GNU Tools for ARM Embedded Processors 4.7 2013q3 Setup
Installation Abgeschlossen
InstallJammer hat GNU Tools for ARM Embedded Processors 4.7 2013q3 erfolgreich installiert. Klicken Sie auf Beenden, um den Assistenten zu verlassen.
Liesmich ansehen
Launch gccvar.bat
Add path to environment variable
Beenden
Abbrechen

3. OPTIONAL – STM32 Support Libraries

Wer auch eigene Projekte auf Basis der STM32-MCUs realisieren möchte sollte auch gleich die original STM Support Libraries laden und installieren.

Für die Arbeit mit dem Baseflight- bzw. Harakiri-Projekt werden diese nicht benötigt!

If you intend to use the toolchain for your own projects based on the STM32 MCUs you should also install the STM Support Libraries.

You can skip this step if you intend to work with Baseflight and/or Harakiri only!

Die STM32 Standard Peripheral Drivers werden von der STM Webseite geladen werden. Dieser Download ist z.B. unter den Design Resources zum STM32F103CB zu finden:

The STM32 Standard Peripheral Drivers were available on the STM website. Find them in example among the design resources for STM32F103CB:

http://www.st.com/web/catalog/mmc/FM141/SC1169/SS1031/LN1565/PF189782?s_searchtype=partnumber

Man verwendet die

You need to download

STM32F10x standard peripheral library

(zum Zeitpunkt der Erstellung dieses Tutorials in der Version V3.5.0, STSW-STM32054)

(at the time of writing this tutorial the current version is V3.5.0, STSW-STM32054)

Z.B. wird nun im ROOT-Verzeichnis ein Arbeitsverzeichnis anlegegt:

I.e. place your working directory in ROOT:

C:\STM32_Workspace

Nun werden die Standard Peripheral Library Files entpackt und folgende Dateien in das Arbeitsverzeichnis kopiert:

Now unzip the files and copy the files as shown below:

STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\CMSIS\CM3\CoreSupport
nach / to
STM32_Workspace\CMSIS

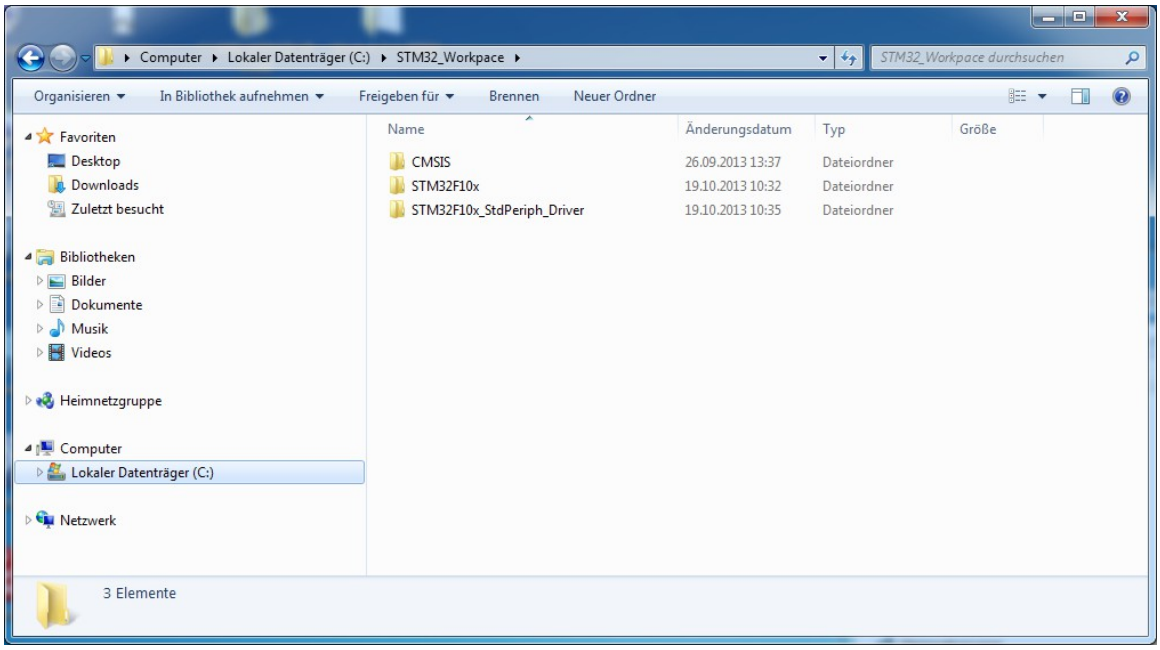
STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x
nach / to
STM32_Workspace\STM32F01x

STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\STM32F10x_StdPeriph_Driver
nach / to
STM32_Workspace\STM32F10x_StdPeriph_Driver

STM32F10x_StdPeriph_Lib_V3.5.0\Project\STM32F10x_StdPeriph_Template
nach / to
STM32_Workspace\STM32F10x_StdPeriph_Driver

Das Arbeitsverzeichnis sollte nun wie folgt aussehen:

You working directory should look like shown below now:



4. Baseflight Sources auschecken / checking out Baseflight sources

Da ich lieber auf dem Mac OS X und nicht unter Windows arbeite checke ich die Sourcen mit X-Code aus (Shame on me!). Hier gibt es jedoch verschiedenste Programme zur Verwendung unter Windows → eine entsprechende Ergänzung des Tutorials folgt.

Since I prefer to work on Mac OS X I'm using X-Code for checking out the sources. There are many programs for use on windows available → the tutorial will be updated!

Die Baseflight Sourcen in der Version r468 liegen bei. So kann jeder die nachfolgenden Schritte durchspielen.

The Baseflight sources version r468 were enclosed to this tutorial. So everybody is able to follow the next steps.

Die Baseflight Sourcefiles sind hier verfügbar:

The latest Baseflight source files were availabe here:

https://code.google.com/p/afrodevices/source/checkout

Die Harakiri Sourcefiles sind hier verfügbar:

The latest Harakiri sources can be found here:

https://github.com/Crashpilot1000

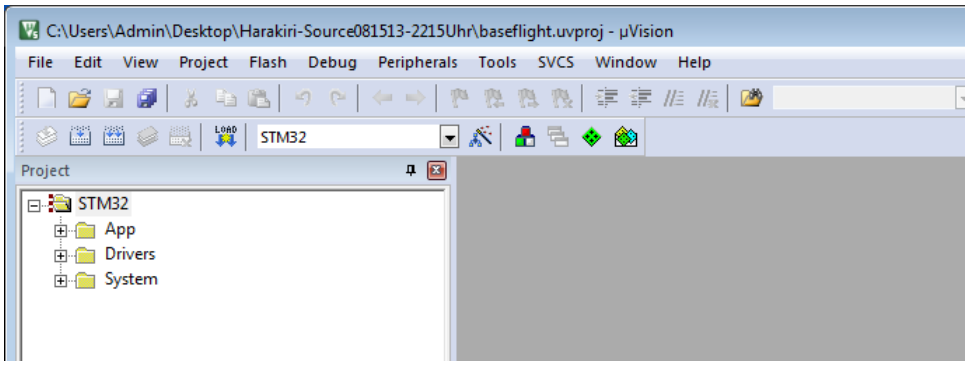
5. Öffnen des Projektfiles / Opening the project file

Timecop verwendet für die Baseflight Entwicklung ebenfalls die KEIL MDK IDE, weshalb er auch gleich ein passendes Projektfile mitliefert.

Timecop is using the Keil MDK IDE for code development too. Because of that, a suitable project file is enclosed with the sources.

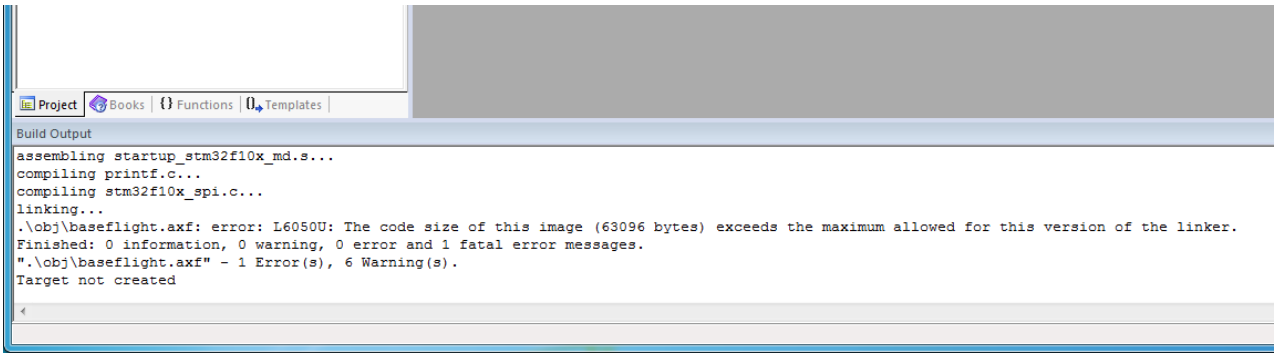
Mit einem Doppelklick auf die Datei *BASEFLIGHT.UVPROJ* wird dieses geöffnet. In der µVision IDE sieht das dann wie folgt aus:

Double-clicking the *BASEFLIGHT.UVPROJ* opens the IDE which is now looking like:



Wenn das Projekt schon mal offen ist kann ein BUILD (F7) nicht schaden!
Da aktuell jedoch noch der auf 32kB limitierte Compiler verwendet wird endet dieser Versuch mit einer Fehlermeldung.

With the project already open, why not try BUILDING it (F7)?
Since the toolchain has a limited codesize of 32kB this attempt will present you the error message shown.

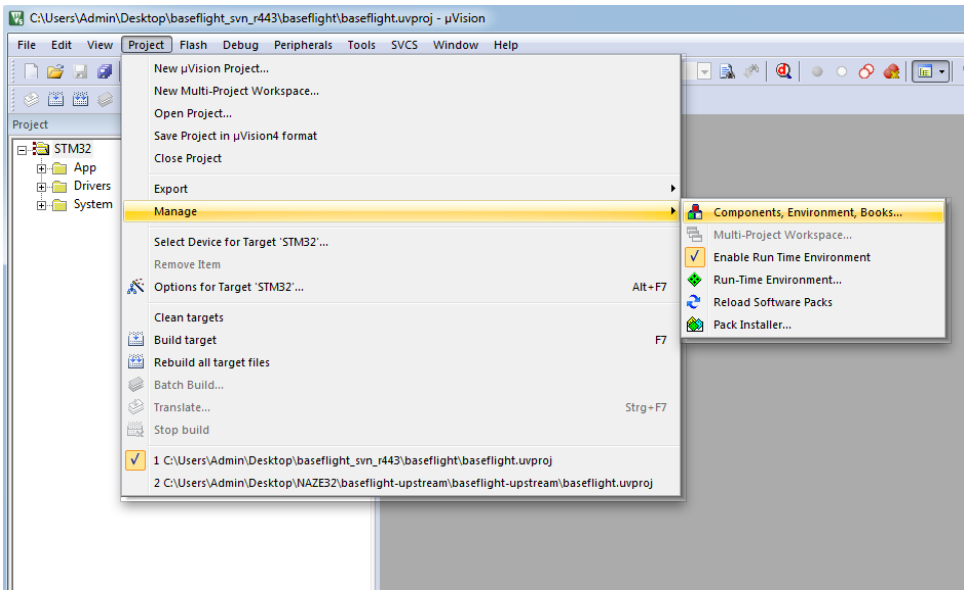


6. Aufheben der 32KB Limitierung / Breaking the 32kB limit

Klicken Sie auf

Click on

PROJECT-->MANAGE-->COMPONENTS,ENVIRONMENT,BOOKS...



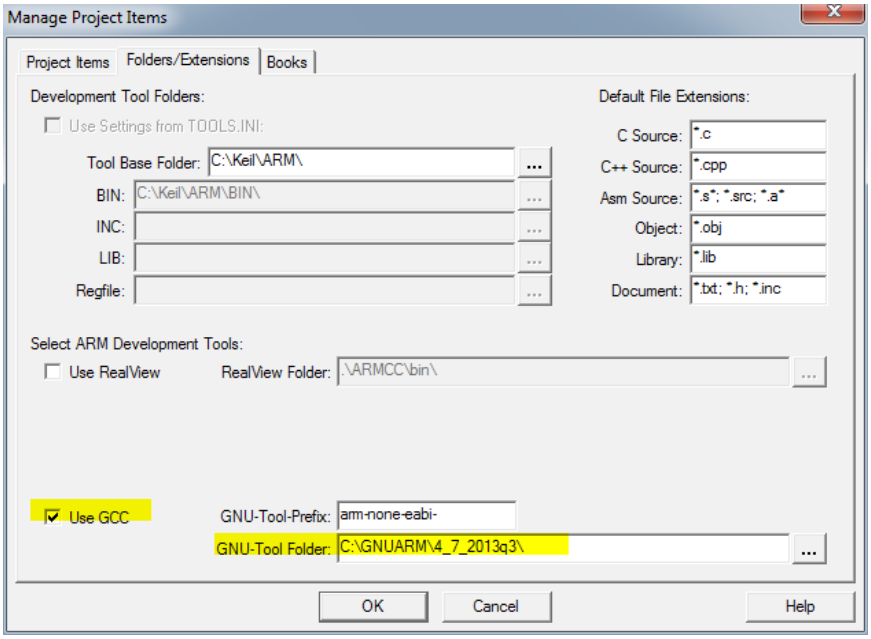
Klicken Sie jetzt auf den Reiter

Now click the tab

FOLDERS/EXTENSIONS

und aktivieren sie die `USE GCC` Checkbox.

and activate the `USE GCC` checkbox.



Es erscheint eine Warnung dass alle Project-Settings verloren gehen. Diese wird mit YES bestätigt.
Jetzt wird der Pfad zur ARM GCC EMBEDDED Installation (aus 1.) eingetragen. Mit OK bestätigen.

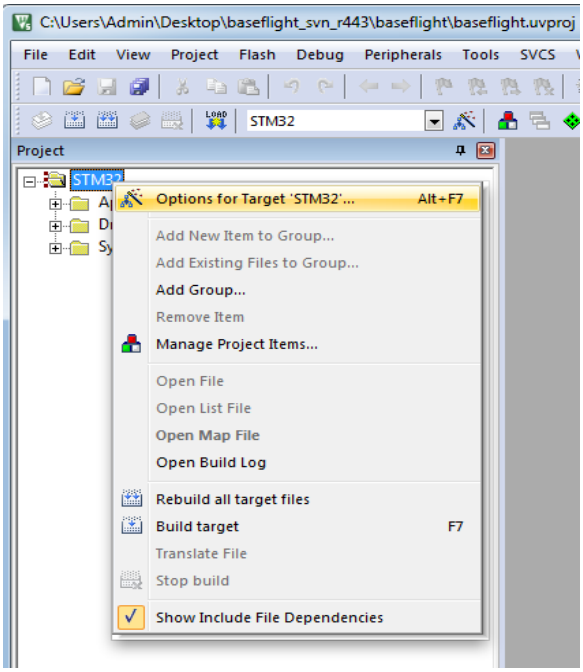
A warning message informs you that all project setting will be lost. Click YES to proceed.
Now edit the path to match your ARM GCC toolchain installation (refer to step 2.). Click OK to finish the dialog.

Im nächsten Schritt müssen nun die Project-Settings angepasst werden damit mit der ARM GCC Toolchain compiliert/gelinked werden kann.

Rechts auf `STM32` klicken und `OPTIONS FOR TARGET STM32` auswählen (alternativ über ALT+F7).

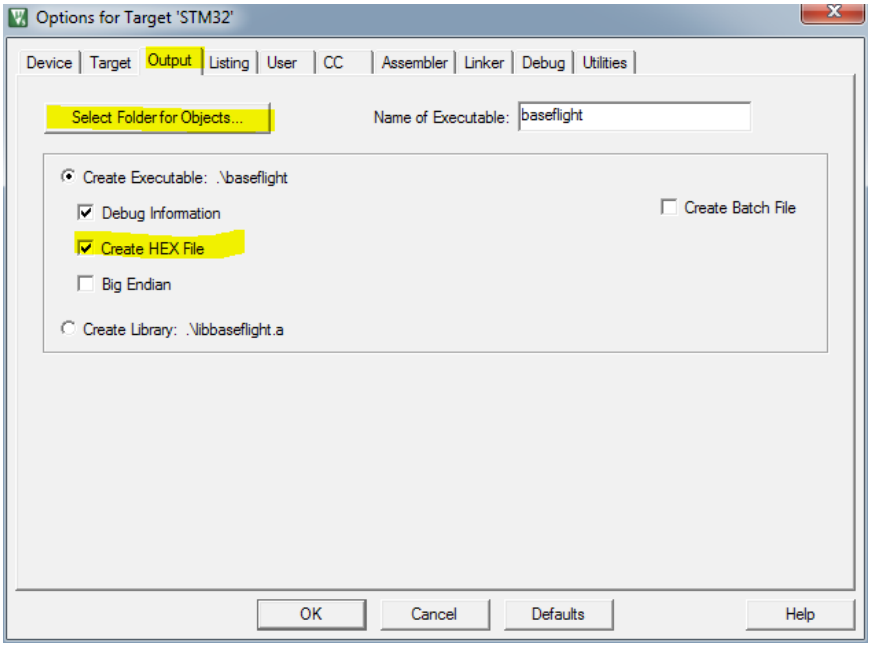
Now we need to make the projects settings to be able to use the ARM GCC toolchain for compiling/linking.

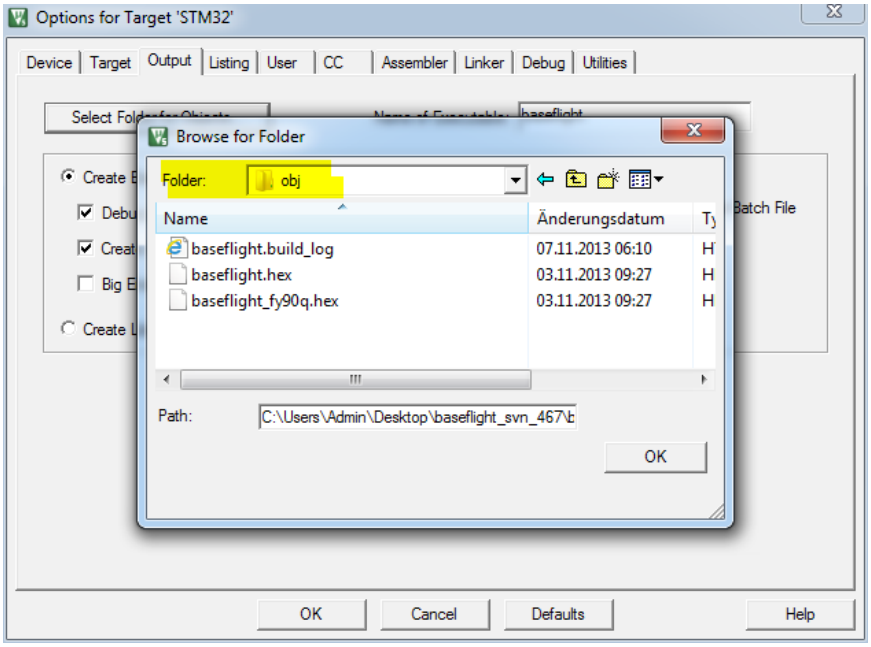
Right-click on `STM32` and click on `OPTIONS FOR TARGET STM32` (alternativly, use ALT+F7)



Auf den Reiter `OUTPUT` klicken, die Checkbox für die Erstellung des .HEX-Files anwählen, und über `SELECT FOLDER FOR OBJECTS` den Dateipfad auf `'obj'` umstellen und bestätigen.

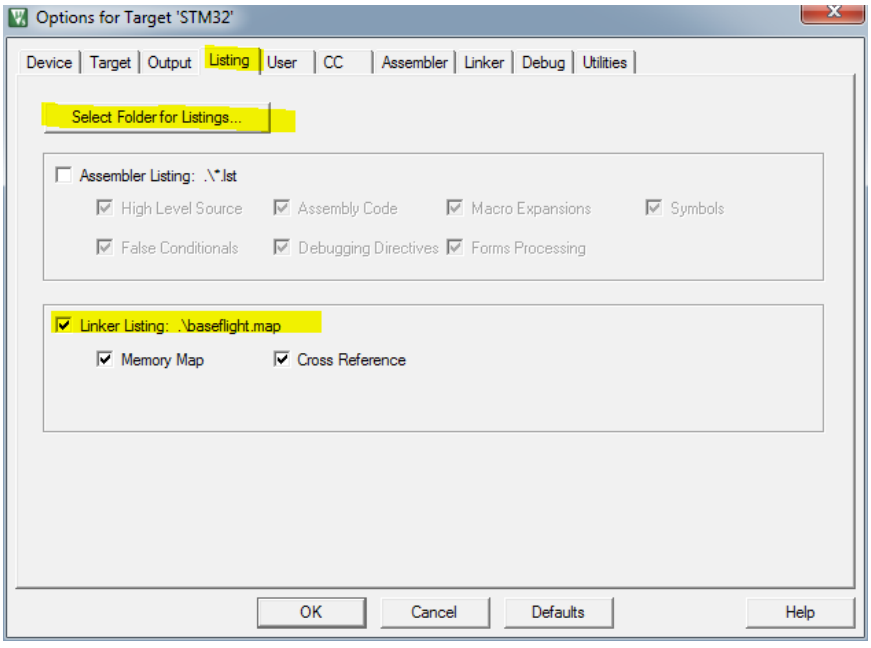
Click the `OUTPUT` tab, activate the checkbox to create the .HEX-file, click on the `SELECT FOLDER FOR OBJECTS` buttons and change the path to `'obj'`.

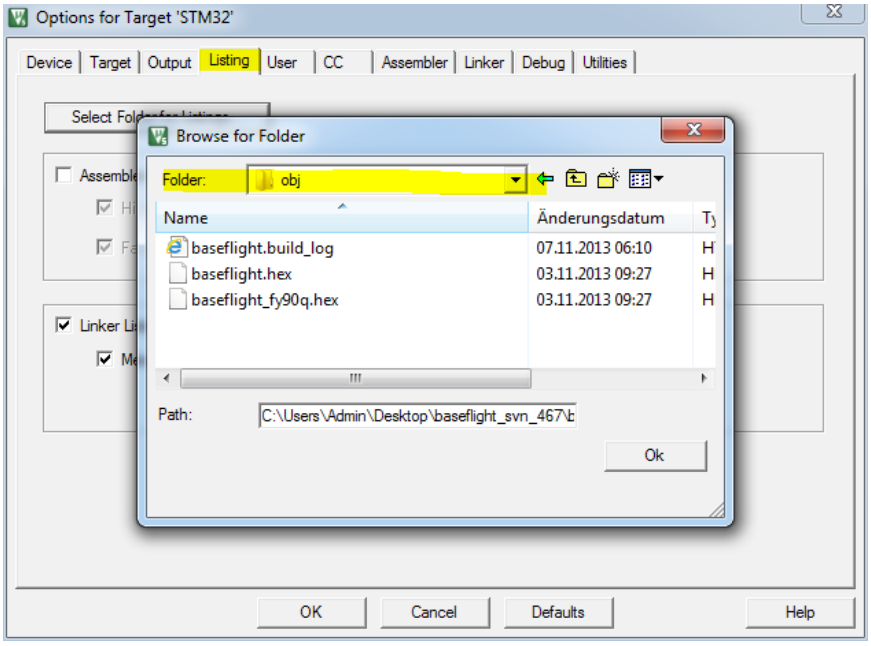




Im `LISTING` Reiter wird die Erstellung des .MAP-Files aktiviert und als Dateipfad ebenfalls der `'obj'`-Folder ausgewählt.

Click the `LISTING` tab, activate the checkbox to create the .MAP-file, click the `SELECT FOLDER FOR LISTINGS` button and change the path to `'obj'`.





Weiter geht es mit dem `CC` Reiter – den Compilereinstellungen. Hier werden folgende Änderungen gemacht:

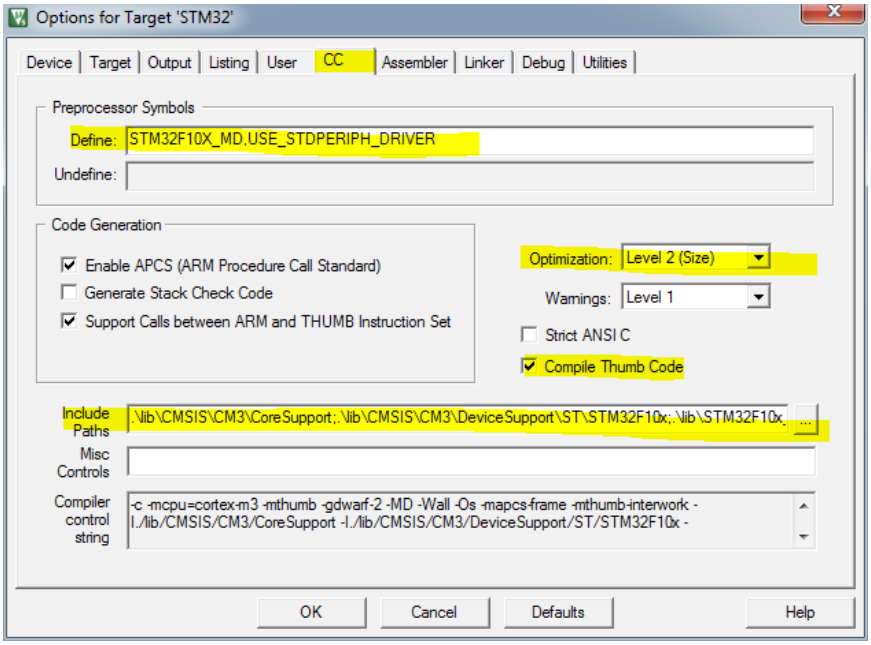
Next comes the `CC` tab – the compiler settings. You need to make the following changes:

Preprocessor Symbols: `STM32F10X_MD,USE_STDPERIPH_DRIVER`

Optimization Level: `LEVEL 2 (SIZE)`

Include Paths:

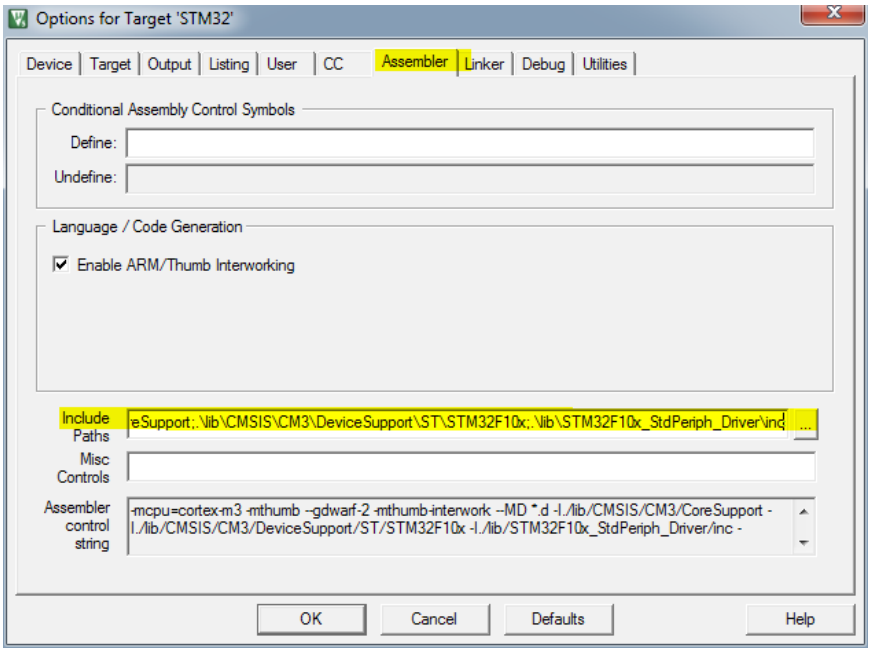
```
.\lib\CMSIS\CM3\CoreSupport;
.\lib\CMSIS\CM3\DeviceSupport\ST\STM32F10x;
.\lib\STM32F10x_StdPeriph_Driver\inc
```



Im ASSEMBLER Reiter werden ebenfalls die INCLUDE PATHS wie folgt eingetragen:

On the ASSEMBLER tab add the INCLUDE PATHS as follows:

```
.\lib\CMSIS\CM3\CoreSupport;
.\lib\CMSIS\CM3\DeviceSupport\ST\STM32F10x;
.\lib\STM32F10x_StdPeriph_Driver\inc
```

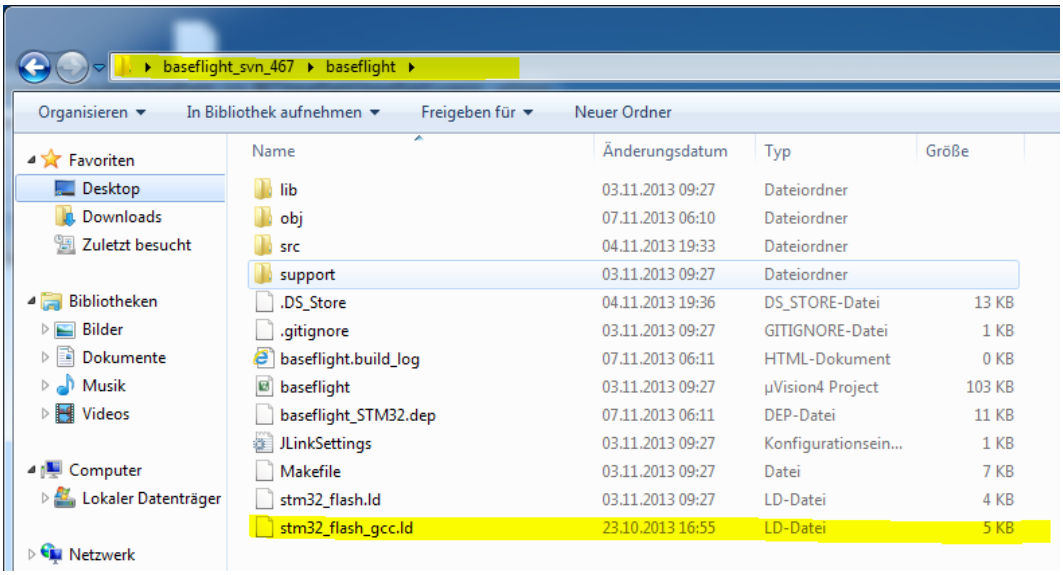


Jetzt folgt noch der LINKER Reiter.

Um hier die Einstellungen vornehmen zu können wird die beigefügte Datei stm32_flash_gcc.ld benötigt. Es handelt sich dabei um das Linker-Script-File für den ARM GCC Linker.
Die Datei wird einfach in das Baseflight-Verzeichnis kopiert.

Now proceed with the LINKER tab.

Here you will need the enclosed file stm32_flash_gcc.ld. It is the linker-script-file for use with the ARM GCC linker. Copy it to your Baseflight folder first.



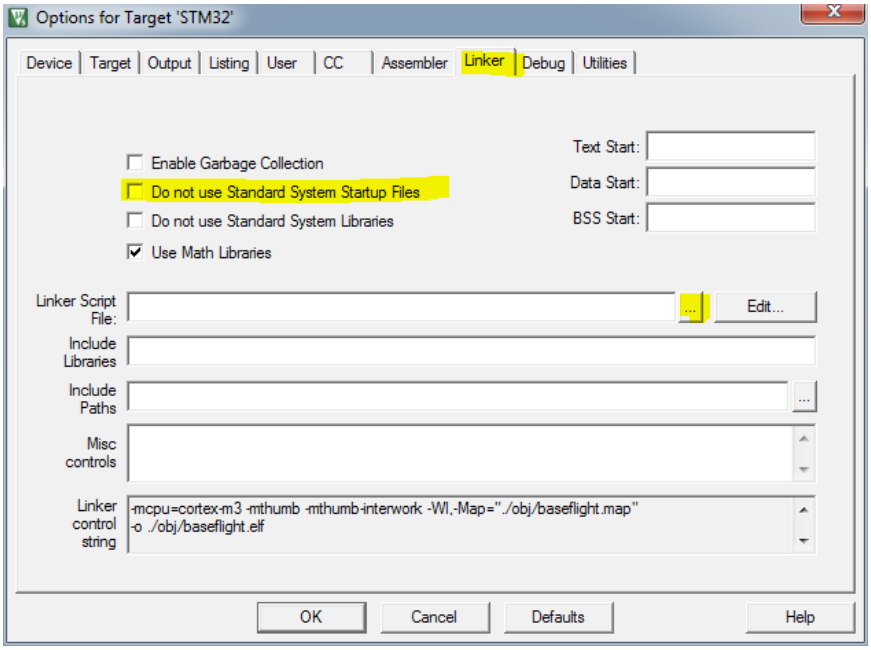
Nun werden die folgenden Einstellungen gemacht:

Now you need to make the following changes:

DO NOT USE STANDARD SYSTEM STARTUP FILES

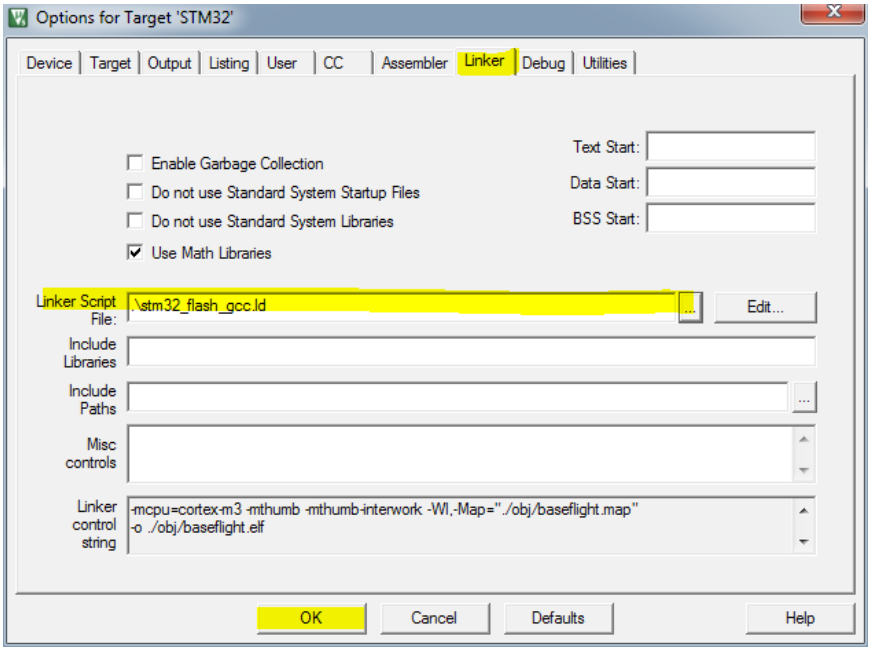
Diese Checkbox deavtivieren.

Deactivate this checkbox.



Über den Browse-Button wird das vorher hinzugefügte Linker-Script-File ausgewählt.

Use the browse-button to navigate to your Baseflight directory and select the linker-script-file.



Durch einen click auf OK werden alle gemachten Einstellungen übernommen.

Now click OK to accept all of the above changes.

8. Modifizieren des Projekts / Altering the project

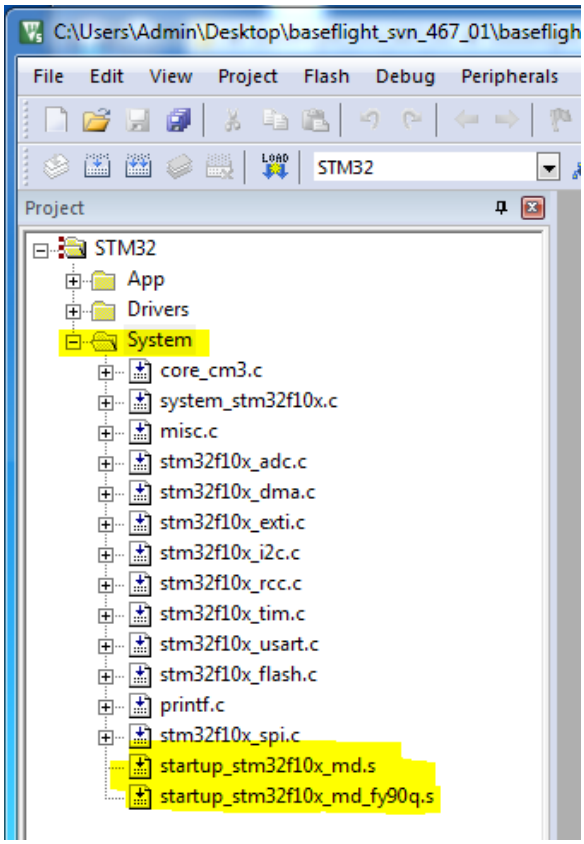
Nachdem nun alle Einstellungen zur Verwendung des ARM GCC Toolchain gemacht wurden muss das Projekt noch leicht angepasst werden um erfolgreich compiliert/gelinkt werden zu können.

Now that all setting for using the ARM GCC toolchain were made the projects needs some little tweaking to make is successfully compile/link.

8.1 Startup-File

Da ein anderer Compiler verwendet werden soll muss ein passendes Startup-File verwendet werden. Dazu wird der SYSTEM-Folder aufgeklappt und die beiden vorhandenen Startup-Files `startup_stm32f10x_md.s` sowie `startup_stm32f10x_md_fy90q.s` werden vom Projekt entfernt. Die erfolgt durch Rechts-Click auf die entsprechende Datei → im Kontext-Menü dann auf '`REMOVE FILE ..`' clicken.

Since a different compiler will be used a suitable startup-file is needed. Expand the SYSTEM-folder and remove the existing startup-files `startup_stm32f10x_md.s` and `startup_stm32f10x_md_fy90q.s` from the project. This is done by right-clicking on the file and selcting '`REMOVE FILE ..`' from the context menu.

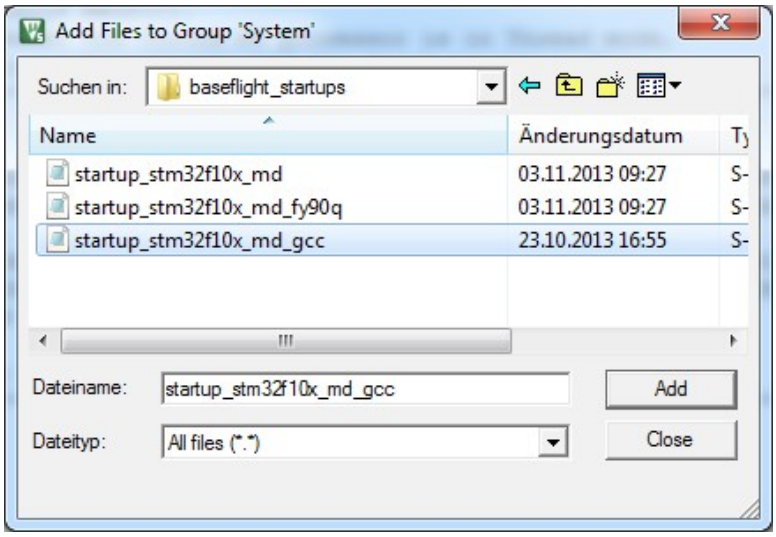


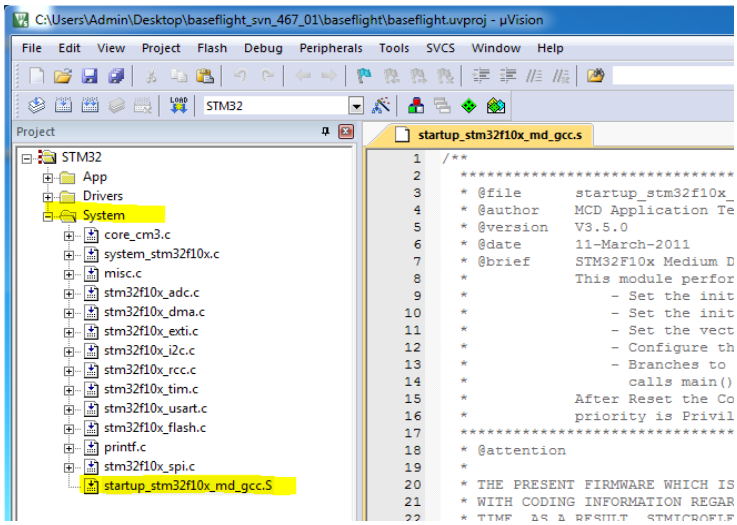
Das neue Startup-File ist die Datei `startup_stm32f10x_md_gcc.s` welche ebenfalls diesem 'How-To' beigefügt ist. Sie wird in den `SRC->BASEFLIGHT_STARTUPS` Ordner des Baseflight Projektordners kopiert. Dort befindet sich bereits eine Datei mit diesem Namen welche durch die neue Version ersetzt werden muss!

The new startup-file is `startup_stm32f10x_md_gcc.s` which is part of this 'How-to'.
It needs to be copied into the `SRC->BASEFLIGHT_STARTUPS` folder of the Baseflight Project. There is a file with this name already which needs to be replaced by the new version.

Nun per Rechtsklick auf den SYSTEM-Folder das Kontext-Menü öffnen und auf `ADD EXISTING FILES TO GROUP 'SYSTEM'` … clicken. In dem sich öffnenden Browserfenster navigiert man in den `BASEFLIGHT_STARTUPS` Ordner und wählt dort das neue Startup-File aus (den Datei-Typ auf 'All files (*.*)' umstellen, sonst werden keine Dateien angezeigt).

Now right-click the SYSTEM-folder to open the context menu and click on `ADD EXISTING FILES TO GROUP 'SYSTEM'` … . A browser window opens, navigate to the `BASEFLIGHT_STARTUPS` folder and select the new startup-file to be added (don't miss to change the file type to 'All files (*.*)').





8.2. errno.h Header-File

Jetzt wird der `APP` Ordener aufgeklappt und das Header-File `BOARD.H` geöffnet.

Expand the APP-folder and open the header file `BOARD.H`.

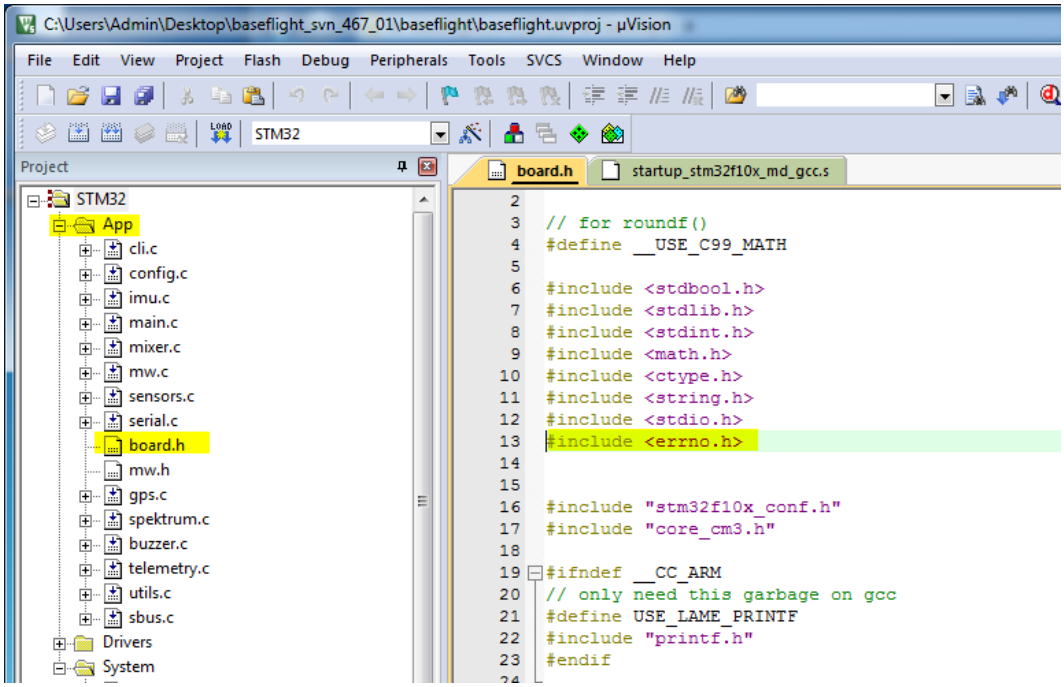
Der Befehl

```
#include <errno.h>
```

wird nach den vorhandenen include-Anweisungen hinzugefügt.

The instruction

needs to be added after the existing include instructions



8.3. __errno und / and _exit Symbol

Damit das Projekt erfolgreich compiliert und gelinkt werden kann werden zwei zusätzliche Symbole benötigt.
Es handelt sich dabei um `__errno` und `_exit`.

To make the project compile and link successfully two new symbols needs to be added.
The new symbols are `__errno` and `_exit`.

Im `APP` Folder wird die Datei `MW.C` geöffnet und der nachstehende Code wird hinzugefügt (z.B. als erste Funktion im Source, vor 'blinkLED(...)').

Open the file `MW.C` from the `APP` folder and add the code below to it (i.e. In front of the first function, blinkLED(...)).

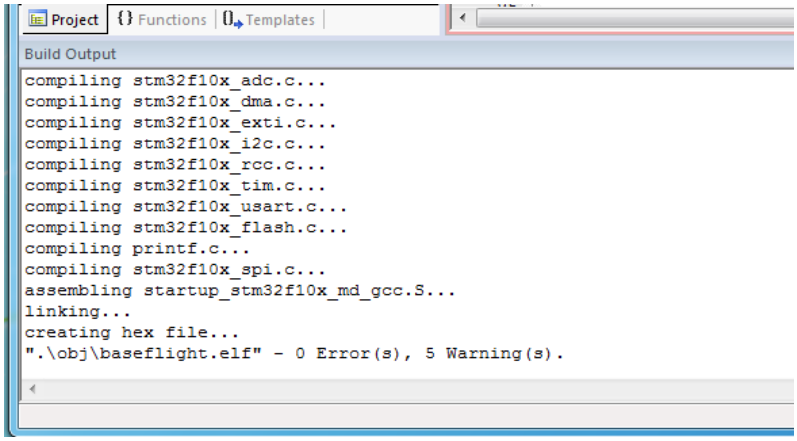
```
// aBUGSworstnightmare, changed on 10/20/2913
// __errno and _exit symbols were needed to compile/link the project under Keil+GCC

// add '#include <errno.h>' to 'board.h'
// There is no default implementation of __errno. Keeping it out of the library
// means that it's possible to customize it's behavior whitout rebuilding the library.
// This is just one possible default implementation
int *__errno (void) {return &errno;}

// exit is defined in 'stdlib.h'
// exit (usually) terminates the calling process, returns to the startup code
// and performes the appropriate cleanup process.
// This implementation is a simple trap and must be terminated by reset!
void _exit(int exit_code)
{
	while(1)
	{
		// Loop until reset
	}
}
// END OF NEW SYMBOLS
```

Jetzt auf REBUILD clicken und die Toolchain Arbeit verrichten lassen.
Das Projekt sollte ohne Fehler compiliert und gelinkt werden. Lediglich einige Warnings werden ausgegeben.

Now click on REBUILD and let the toolchain do it's job.
The project should be compiled and linked successfully. Only some warnings will be shown.



TODO:

Debugging

SVN Checkout