OST Ostschweizer Fachhochschule

Code Composer™ Studio

CCS6_Readme

Version 1.02

Inhaltsverzeichnis

1. Arbeitsverzeichnisse 2
2. Erstellen eines Projektes im Code Composer Studio v6 2
3. Import eines bestehenden Projektes 3
4. Erstellen eines (weiteren) Target Configuration Files 4
5. C-Projekt mit Inline-Assembler Befehlen 4
6. Compiler Konfiguration für C I/O-Ausgaben 5
6.1. Code Beispiel 6
7. Benützung der StellarisWare 7
7.1. Pfad der StellarisWare definieren 7
7.2. Zusätzlich notwendige Einstellungen für den Compiler 7
7.3. Zusätzlich notwendige Einstellungen für den Linker 8
8. Benützung der TivaWare 10
9. CMSIS 10
9.1. CMSIS in CCS Projekt einbinden 10
9.1.1. Zusätzlich notwendige Einstellungen für den Compiler 10
10. Fehlermeldungen / Hints / Tipps 12
10.1. Code Editierung 12

	Verwendete Tool-Versionen:	*Verwendete Installationspfade:*
Code Composer Studio	*6.1.3.00034*	*C:\ti\ (default)*
MSP430 Compiler CCS	*TI v15.12.1 LTS*	
ARM Compiler CCS	*TI v15.12.1 LTS*	
MSPWare	*2.21.00.39*	*C:\ti\msp (default)*
TivaWare	*2.1.1.71*	*C:\ti\TivaWare_C_Series-2.1.1.71*
StellarisWare	*9107*	*C:\StellarisWare*
CMSIS	*Core: V3.01* *TI: V8636*	*C:\Apps\TI_CMSIS*

1. Arbeitsverzeichnisse

Ausgangsdaten: C:\Student\... z.B. C:\Student\ComEng\

Workspace: U:\... z.B. U:\ComEng2\CCS_Workspace\

2. Erstellen eines Projektes im Code Composer Studio v6

1. File / New / CCS Project

Folgendes Fenster erscheint:

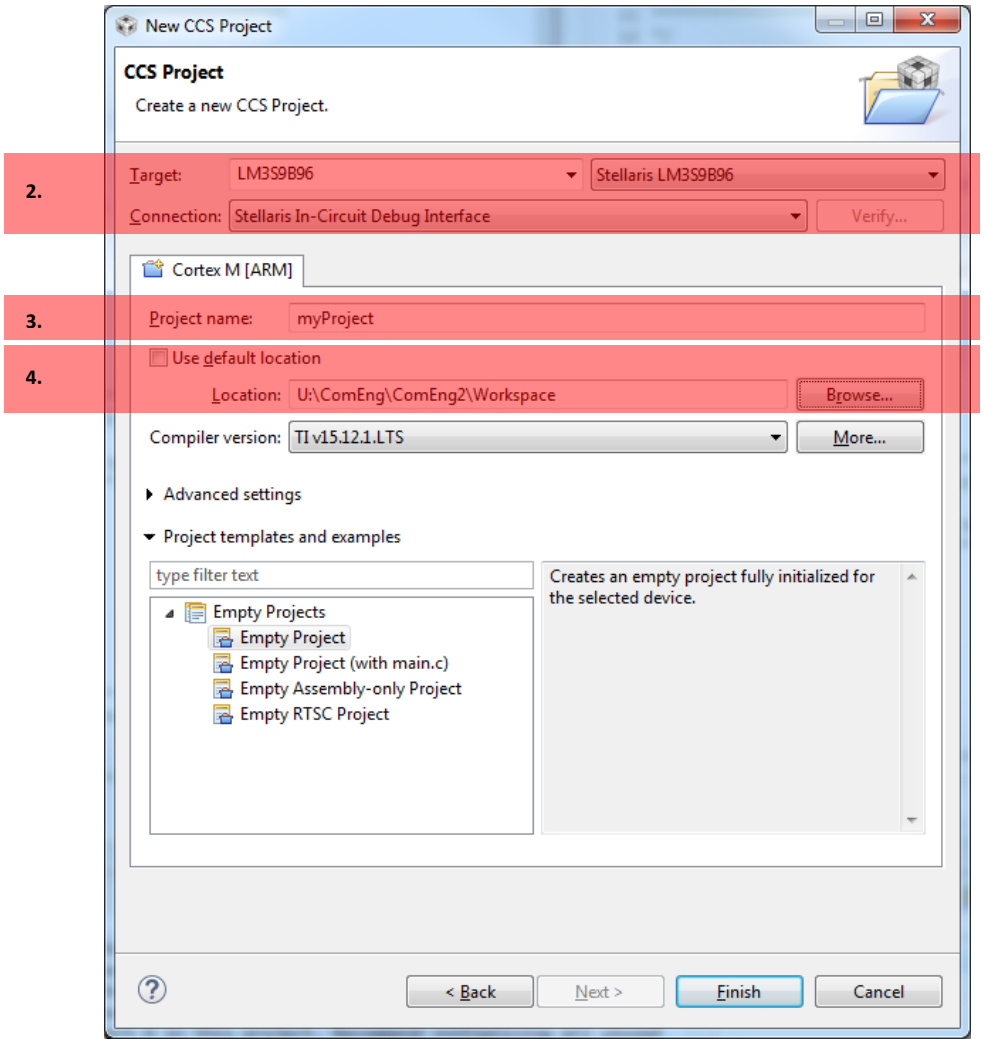

Abbildung 1 CCS Projekt erstellen

2. Device Einstellungen vornehmen

1. Target: LM3S9B96

2. Connection: Stellaris In-Circuit Debug Interface

Hinweis: Den Texas Instruments Simulator gibt es nicht mehr aber Code Composer Studio v6 und neuer

3. Projektname eingeben

4. Projektpfad einstellen

U:\....

Hinweis: Die Entwicklungsumgebung kann nicht mit UNC-Pfaden umgehen (z.B. \\c101.hsr.ch\<userName>) , solche Pfade können nur genutzt werden, wenn sie als Laufwerk hinzugefügt sind (z.B. als U:)

5. Projekt Template wählen

6. Finish

Erzeugt Projekt, inklusive TargetFile (abhängig von der gewählten Verbindung)

3. Import eines bestehenden Projektes

File / Import…

1. Code Composer Studio: CCS Projects auswählen → Next

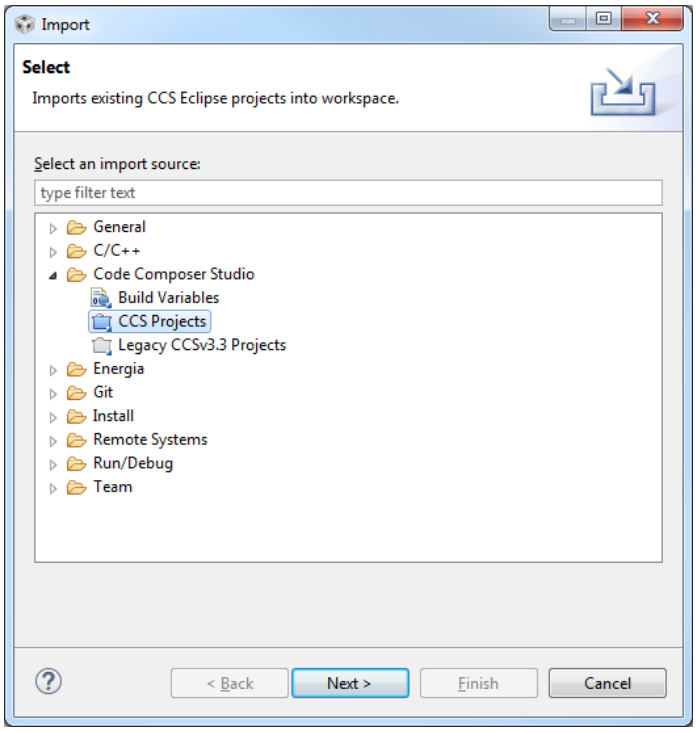

Abbildung 2 Import eines bestehenden Projektes

2. Verzeichnis auswählen, wo das/die zu importierende(n) Projekte sind
   Beispiel: C:\Student\ComEng\
3. Das Projekt in den Workspace kopieren → Checkbox markieren!

→ Finish

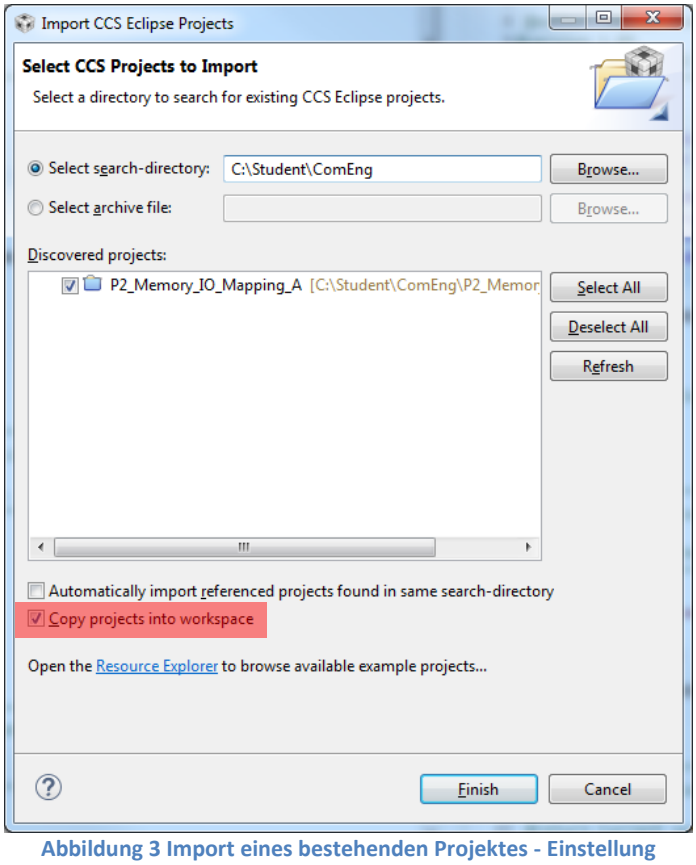

Abbildung 3 Import eines bestehenden Projektes - Einstellung

4. Erstellen eines (weiteren) Target Configuration Files

1. File / New / Target Configuration File (wird im aktiven Projekt erzeugt)

2. Name definieren (z.B. XDS100_v2), mit Finish weiter

3. Verbindung und Device auswählen

Beispiel: XDS100_v2 mit LM3S9B96 Device

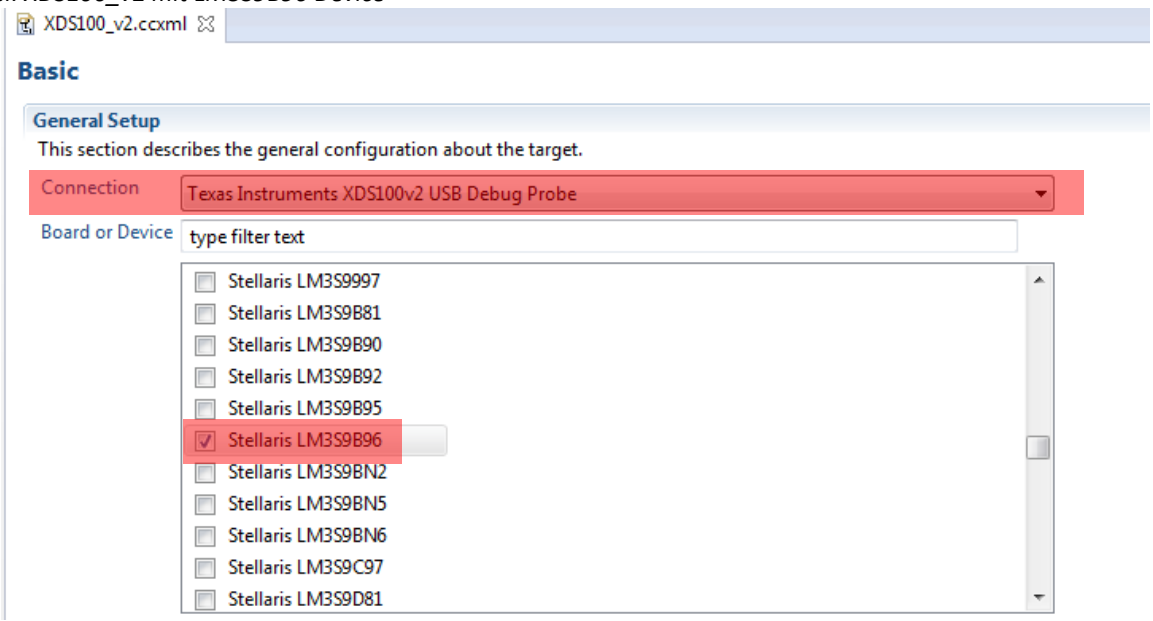

Abbildung 4 Target Configuration File - Einstellungen

5. C-Projekt mit Inline-Assembler Befehlen

Beispiel: Ausgabe von 0x07 auf dem LED-Port (Adresse 0x6E00'0000).

```
//local assembler calls (Write 0x07 to LEDs)
//Style is __asm("Label Instruction");
__asm(" MOV    R0, #0x07");
__asm(" MOV.W  R1, #0x6E000000");
__asm(" STRB   R0, [R1]");
```

6. Compiler Konfiguration für C I/O-Ausgaben

1. Rechtsklick auf das Projekt / Properties / Build / Advanced Options / Library Function Assumptions

→ Entsprechendes Level für die printf/scanf Funktionen auswählen

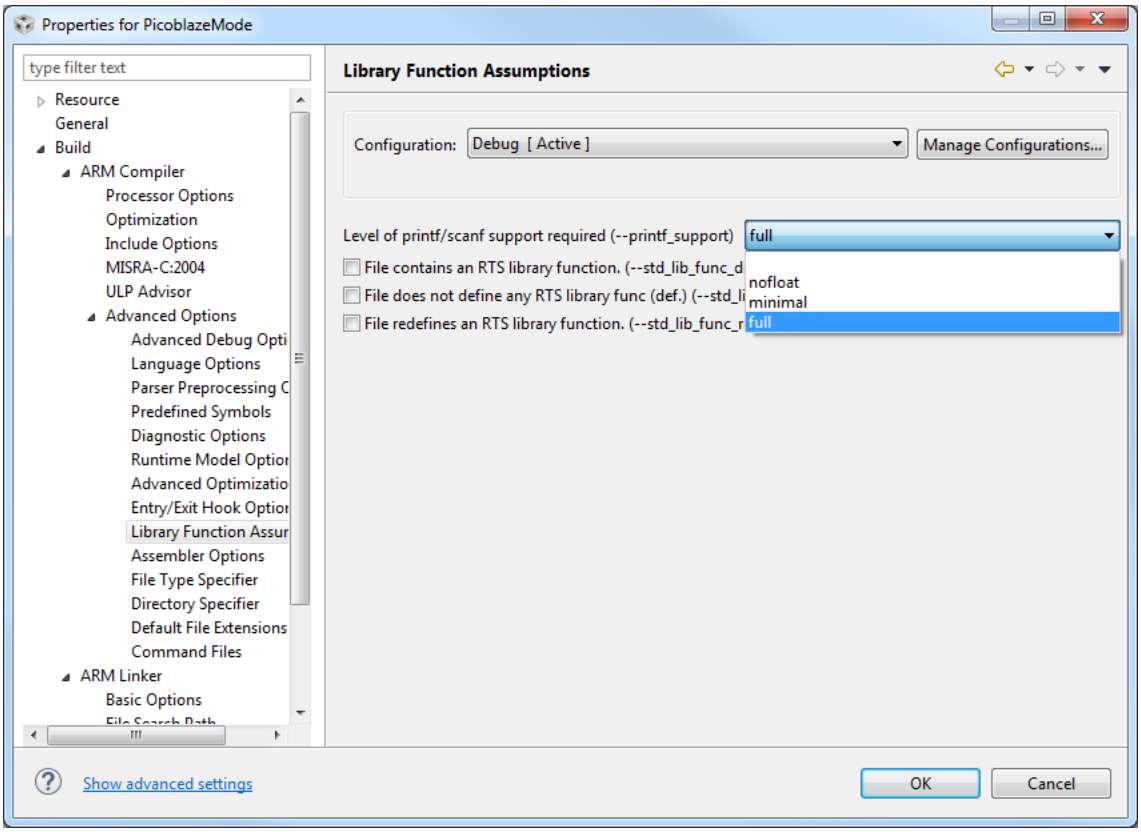

Abbildung 5 Printf/Scanf - Library Function Setup

2. Notwendige minimale Stack und Heap Grössen definieren/anpassen:

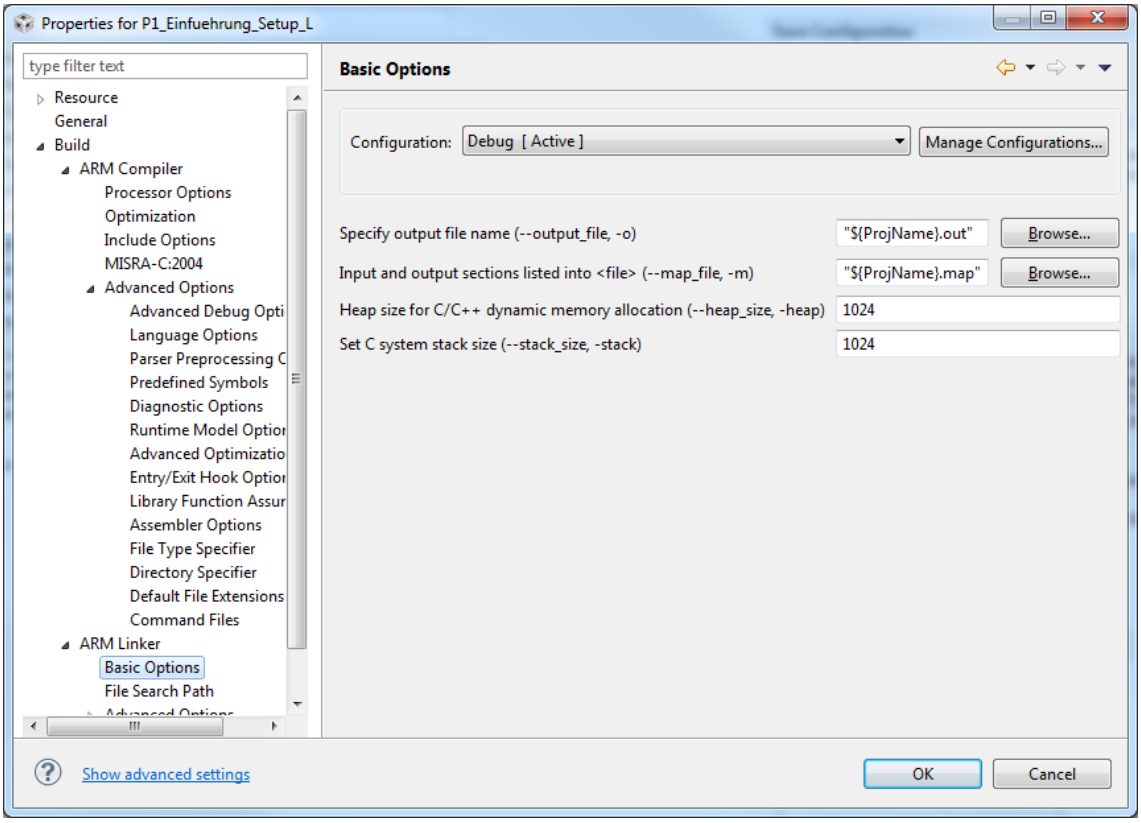

Abbildung 6 ARM Linker - Stack / Heamp Setup

Hinweis: *Die Angegebenen Speichergrössen (Byte) sind als Minimal-Werte anzusehen und sind je nach Anwendung entsprechend anzupassen.*

6.1. Code Beispiel

```
// --- Includes
#include <stdio.h>
#include <stdint.h>

// --- Local Function Prototypes

/**
*****************************************************************
@brief   Main program
@param   -
@return  -
*****************************************************************
*/
int main(void)
{
  uint32_t value = 0;
  int scanfReturn = 0;

  while(1)
  {
    printf("Your Input: ");
    fflush(stdout); // This will flush any pending printf output
    scanfReturn = scanf("%d", &value);
    printf(" InValue = %d\n", value);
    fflush(stdout); // This will flush any pending printf output
  }
}
```

Hinweis: *Der Funktionsaufruf fflush(stdout) wird desshalb gemacht, damit wenn ein Breakpoint auf die nächste Zeile gemacht wird, die Ausgabe auch sicherlich stattgefunden hat. Wird der printf(„Txt\n“); mit dem ‚\n‘ abgeschlossen, so wird auch diese Ausgabe vorgenommen.*

Hinweis: *Weitere Informationen unter http://processors.wiki.ti.com/index.php/Tips_for_using_printf.*

7. Benützung der StellarisWare

7.1. Pfad der StellarisWare definieren

Rechtsklick auf aktives Projekt / Properties / Build → Environment, neue Variable SW_ROOT hinzufügen

Name: SW_ROOT

Value: C:\StellarisWare

Hinweis: Die Variable SW_ROOT muss auf den Installationsordner der StellarisWare zeigen. Es können absolute wie auch relative Pfade verwendet werden.

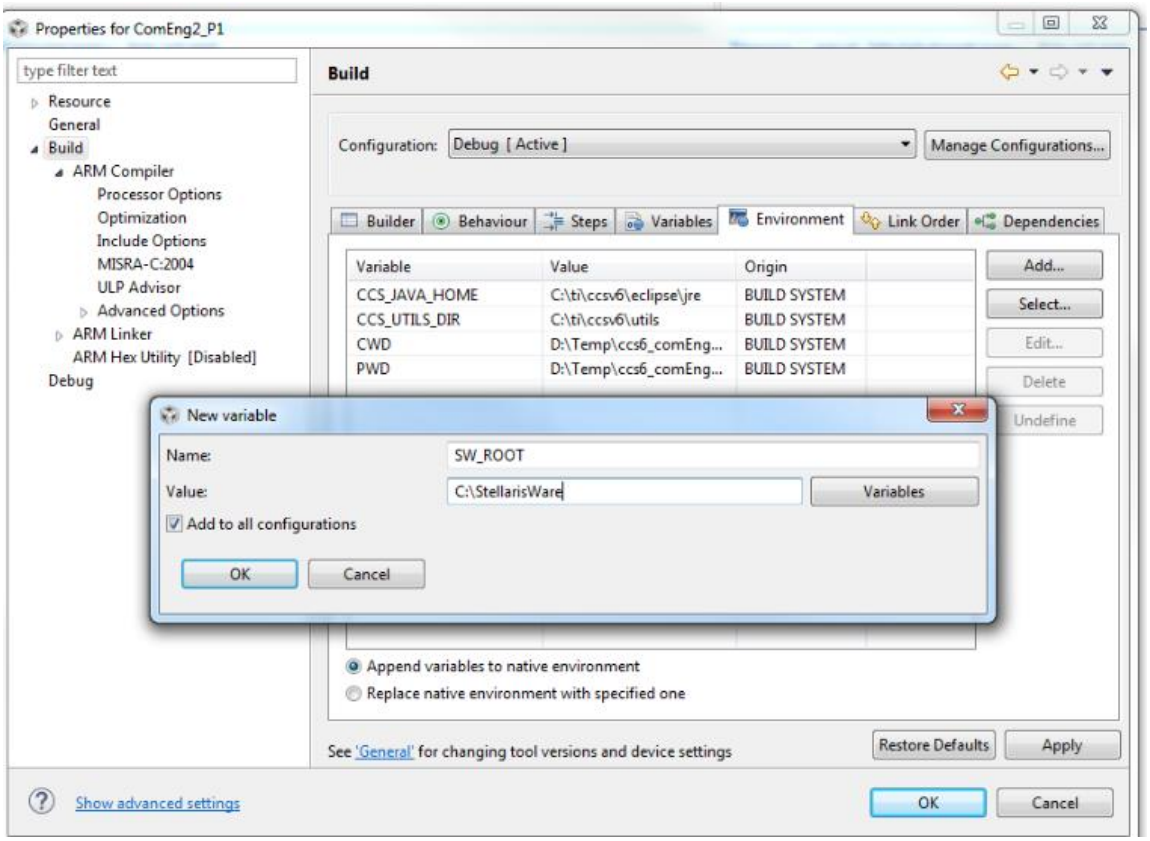

Abbildung 7 StellarisWare - Environment Variable für Pfad

7.2. Zusätzlich notwendige Einstellungen für den Compiler

Build / ARM Compiler / Include Options → Add search path

Variable "${SW_ROOT}" hinzufügen

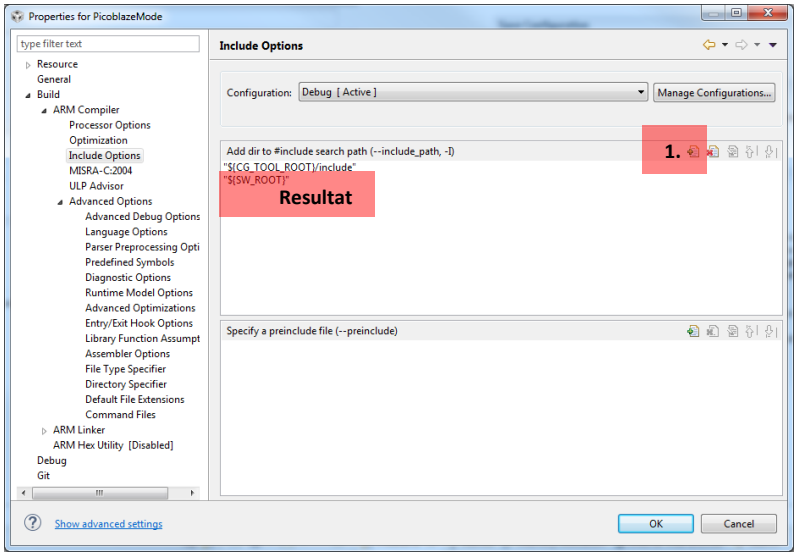

Abbildung 8 StellarisWare - Compiler Include Setup

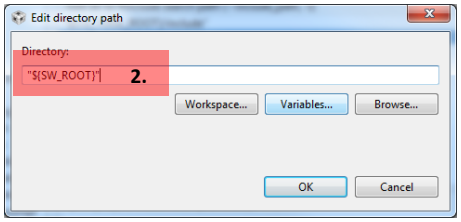

Abbildung 9 StellarisWare - Compiler Include Path

Build / ARM Compiler / Advanced Options / Predefined Symbols

Diese beiden Predefined Symbols erstellen

- ccs Definiert Code Composer Studio als IDE
- PART_LM3S9B96 Definiert den verwendeten Mikrocontroller

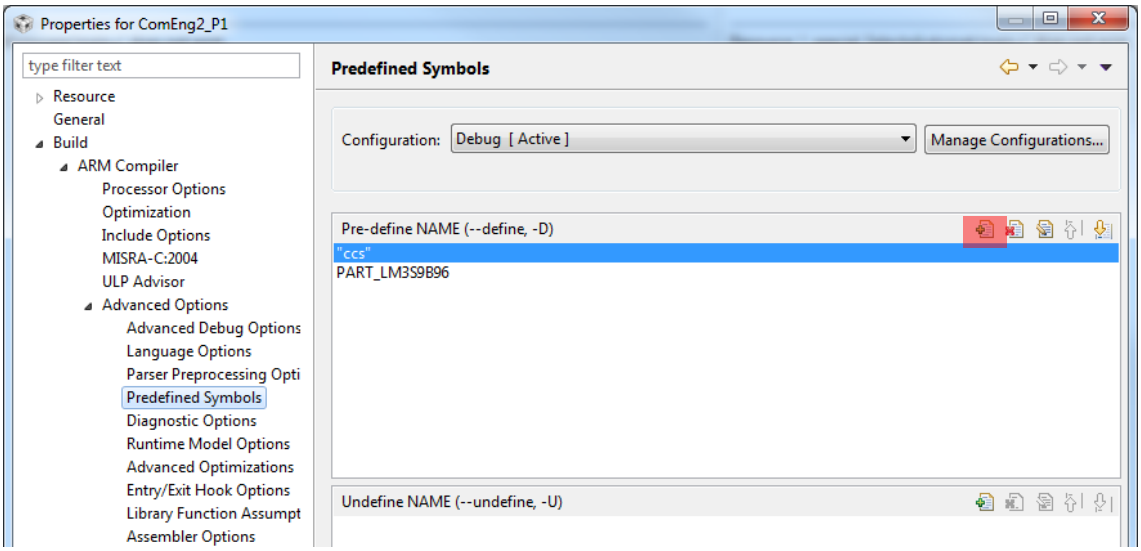

Abbildung 10 StellarisWare - ARM Compiler Predefined Symbols

Hinweis: Anstelle von PART_LM3S9B96 kann auch "PART_${DEVICE}" verwendet werden, dann wird das bei der Projekterstellung definierte Device automatisch korrekt verwendet. Ab CCS6 werden diese Symbole automatisch bei der Projekterstellung vom CodeComposerStudio generiert.

7.3. Zusätzlich notwendige Einstellungen für den Linker

1. Stack und Heap definieren

- Grösse des Heap-Speichers festlegen
- Grösse des Stack-Bereichs festlegen

Build / ARM Linker / Basic Options

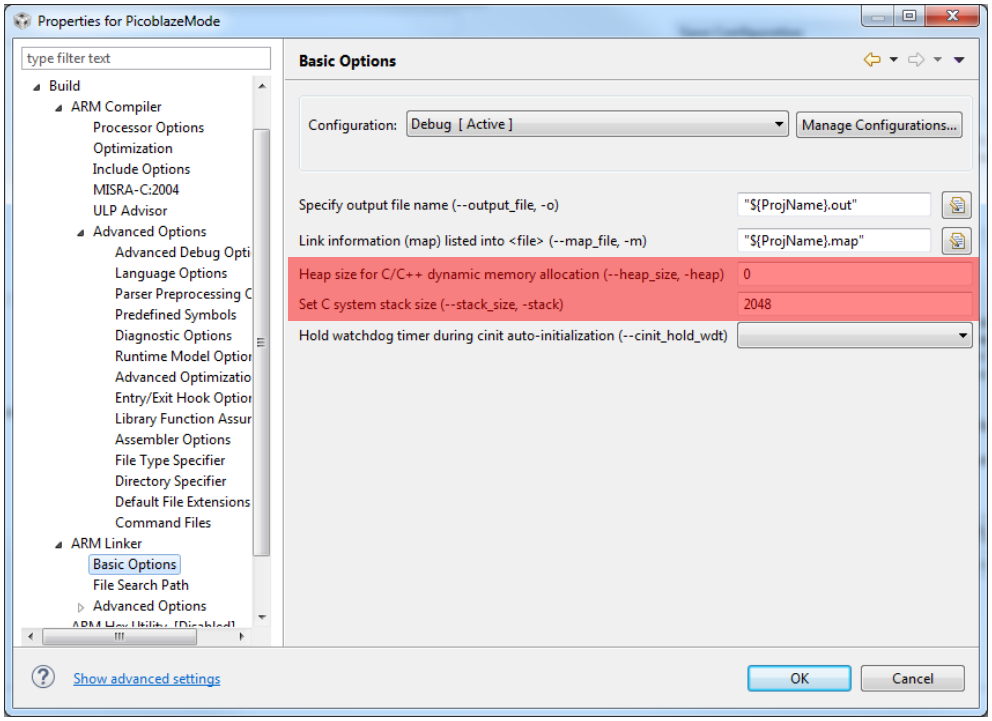

Abbildung 11 StellarisWare - Linker Heap/Stack Setup

Hinweis: Die Abgebildeten Werte sind als Beispiel anzusehen.

2. Driver-Library File hinzufügen:

Build / ARM Linker / File Search Path → Include library file or command file as input

"${SW_ROOT}\driverlib\ccs-cm3\Debug\driverlib-cm3.lib"

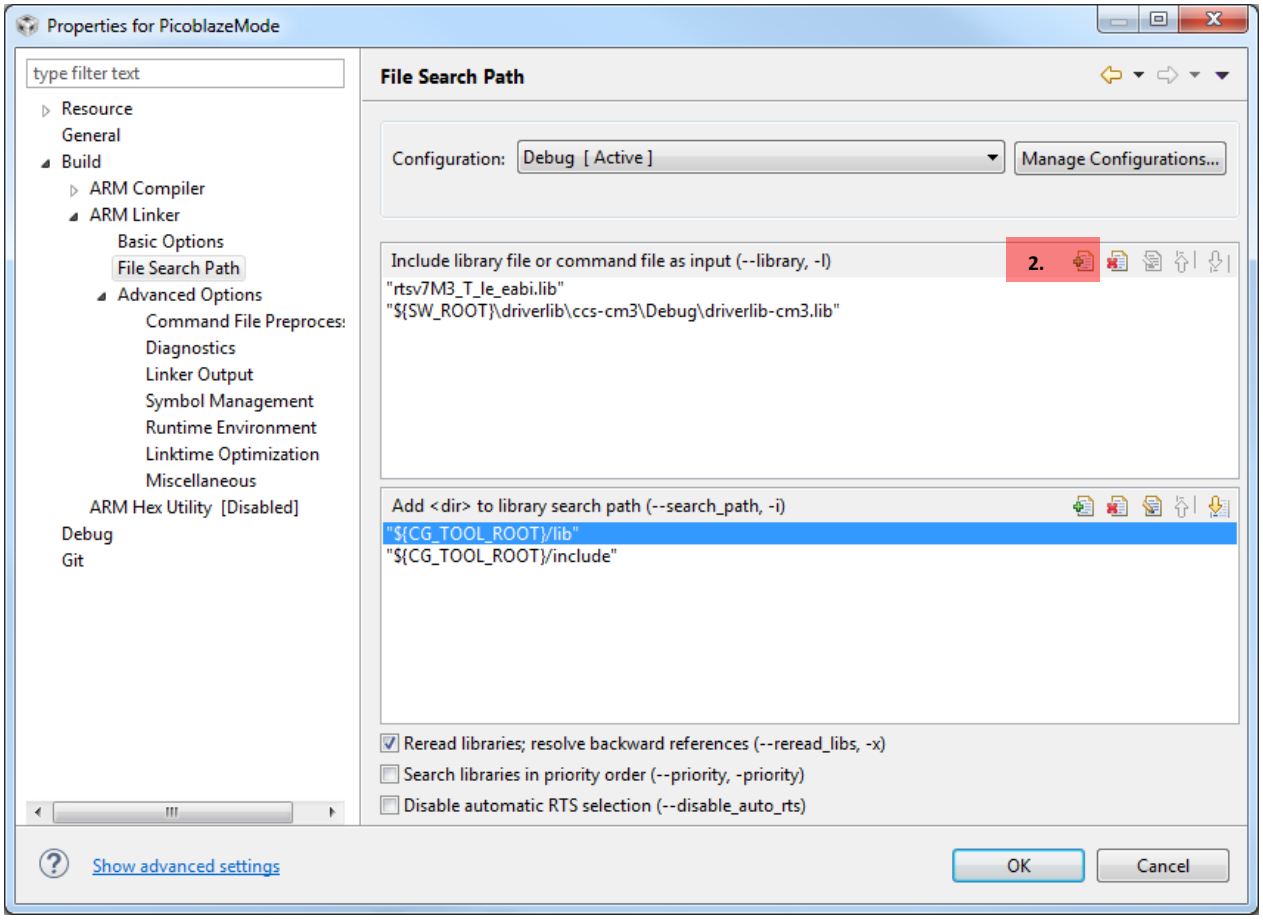

Abbildung 12 StellarisWare - ARM Linker Setup

Hinweis zu allen möglichen Libraries für CortexM3:
Driver: "${SW_ROOT}\driverlib\ccs-cm3\Debug\driverlib-cm3.lib"
Grafisch: "${SW_ROOT}\grlib\ccs-cm3\Debug\grlib-cm3.lib"
USB: "${SW_ROOT}\usblib\ccs-cm3\Debug\usblib-cm3.lib"

Hinweis zu allen möglichen Libraries für CortexM4F:
Driver: "${SW_ROOT}\driverlib\ccs-cm4f\Debug\driverlib-cm4f.lib"
Grafisch: "${SW_ROOT}\grlib\ccs-cm4f\Debug\grlib-cm4f.lib"
USB: "${SW_ROOT}\usblib\ccs-cm4f\Debug\usblib-cm4f.lib"

8. Benützung der TivaWare

Ist analog zur StellarisWare zu handhaben.

9. CMSIS

CMSIS (Cortex Microcontroller Software Interface Standard) ist eine Herstellerunabhängige Abstraktionsschicht.

9.1. CMSIS in CCS Projekt einbinden

Rechtsklick auf aktives Projekt / Properties / Build → Environment, neue Variable CMSIS_ROOT definieren

Name: CMSIS_ROOT

Value: C:\Apps\TI_CMSIS

Hinweis: Die Variable CMSIS _ROOT muss auf den TI_CMSIS Ordner zeigen. Es können absolute wie auch relative Pfade verwendet werden.

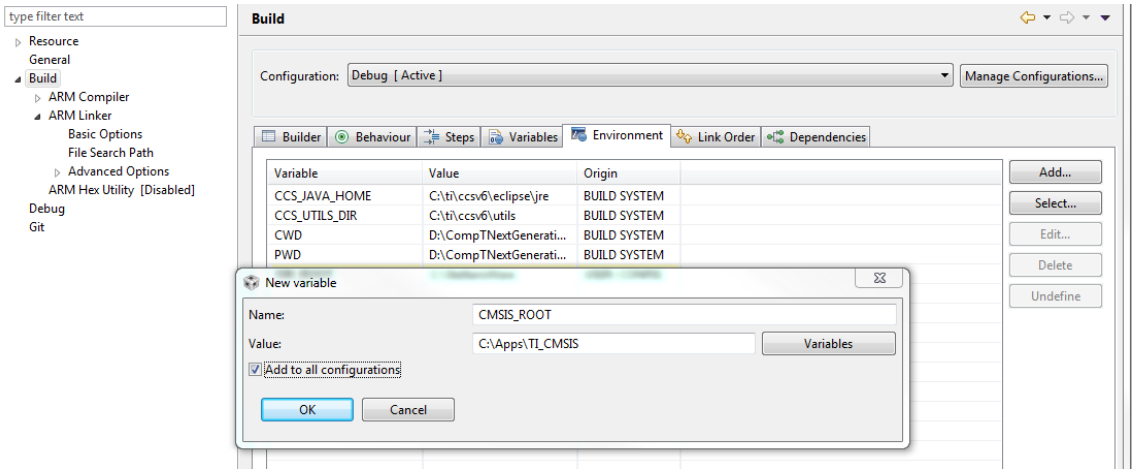

Abbildung 13 CMSIS - Environment Varible

9.1.1. Zusätzlich notwendige Einstellungen für den Compiler

Build / ARM Compiler / Include Options → Add dir to #include search path

1. "${CMSIS_ROOT}\TI\LM3S\include" hinzufügen
2. "${CMSIS_ROOT}\CORE\include" hinzufügen

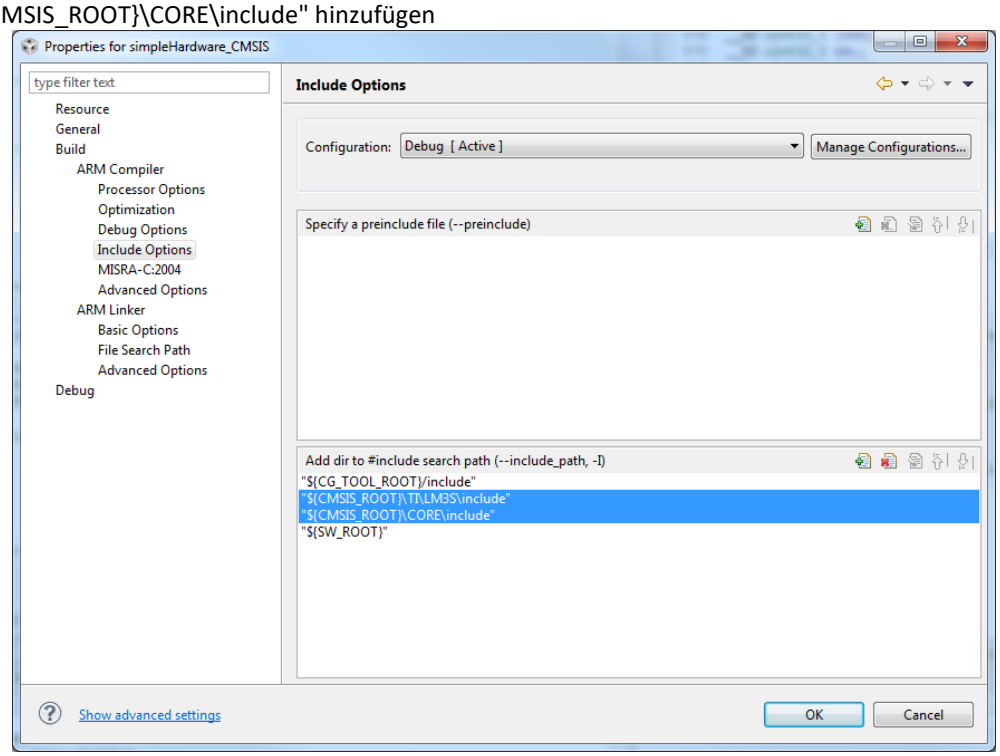

Abbildung 14 CMSIS - Compiler Include Options

Build / ARM Compiler / Advanced Options / Language Options

Enable support for GCC extensions aktivieren

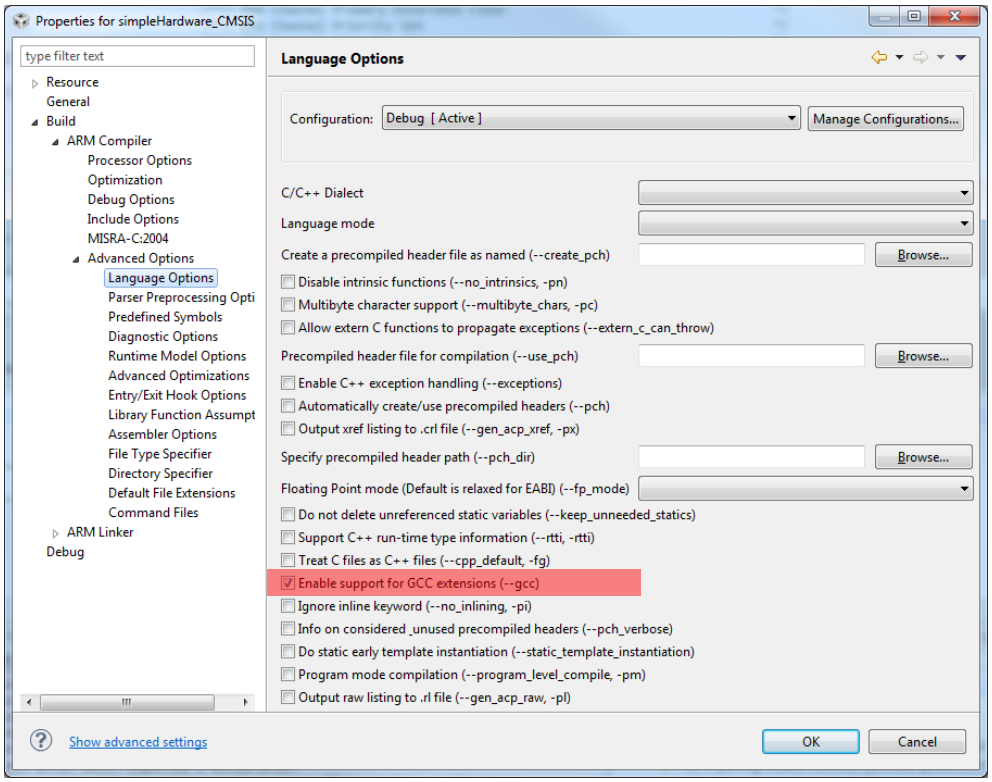

Abbildung 15 CMSIS - Enable support for GCC extensions

Hinweis: Die GCC Zusätze sind unter anderem notwendig für anonyme unions, welche zum Teil innerhalb der CMSIS Files verwendet werden.

Build / ARM Compiler / Advanced Options / Predefined Symbols

Dieses zusätzliche Symbol erstellen (Definition welcher Compiler verwendet wird)

- __TMS470__ Definiert TMS470 als Compiler

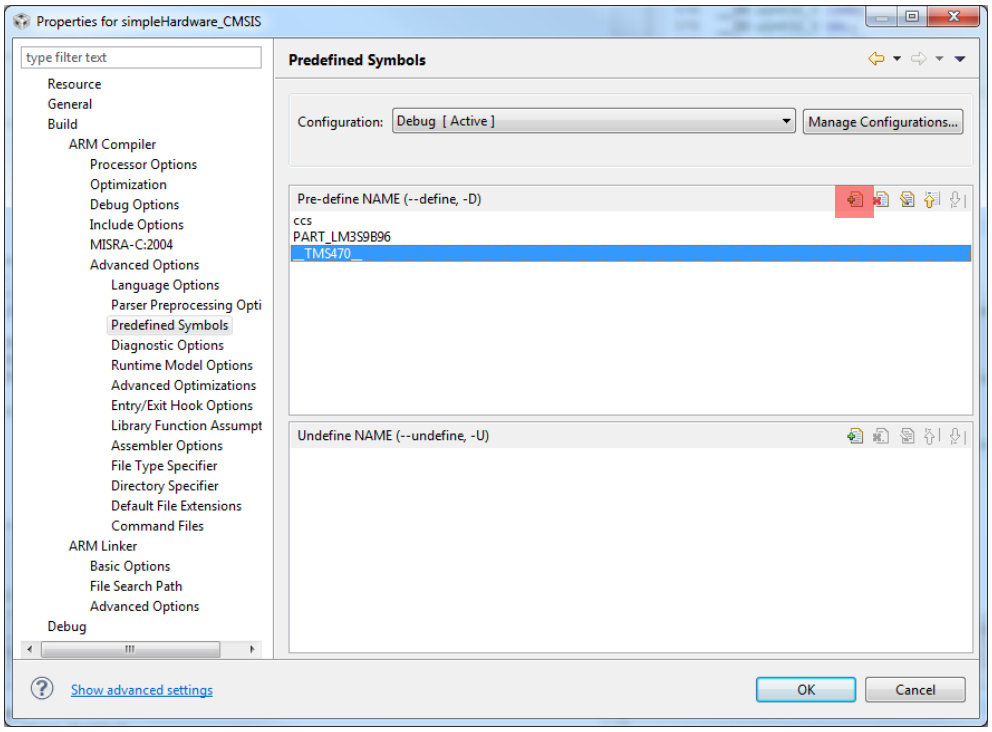

Abbildung 16 CMSIS - ARM Compiler Predefined Symbols

10. Fehlermeldungen / Hints / Tipps

10.1. Code Editierung

Falls der Code nicht "normal" editiert werden kann, könnte das Add-On "Toggle Vrapper" aktiv sein (es sieht optisch aus wie der "Insert"-Modus, ist aber ein anderer). Dieser Modus kann wie in der untenstehenden Grafik ersichtlich ist aktiviert sein und über das entsprechende Symbol deaktiviert werden.

http://vrapper.sourceforge.net/documentation/?topic=basics

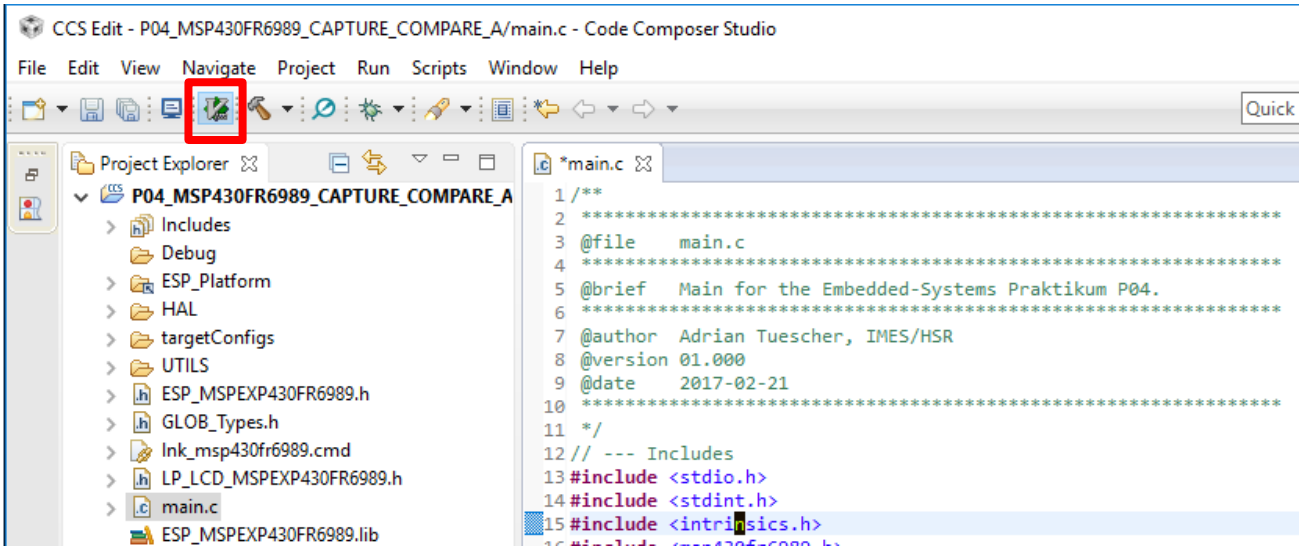

Abbildung 17 Toggle Vrapper