Hochschule **Rosenheim**
University of Applied Sciences

Fakultät für Ingenieurwissenschaften

im Studiengang Mechatronik

Bachelorarbeit

Studie zur Toolchain des modellbasierten Softwareentwurfs und Test eines C2000 Mikrocontroller mit MATLAB/Simulink

von

Trautmann, Dietrich
Schillerstraße 2b
83071 Stephanskirchen
Matrikelnummer: 636258

Erstkorrektur:	Prof. Dr.-Ing. Martin Versen
Zweitkorrektur:	Prof. Dr.-Ing. Peter Zentgraf, M.Sc.

Rosenheim, 28.09.2012

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet. Dies gilt auch für bildliche Darstellung sowie für Quellen aus dem Internet.

..... Rosenheim, den.....

D. Trautmann

Danksagung

An dieser Stelle möchte ich mich bei all jenen bedanken, die mir beim Erstellen meiner Bachelorarbeit durch ihre fachliche und persönliche Unterstützung zur Seite gestanden haben. Allen voran danke ich Prof. Dr.-Ing. Martin Versen für die Ermöglichung und den konstruktiven Beistand dieser Arbeit, sowie Andreas Bernhardt für die vielfältige Unterstützung. Des Weiteren danke ich meiner Freundin Caren für das Korrekturlesen der Arbeit. Außerdem bedanke ich mich ganz besonders bei meinen Eltern, die mir dieses Studium ermöglicht haben und bei meiner ganzen Familie für die emotionale Unterstützung.

Zusammenfassung

In der Bachelorarbeit mit dem Titel „Studie zur Toolchain des modellbasierten Softwareentwurfs und Test eines C2000 Mikrocontroller mit MATLAB/Simulink“ wurde der modellbasierte Softwareentwurf für Eingebettete Systeme untersucht.
Die verwendete Software war MATLAB/Simulink mit Toolboxen (Emdedded Coder) und die integrierte Entwicklungsumgebung Code Composer Studio.
Das Eingebettete Systeme war eine controlCARD mit einem Delfino DSP von Texas Instruments die in einem sogenannten Peripheral Explorer Kit eingesteckt war und mit der vorhandenen Peripherie interagierte. Damit wurde eine Linearachse aus dem Praktikumsversuch Nachlaufregelung geregelt.

Inhaltsverzeichnis

1. Einleitung..... 1
2. Stand der Technik..... 3
3. Verwendete Hardware..... 4
3.1. TI C2000 Peripheral Explorer Kit..... 4
3.2. Headerboard HILEVEL Tester..... 6
3.3. Prakikumsversuch Nachlaufregelung..... 8
4. Verwendete Software und Realisierung der Toolchain..... 11
4.1. Software..... 11
4.1.1. MATLAB..... 11
4.1.2. Simulink..... 13
4.1.3. Embedded Coder..... 14
4.1.4. Code Composer Studio v5..... 15
4.2. Konfiguration..... 16
4.2.1. MATLAB Konfiguration..... 16
4.2.2. CCSv5 Konfiguration mit einem Beispielprogramm.. 20
4.3. Datenverarbeitung an einem Beispielmodell..... 24
5. Demonstration der Toolchain..... 29
6. Diskussion und Zusammenfassung..... 35
7. Quellenangabe..... 36

A. Anhang Headerboard (Schaltplan + Layout)
B. Anhang Praktikumsanleitung

Abbildungsverzeichnis

Abbildung 1 : V-Modell..... 2
Abbildung 2 : Peripheral Explorer Kit mit der controlCARD..... 3
Abbildung 3 : Peripherie [7]..... 4
Abbildung 4 : Headerboard Detailansicht..... 6
Abbildung 5 : Headerboard Gesamtansicht (Rückseite)..... 6
Abbildung 6 : Linearachse..... 7
Abbildung 7 : Blockschaltbild Linearachse [8]..... 7
Abbildung 8 : Schaltplan Teil 1..... 9
Abbildung 9 : Schaltplan Teil 2..... 9
Abbildung 10 : MATLAB Oberfläche..... 10
Abbildung 11 : MATLAB Version und die Toolboxen..... 11
Abbildung 12 : Simulink Aufruf in MATLAB..... 12
Abbildung 13 : Simulink Library Browser..... 12
Abbildung 14 : Embedded Coder in der Simulink Library..... 13
Abbildung 15 : CCS Oberfläche..... 14
Abbildung 16 : Command Window Befehleingabe..... 15
Abbildung 17 : XMakefile User Configuration..... 15
Abbildung 18 : XMakefile User Configuration: Tool Directories..... 16
Abbildung 19 : New Configuration Name..... 17
Abbildung 20 : Compiler..... 18
Abbildung 21 : Linker..... 18
Abbildung 22 : Archiver..... 18
Abbildung 23 : Workspace..... 19
Abbildung 24 : New CCS Project..... 19

Abbildung 25 : Angelegtes Projekt mit einer C-Code Vorlage von main.c... 20
Abbildung 26 : System Stack..... 20
Abbildung 27 : C-Code..... 21
Abbildung 28 : Debug..... 21
Abbildung 29 : CCS Debug..... 22
Abbildung 30 : Breakpoint..... 22
Abbildung 31 : Baustein Target Preferences..... 23
Abbildung 32 : Target Preferences..... 24
Abbildung 33 : Simulink Modell..... 25
Abbildung 34 : Digital Output..... 25
Abbildung 35 : Launch Selected Configuration..... 26
Abbildung 36 : Connect Target..... 27
Abbildung 37 : OUT-Datei Auswahl..... 27
Abbildung 38 : Resume..... 28
Abbildung 39 : Neues Modell..... 29
Abbildung 40 : Konfigurations-Parameter..... 29
Abbildung 41 : Simulink Modell..... 30
Abbildung 42 : SOLL-Wert ADC Control..... 31
Abbildung 43 : SOLL-Wert Input Channels..... 31
Abbildung 44 : IST-Wert ADC Control..... 31
Abbildung 45 : IST-Wert Input Channles..... 31
Abbildung 46 : ePWM General..... 32
Abbildung 47 : ePWMA..... 32
Abbildung 48 : Subsystem 1..... 33
Abbildung 49 : Subsystem 2..... 33
Abbildung 50 : Subsystem 3..... 33
Abbildung 51 : Build Model..... 34

Abkürzungsverzeichnis

UML : **Unified Modeling Language**
DIMM : **Dual In-Line Memory Module**
ADC : **Analog-to-Digital Converter**
PWM : **Puls-Width Modulation**
AGND : **Analog Ground**
DSP : **Digital Signal Processor**
ePWM : **Enhanced Puls-Width Modulation**

1. Einleitung

In der hochtechnisierten Welt von heute werden die Funktionen der Maschinen maßgeblich durch die erstellte Software bestimmt. Die dafür notwendige Mechanik und Elektronik, die die Basis eines Produktes ist erfährt zwar weiterhin kleine Verbesserungen, ihr Anteil an der Innovation ist im Vergleich zur Software viel geringer. In solchen sogenannten Eingebetteten Systemen ist die Software damit die (Funktions-)Brücke zwischen der Mechanik und der Elektronik. Damit werden hoch automatisierte intelligente und effiziente Systeme möglich.

Früher war die Software viel einfacher auf Grund der begrenzten Ressourcen, durch die Hardware (Elektronik) und die nicht so weit-entwickelten Programmiersprachen und -techniken. Über den Programmaufbau machte man sich in dieser Hinsicht weniger Gedankten, da die Programme überschaubar waren und während des Entwurfs einfach sequentiell runter geschrieben wurden.

Das geht heute nicht mehr, denn die Softwareprojekte für eingebettete Systeme haben eine enorme Komplexität erreicht und ein großer Teil der Entwicklungszeit eines Produktes wird für das Programmieren benötigt. Das bedeutet eine vorher gut überlegte Softwarearchitektur, die mit ingenieurmäßigen Vorgehensweisen, Methoden, Techniken sowie Werkzeugen zu bewältigen ist. Das Endprodukt soll softwaremäßig in einem vernünftigen Rahmen wartbar und nicht für Fehler anfällig sein, damit eine garantierte Qualität gegeben ist. Eine mögliche Lösung ist der Ansatz des modellbasierten Softwareentwurfs [1].

Der modellbasierte Softwareentwurf bedeutet, am Ende gibt es lauffähige Software. Sie ist also nicht nur für Dokumentationszwecke oder als Spezifikation gedacht [2]. Die lauffähige Software wird automatisiert aus Modellen, in dieser Bachelorarbeit aus MATLAB/Simulink-Modellen generiert. Modelle können dabei textuell oder grafisch sein. Die Simulink-Modelle sind grafische Modelle. Ein Beispiel für textuelle Modelle wären UML[1]-Modelle, die hier aber nicht behandelt werden.

Der modellbasierte Softwareentwurf bietet Vorteile bei der Qualität der erstellten Software, da die Code Erstellung aus den Modellen standardisiert und zertifiziert werden kann, infolgedessen also keine formalen Fehler im Quelltext vorkommen. Ein weiterer wichtiger Punkt ist die Dokumentation der Programme die parallel aus den Modellen generiert wird, denn die Modelle sind viel verständlicher als Quelltext-Passagen.

Der Aufbau der Bachelorarbeit gliedert sich wie folgt. Nach der Einleitung folgt der Stand der Technik des modellbasierten Softwareentwurfs in Kapitel 2. In Kapitel 3 wird die Hardware beschrieben die bei den Tests zum Einsatz kam. Daran anschließend wird in Kapitel 4 die eingesetzte Software beschrieben, sowie die Realisierung der Toolchain gezeigt. Das Kapitel 5 beschäftigt sich mit der Demonstration der gesamten Toolchain an einem Praktikumsversuch. Abschließend werden die Ergebnisse in Kapitel 6 diskutiert und zusammengefasst.

[1] UML: Unified Modeling Language

2. Stand der Technik

Der modellbasierte Softwareentwurf findet zunehmend Verwendung im industriellen Einsatz. Dabei wird unter anderem nach dem V-Modell geplant und entwickelt [4]. Das V-Modell organisiert den Systementwurf in Phasen (Abbildung 1). Der modellbasierte Softwareentwurf ist dabei am detailliertesten, hier wird die genaue Funktion umgesetzt, und befindet sich deshalb in der untersten Phase. Dabei beschreiben die Modelle das System mit größeren Bausteinen, als es eine Programmiersprache kann, somit wird auf einer höheren Abstraktionsebene programmiert [2]. Die Modellierungstechnik kann prinzipiell in jeder Phase stattfinden, in dieser Arbeit wurde auf das Programmieren der eingebetteten Systeme eingegangen. Zu erwähnen wäre, dass nicht nur die einzelnen Stufen sondern auch zwischen der linken und der rechten Hälfte getestet wird, ob die Vorgaben erfüllt werden.

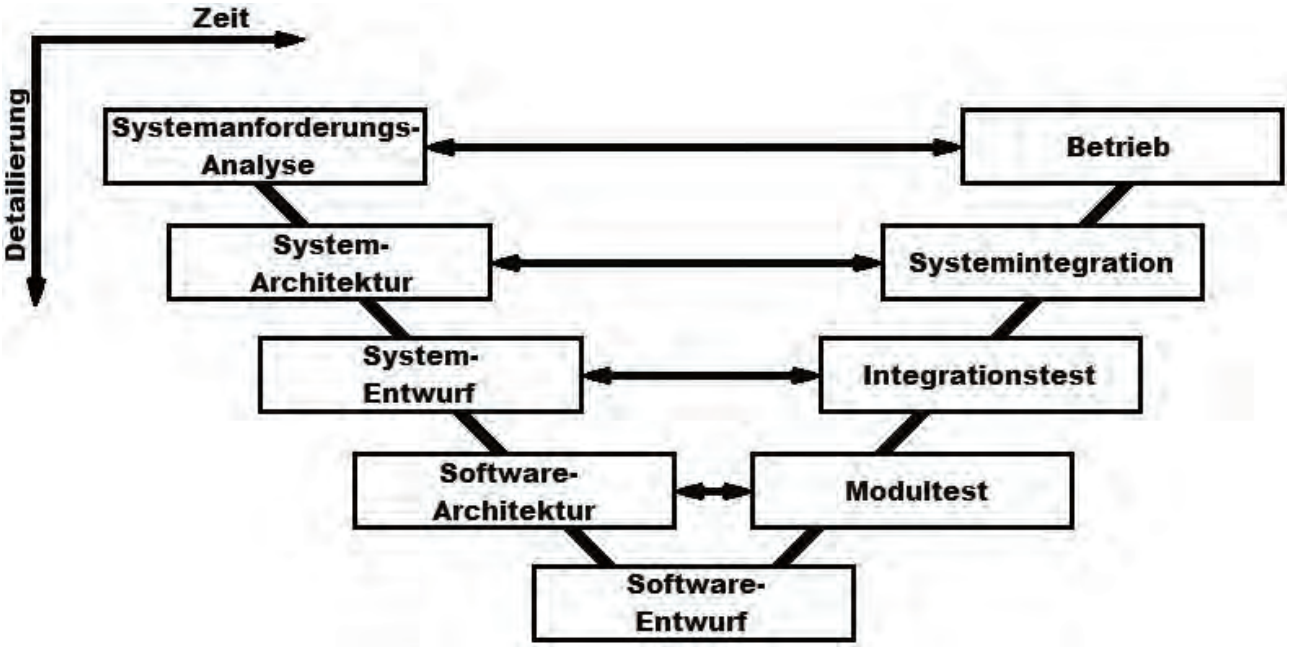


Abbildung 1: V-Modell [4]

In der Industrie wird für eine vollständige Modellierung eingebetteter Software meist nicht nur ein Modellierungssprache verwendet. Es werden mehrere Modellierungssprachen miteinander kombiniert [1]. Dabei wird unter anderem auf das V-Modell zurück gegriffen um zu bewerten, wo welche Modellierungssprache zum Einsatz kommt.
Nach [5] ist die Qualität modellbasierter Softwareentwicklung mit zwei Aspekten festlegbar: Die richtige Ausführung des Algorithmus und die Zuverlässigkeit und Reproduzierbarkeit der Softwareerstellung aus den Modellen.
Das Testen wird zunehmend wichtiger und umfangreicher. Die Verfahren dafür werden unter dem Stichwort „modellbasiertes Testen“ zusammengefasst.

3. Verwendete Hardware

Bei der Erstellung der Bachelorarbeit ist die nachfolgend beschriebene Hardware verwendet worden. Dabei wurde das TI C2000 Peripheral Explorer Kit gekauft, das Headerboard für den HILEVEL Tester selbst erstellt (Schaltplan und Layout) und der Praktikumsversuch „Nachregelung (Linearachse)“ aus dem Regelungstechnik Labor der FH Rosenheim im Studiengang Produktionstechnik verwendet.

3.1. TI C2000 Peripheral Explorer Kit

Das Board des TI C2000 Peripheral Explorer Kit [6] enthält Peripherie z.B. LEDs, Drehpotentiometer, Temperatur Sensor, Anschlüsse für Audio-In und Headphone-Out und viel mehr für einen schnellen Einstieg in die Programmierung eines C2000 Mikrocontroller und zum Experimentieren. Über den 100-pin DIMM[1] Sockel können verschiedene controlCARDs eingesteckt werden. Die verwendete controlCARD [7] enthält den C2000 TMS320F28335 Mikrocontroller [8].

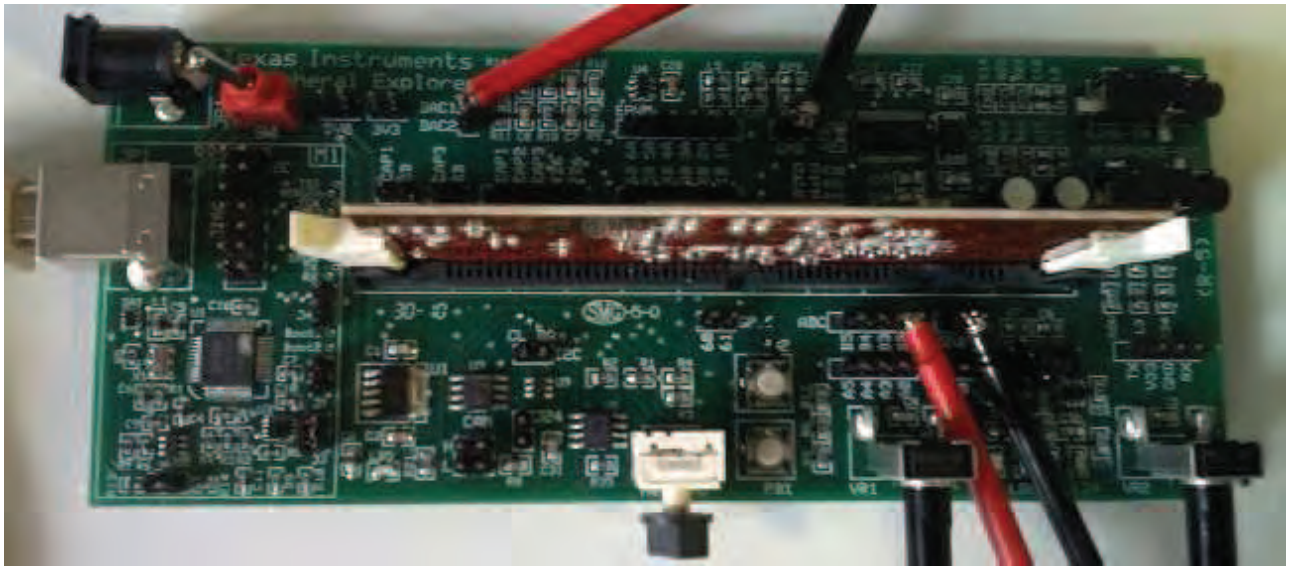


Abbildung 1: Peripheral Explorer Kit mit der controlCARD

In der Abbildung 2 ist das verwendete Peripheral Explorer Kit Board mit der eingesteckten controlCARD zu sehen.

[1] DIMM: Dual In-Line Memory Module

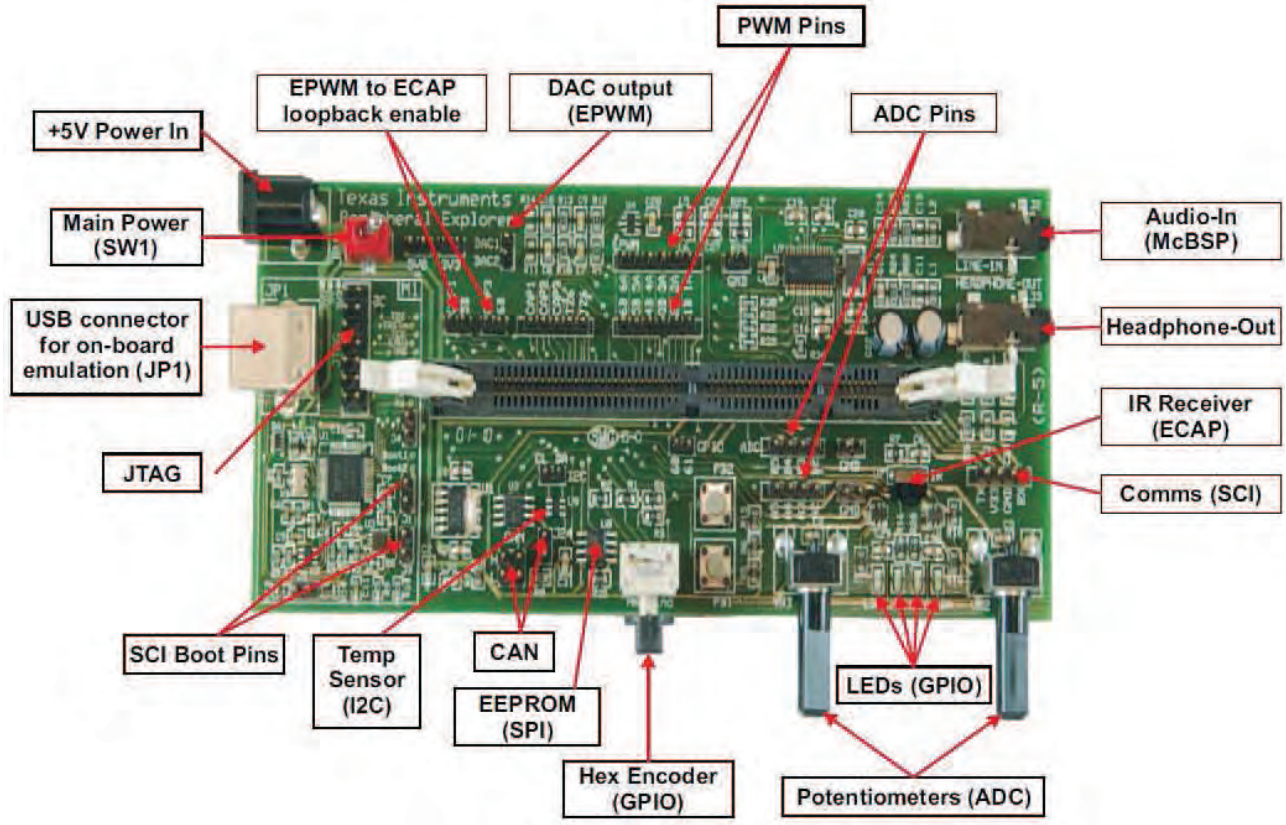


Abbildung 2: Peripherie [9]

Die Abbildung 3 stellt das Peripheral Explorer Kit Board dar und beschreibt die verfügbare Peripherie. Für die ersten Tests der Toolchain wurde einer der beiden Potentiometer, die ADC[2] Pins und die PWM[3] Pins verwendet. Die genaue Umsetzung und Verwendung wird im Kapitel 5 beschrieben.

[2] ADC: Analog-to-Digital Converter
[3] PWM: Puls-Width Modulation

3.2 Headerboard HILEVEL Tester

Für das Testen des Chips (Hardware-in-the-Loop - HiL) dient das Headerboard als Verbindungsglied zwischen den HILEVEL Tester (HiL-Simulator) und der controlCARD (HiL) mit dem Mikrocontroller TMS320F28335. Das Testen ist der nächste logische Schritt nach dem Realisieren der Toolchain, damit sichergestellt werden kann, dass der Code fehlerfrei ausgegeben wird. Die PINs des Headerboards „GPIO-PIN“ der controlCARD sind jeweils mit den in der Tabelle 1 aufgeführten PINs des HILEVEL Tester verbunden.

Headerboard Bottom		**Headerboard Top**	
Belegung Headerboard	Belegung HILEVEL Tester	Belegung Headerboard	Belegung HILEVEL Tester
GPIO-59	30	GPIO-06	3
GPIO-63	31	GPIO-04	6
GPIO-61	32	GPIO-01	9
GPIO-84	49	GPIO-05	12
GPIO-26	51	GPIO-21	33
GPIO-86	52	GPIO-25	34
-	54	GPIO-85	35
GPIO-12	55	GPIO-29	36
GPIO-18	57	GPIO-17	37
GPIO-15	58	GPIO-13	38
GPIO-22	60	GPIO-33	39
GPIO-24	61	GPIO-31	42
GPIO-28	63	GPIO-23	45
GPIO-48	83	GPIO-19	48
SCK	86	GPIO-03	71
GPIO-62	89	GPIO-00	74
GPIO-60	92	GPIO-02	77
GPIO-58	95	GPIO-08	80
GPIO-32	114	GPIO-27	99
GPIO-34	117	GPIO-14	102
GPIO-30	120	GPIO-49	105
GPIO-87	123	SDA	108
GPIO-20	126	GPIO-09	111
GPIO-16	128	GPIO-07	112

Die Analogen Eingänge des TMS320F2833 sind separat hinausgeführt und können an Lötösen abgegriffen werden, da der verwendete HILEVEL Tester zum aktuellen Stand keine analogen Eingänge verarbeiten/simulieren kann. Parallel dazu wurden die AGND[4] Pins angeordnet.

[4] AGND: Analog Ground

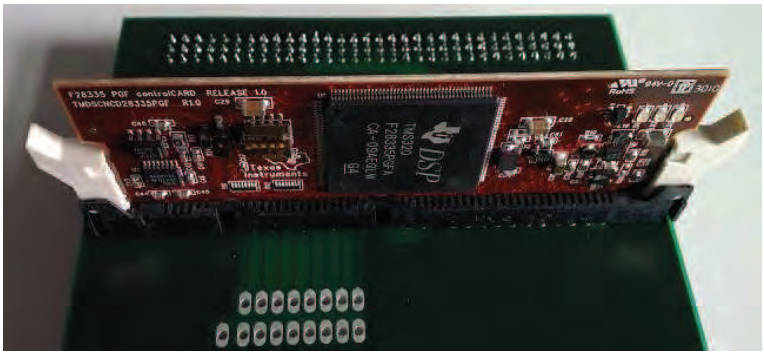


Abbildung 3: Headerboard Detailansicht

In der Abbildung 5 ist das gesamte Headerboard zu sehen (Rückseite).

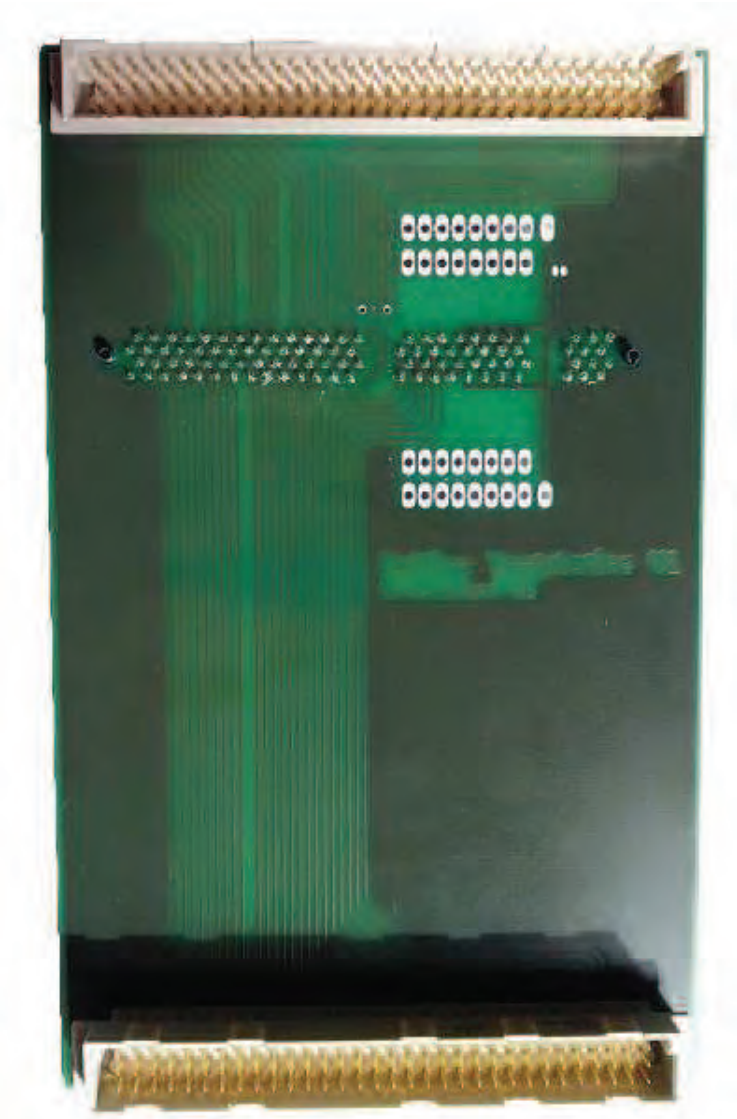


Abbildung 4: Headerboard Gesamtansicht (Rückseite)

Der erstellte Schaltplan und das Layout sind im Anhang A zu finden.

3.3 Praktikumsversuch Nachlaufregelung

Der Praktikumsversuch Nachlaufregelung besteht aus einer Linearachse mit der eine gewünschte Position angefahren werden kann. Der Aufbau der Linearachse ist in der Abbildung 6 zu sehen.

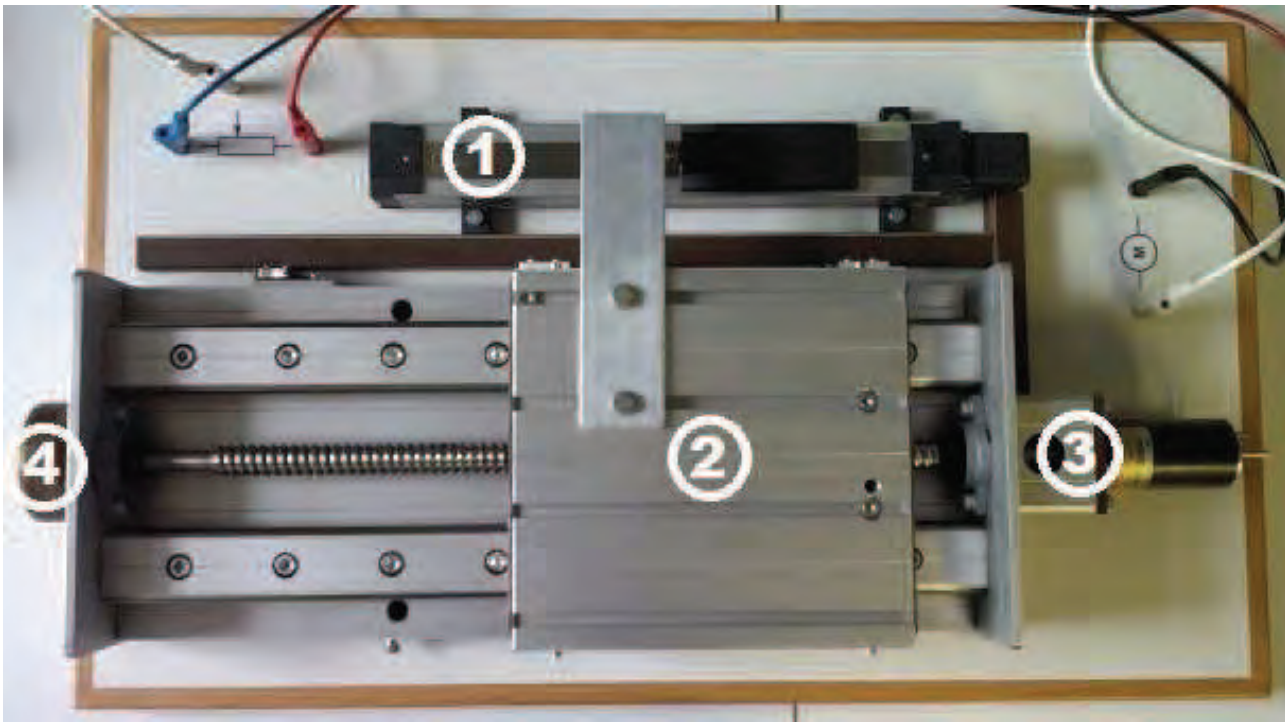


Abbildung 5: Linearachse

Die eingezeichneten Markierung stellen Folgendes dar:

1 Schiebepotentiometer (Positionsbestimmung, Streckeneingang)
2 Schlitten
3 Gleichstrommotor (über Leistungsverstärker, Streckenausgang)
4 Schwungmasse

In Abbildung 7 ist die Linearachse in einem Blockschaltbild dargestellt.

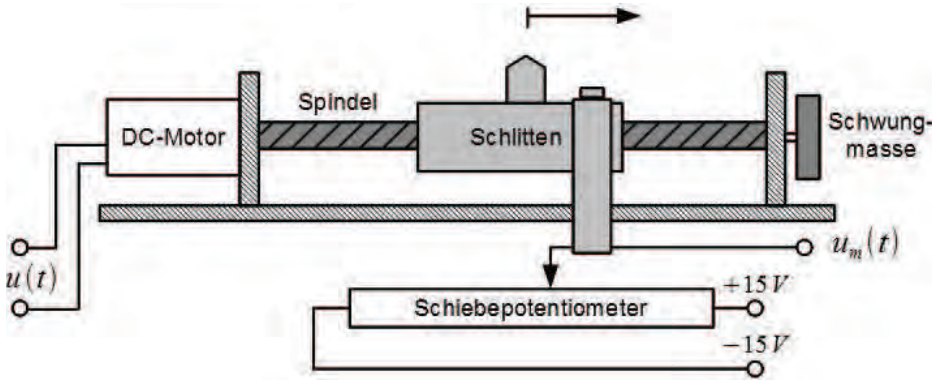


Abbildung 6: Blockschaltbild Linearachse [10]

Funktion der Linearachse:
Beim Anlegen einer Spannung am Gleichstrommotor dreht sich die Spindel und der Schlitten wird je nach Drehrichtung nach links oder rechts bewegt. Der am Schlitten befestigte Schieberegler des Schiebepotentiometers wird dabei mitbewegt. Somit erfährt die am Schiebepotentiometer gemessene Spannung eine proportionale Wertänderung. Durch diesen Vorgang wird eine Positionsbestimmung ermöglicht. Die Linearachse hat die Spannung für den Gleichstrommotor als einen Eingang und liefert mit dem Schiebepotentiometer eine Spannung.

Nachlaufregelung:
Der Regler hat die eingestellte Spannung am Drehpotentiometer des TI C2000 Peripheral Explorer Kit Boards und die gemessene Spannung am Schiebepotentiometer der Linearachse als Eingang. Mit dem Potentiometer des Boards wird die Sollwertvorgabe durchgeführt.

Potentiometer Drehung nach Links	→	Schlitten fährt nach Links
Potentiometer Drehung nach Rechts	→	Schlitten fährt nach Rechts

Es kann jede beliebige Position zwischen den Endwerten angefahren werden.

Als Ausgang muss der auf dem Delfino DSP[5] implementierte Regler einen analogen Wert für den Gleichstrom bereitstellen. Da die verwendete controlCARD keinen analogen Ausgang hat, wird ein ePWM[6] Signal erzeugt. Die controlCARD mit dem Delfino TMS320F28335 kann für die verwendeten PIN nur Spannungen im Bereich von 0-3V verarbeiten (siehe [9] Seite 3 und [11] Seite 77). Aufgrund dessen wurde eine Schaltung entworfen, die den Wertebereich anpasst.

Vorverschaltung:
Die Schaltung in der Abbildung 8 erzeugt aus dem Wertebereich (-15V bis +15V) des Schiebepotentiometers einen für den analogen Eingang [11] am Peripheral Explorer Kit Board gültigen Wertebereich (0 bis 3V).
Der erste am Schiebepotentiometer angeschlossene invertierende Operationsverstärker hat eine Verstärkung von etwa -1/7 was den Wertebereich auf ungefähr -2V bis 2V verkleinert. Der daran anknüpfende invertierende Operationsverstärker ist ein Addierer. Dabei bestimmt U_1 um wie viel der Bereich verschoben werden soll. Es wurde ein Wert von -2V für U_1 eingestellt. Damit ergibt sich ein Wertebereich von 0 bis 4V. Da der Sollwert über das Drehpotentiometer am Board nur im Bereich zwischen 0 und 3,3V eingestellt werden kann, wird vom Schiebepotentiometer auch nur ein Wert in diesem Bereich erwartet.

[5] DSP: Digital Signal Processor
[6] ePWM: Enhanced Pulse-Width Modulation

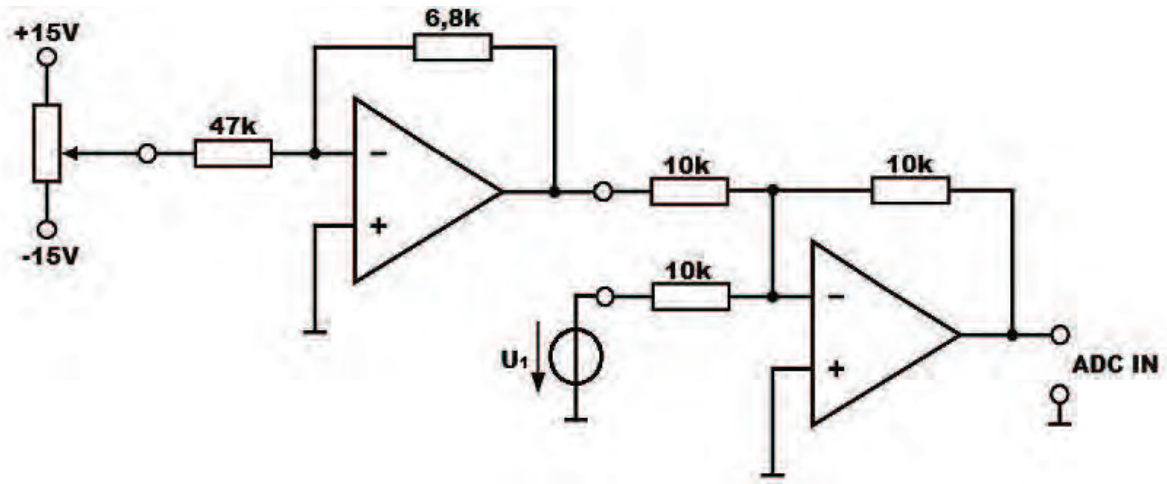


Abbildung 7: Schaltplan Teil 1

Die Schaltung in der Abbildung 9 macht aus dem Wertebereich (0 bis 3,3V) des ausgegebenen ePWM-Signal ein für die Ansteuerung des Motors benötigten Wertebereich (-10V bis 10V).
An den ePWM PIN des Boards ist ein invertierender Operationsverstärker angeschlossen. Mit diesem Addierer wird durch U_2 bestimmt, wie der Eingangswert verschoben wird. Da das ausgegebene ePWM-Signal nur zwischen 0 und 3,3V liegt, wird bei der Wahl von U_2 = -1,65V ein symmetrischer Wertebereich von -1,65V bis +1,65V erzeugt. Durch den folgenden invertierenden Operationsverstärker mit einer Verstärkung von etwa 6, wird der Wertebereich auf -10V bis 10V erweitert. Mit dem vor dem Motor geschalteten Leistungsverstärker ist die Schaltung komplett.

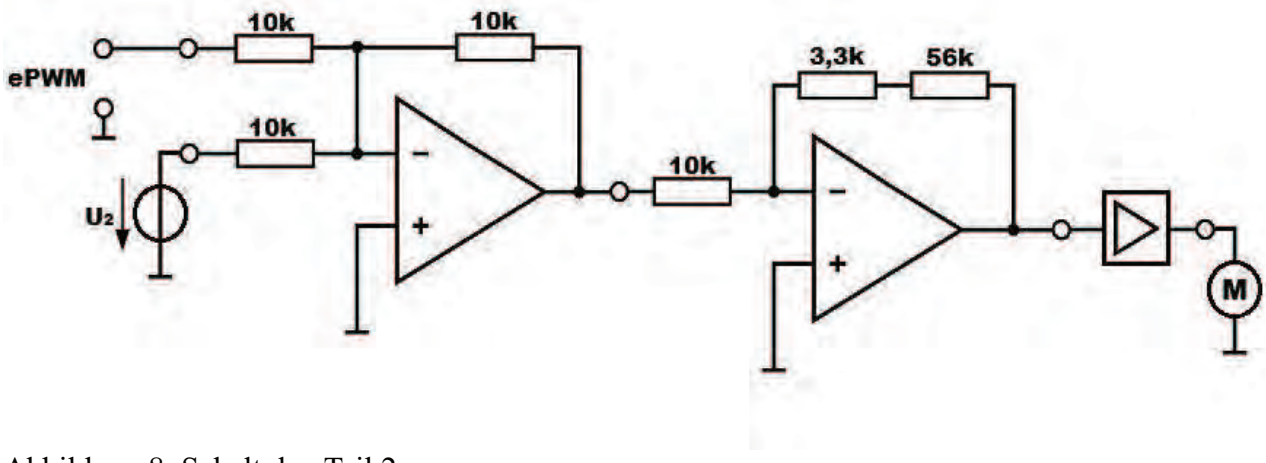


Abbildung 8: Schaltplan Teil 2

4. Verwendete Software und Realisierung der Toolchain

4.1. Software

4.1.1. MATLAB

MATLAB, nach [3] und [12] ist ein Programm und eine Hochsprache. Damit können Berechnungen durchgeführt und anschließend visuell dargestellt werden. Die Software ist ein beliebtes Tool in der Industrie sowie an Hochschulen und wird unter anderem für regelungstechnische Aufgaben eingesetzt. Die grafische Oberfläche ist in der Abbildung 10 gezeigt.

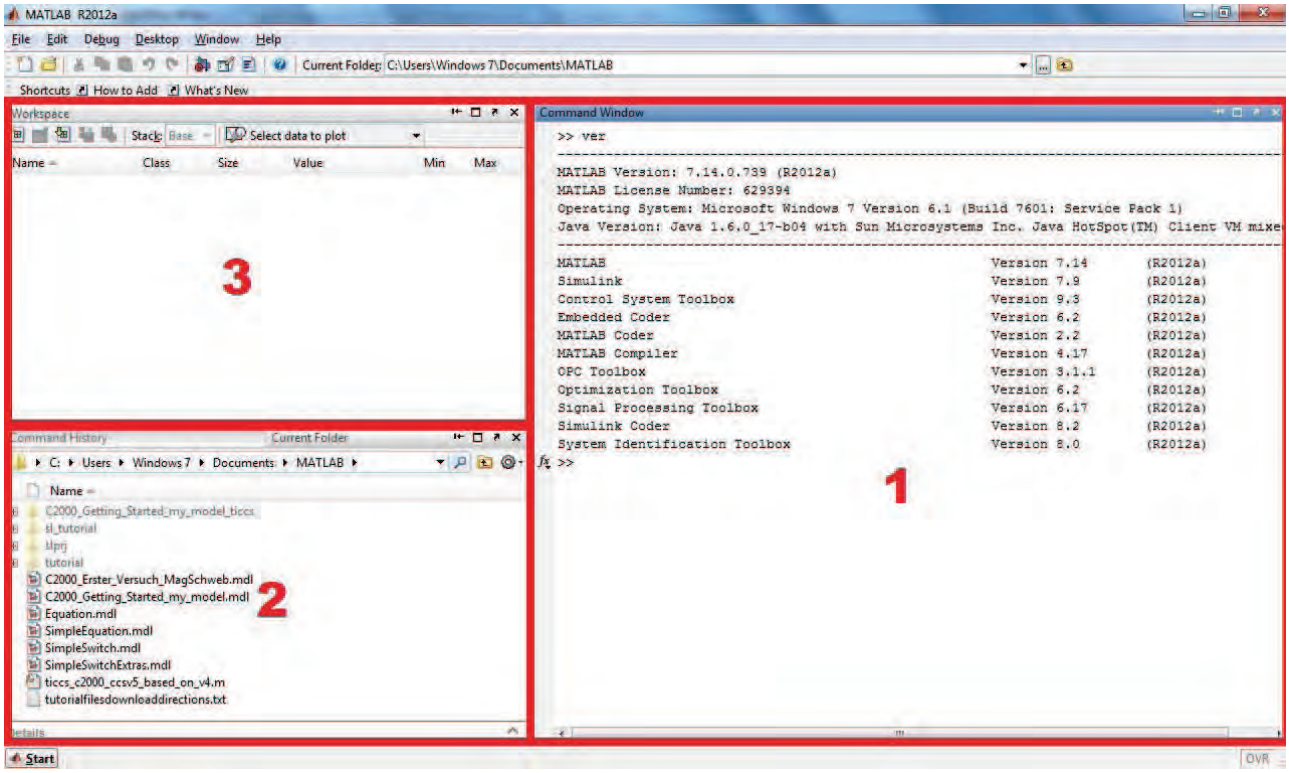


Abbildung 10: MATLAB Oberfläche

Die eingezeichneten Markierungen stellen folgendes dar:

1 Command Window
2 Current Folder und Command History
3 Workspace

Im Command Window werden alle MATLAB Befehle eingegeben und alle Berechnungen, Warnungen sowie Informationen ausgegeben. Der Current Folder zeigt das aktuelle Verzeichnis an und hier können weitere MATLAB Modelle geladen werden. Die Command History listet alle im Workspace eingegebenen Befehle auf. Im Workspace werden alle in der aktuellen MATLAB-Sitzung vorhandenen Variablen mit den entsprechenden Werten hinterlegt.

Nach eingeben des MATLAB Befehls `ver` im Command Window, wird die aktuelle MATLAB Version mit allen installierten Toolboxen angezeigt. Für die Durchführung dieser Anleitung wird Simulink sowie der Embedded Coder benötigt.

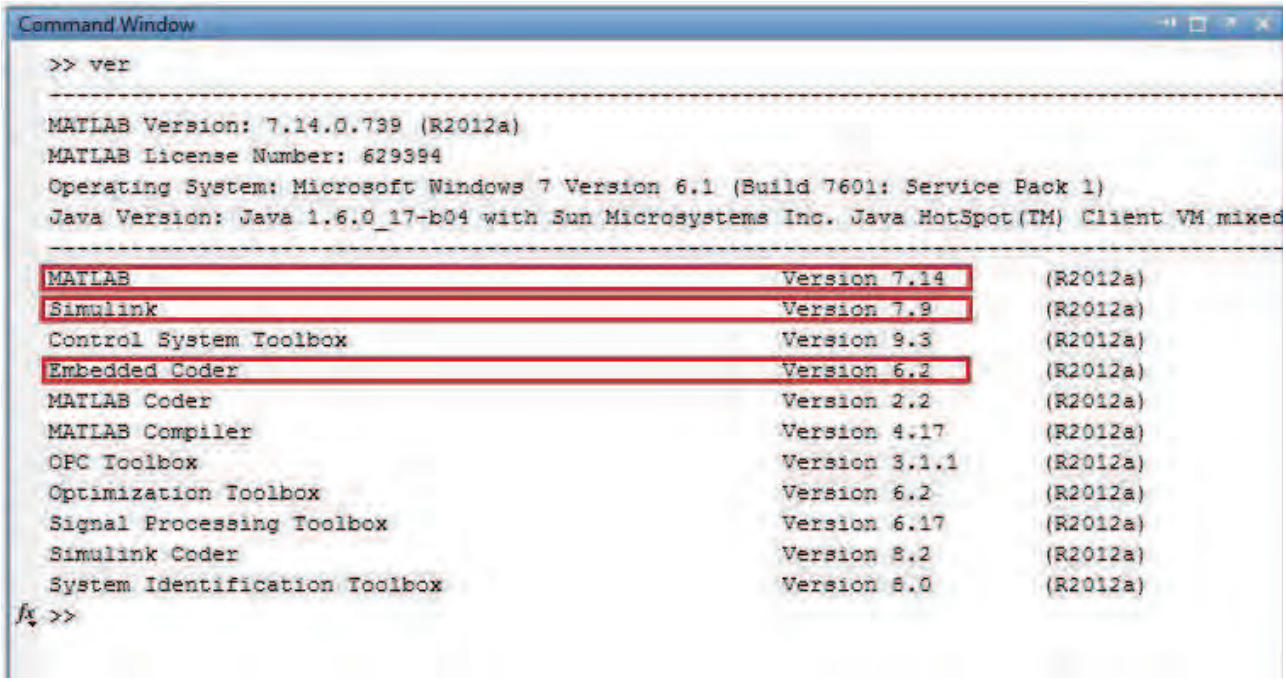


Abbildung 11: MATLAB und Toolboxen Version

Durch das Eingeben des Befehls `ver` soll überprüft werden, ob die für die Durchführung der Anleitung benötigten Toolboxen (Abbildung 11) installiert sind.

MATLAB	Version 7.14
Simulink	Version 7.9
Embedded Coder	Version 6.2

4.1.2. Simulink

Mit Simulink, nach [3] und [13] kann das modellbasierte Design und verschiedene Simulationen durchgeführt werden. Da hier mit Modellen (grafische Programmiersprache) gearbeitet wird und nicht mit Befehlen, erlaubt Simulink einen einfachen Überblick über den Sachverhalt. Die Simulink Library kann aus MATLAB wie in der Abbildung 12 zu sehen, aufgerufen werden.

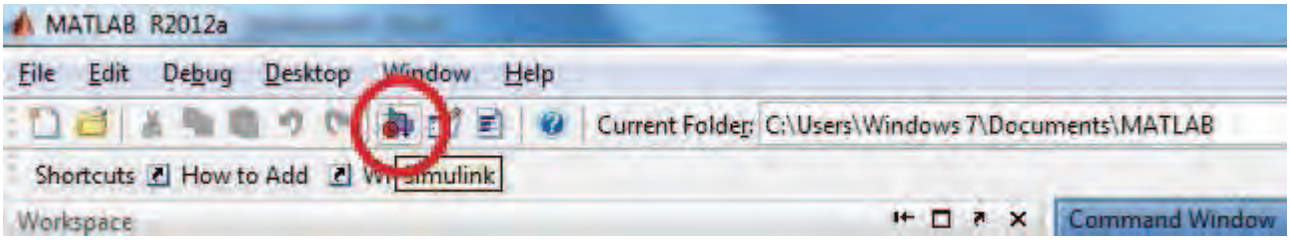


Abbildung 12: Simulink Aufruf in MATLAB

Die daraufhin erscheinende Simulink Library (Abbildung 13) erlaubt das Erstellen eines neuen Modells über ***File > New > Model***. In dem sich geöffneten noch leeren Modellfenster werden die benötigten Blöcke aus der Simulink Library eingefügt.

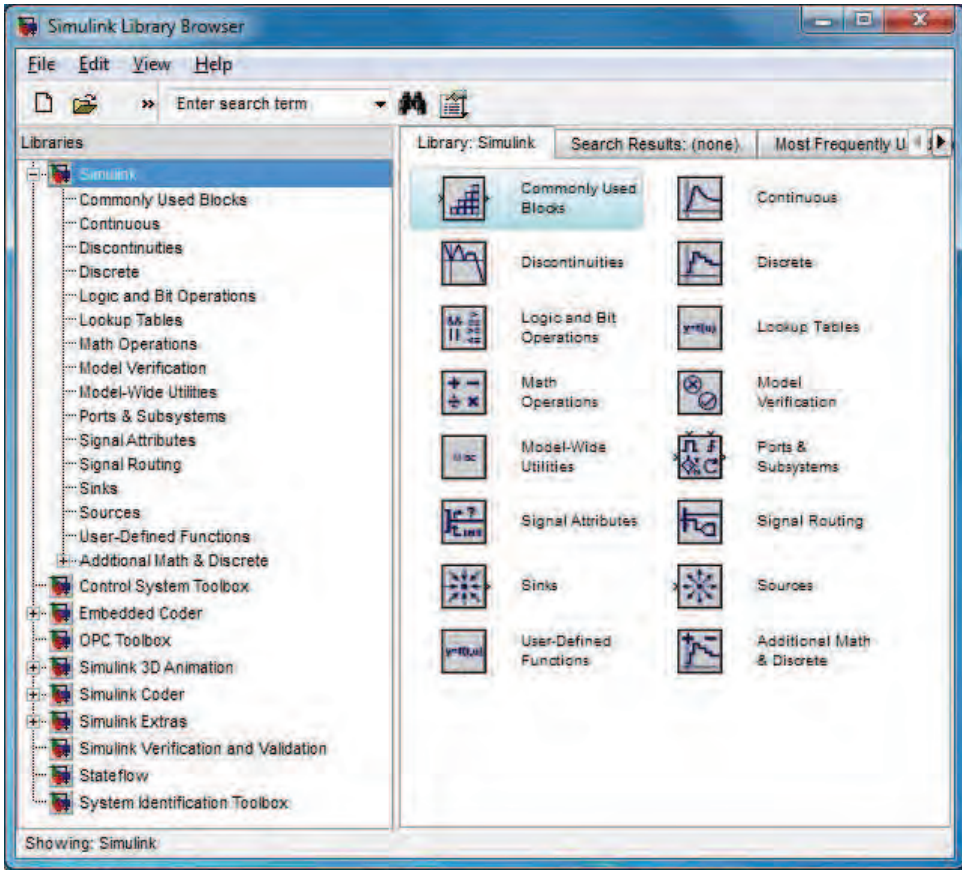


Abbildung 13: Simulink Library Browser

4.1.3. Embedded Coder

Eine während dieser Arbeit verwendete Toolbox in MATLAB ist der Embedded Coder [14]. Damit kann Code für verschiedene DSPs erzeugt werden. Im Simulink Library Browser befindet sich eine Kategorie „Embedded Coder“ in der für die jeweilige Zielhardware bestimmte Blöcke abgelegt sind.

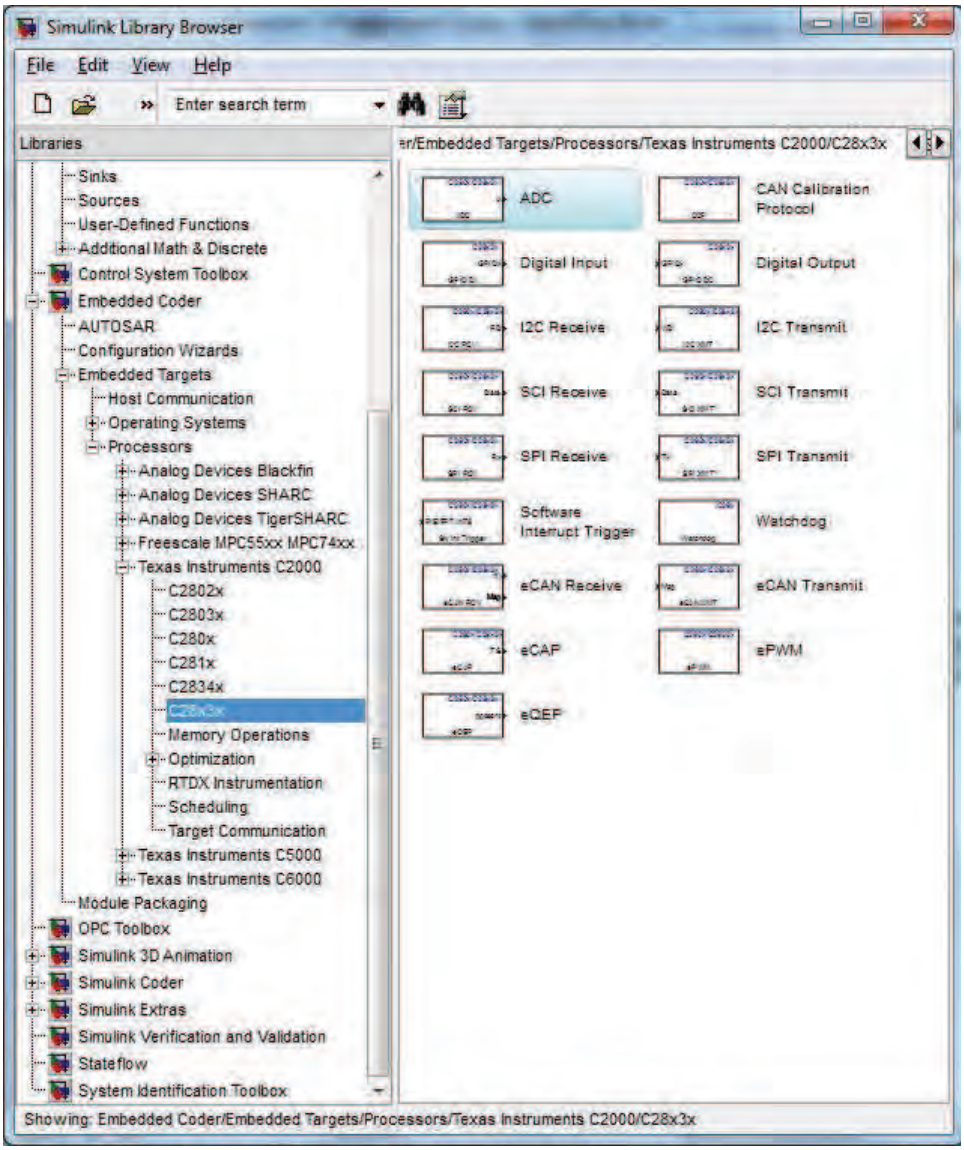


Abbildung 14: Embedded Coder in der Simulink Library

Für das verwendete TI C2000 Peripheral Explorer Kit mit der controlCARD und dem TMS320F28335 DSP finden sich die Blöcke unter ***Embedded Coder > Embedded Targets > Processors > Texas Instruments C2000 > C28x3x*** (Abbildung 14).
Der Target Preference Baustein, der bei Simulink Modellen für Embedded Systeme vorhanden sein muss, findet sich unter ***Embedded Coder > Embedded Targets > Target Preferences*** (Abbildung 31).

4.1.4. Code Composer Studio v5

Die von Texas Instruments im Rahmen vieler Anwendungen kostenlos zur Verfügung gestellte Entwicklungsumgebung CCS [15] basiert auf einer ehemals für Java entwickelten quelloffenen integrierten Entwicklungsumgebung. Durch die vielen Erweiterungen ist es aber mittlerweile möglich für nahezu alle Programmiersprachen zu entwickeln. Mit CCS kann nicht nur der C2000 programmiert werden sondern alle von TI angebotenen Mikrocontroller. Die grafische Benutzeroberfläche ist in der nachfolgenden Abbildung 15 gezeigt.

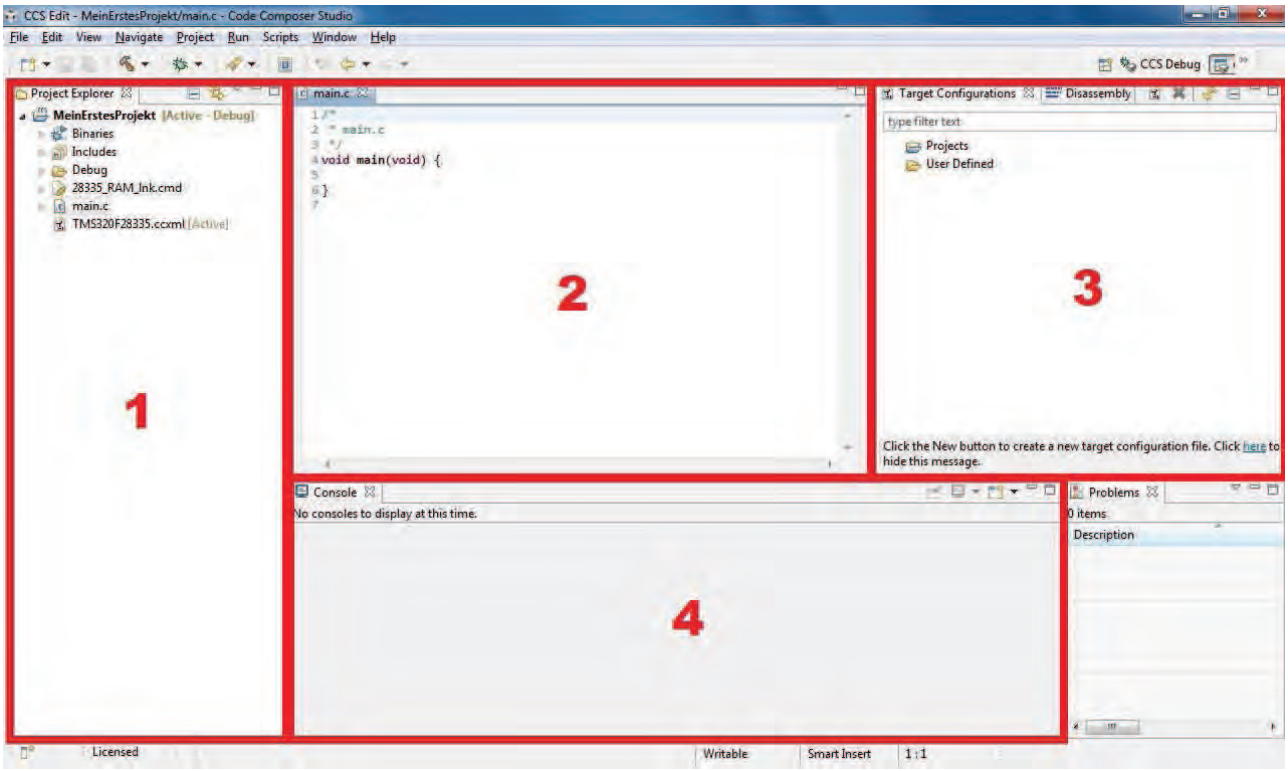


Abbildung 15: CCS Oberfläche

Die eingezeichneten Markierungen stellen folgendes dar:

1 Project Explorer (Überblick der angelegten Projekte)
2 Quellcode (Hier kann der Quellcode erstellt und editiert werden)
3 Target Configuration (Einstellungen der Zielhardware)
4 Console (Ausgabe zu einem Prozess ist hier einsehbar)

Die Fenster können innerhalb der Entwicklungsumgebung frei konfiguriert werden, d.h. verschoben, geschlossen und über die Menüleiste unter ***View*** wieder geöffnet werden.

4.2. Konfiguration

4.2.1. MATLAB Konfiguration

Der folgende Abschnitt beschreibt die Konfiguration in MATLAB.

Im Command Window von MATLAB bitte folgenden Befehl eintippen: `xmakefilesetup` (Abbildung 16)

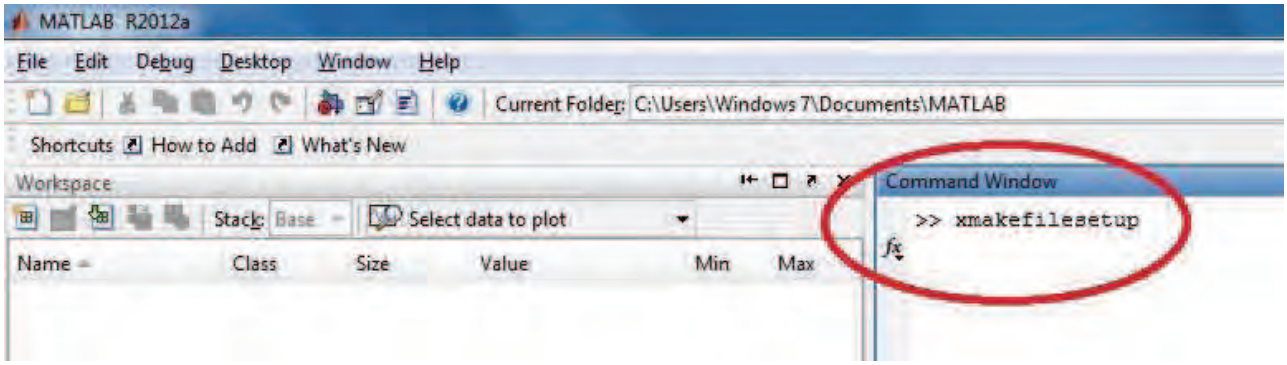


Abbildung 16: Command Window Befehleingabe

Daraufhin erscheint ein neues Fenster ***XMakefile User Configuration*** (Abbildung 17) in dem alle folgenden Einstellungen vorgenommen werden.

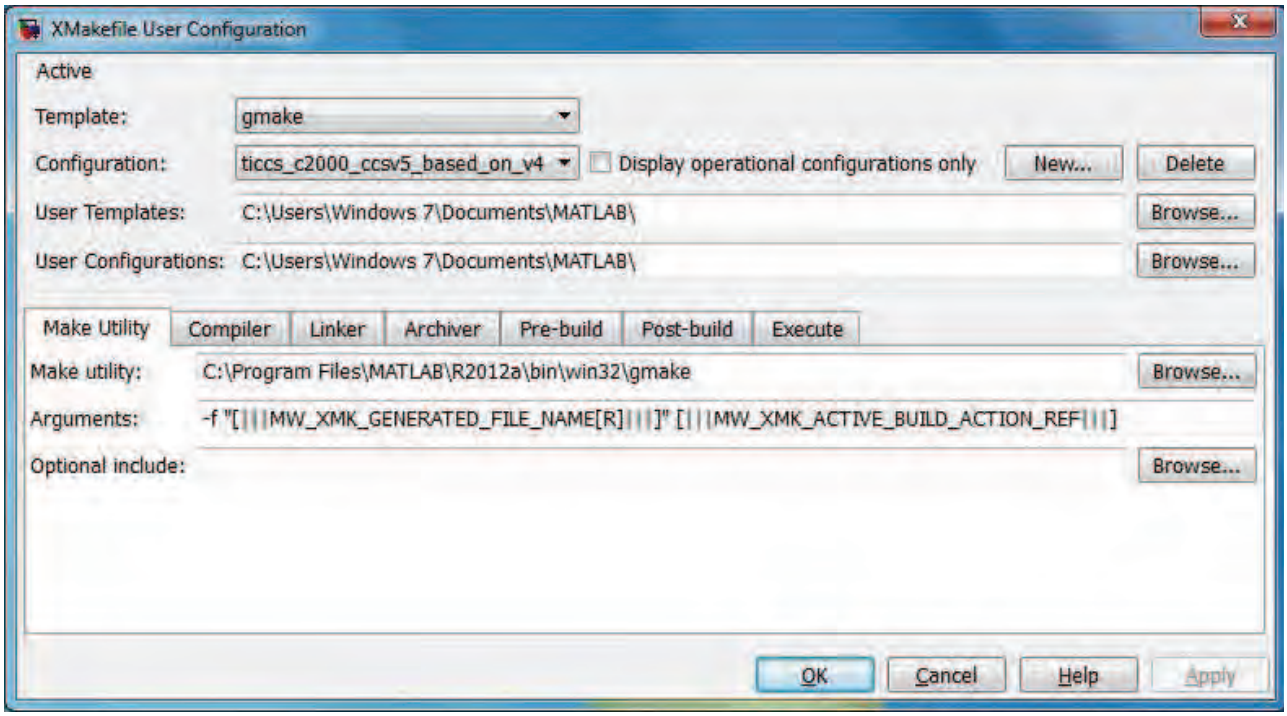


Abbildung 17: XMakefile User Configuration

Unter ***Template*** wird ***gmake*** ausgewählt und das Häckchen bei ***Display operational configurations only*** entfernen. Desweiteren sollte unter ***Configuration***, ***ticcs_c2000_ccsv4***(!) ausgewählt werden. Dann einen Klick auf ***Apply***, ohne das Fenster zu schließen.

Falls der Aufruf der XMakefile User Configuration zum ersten Mal erfolgt, so wird man nach dem Installationspfad für Code Composer Studio v4(!) gefragt. Es wird nach CCSv4 gefragt, da die neueste Version CCSv5 erst vor kurzem veröffentlicht wurde. Hier bitte den Installationspfad der verwendeten Version von Code Composer Studio angeben. Das sieht dann zum Beispiel so aus: ***C:\ti\ccsv5*** bei der Verwendung von CCSv5.

Wenn der Aufruf schon bei der Konfiguration von CCSv4 erfolgte, dann sollen die Angaben für CCSv5 unter ***Tool Directories*** editiert werden.

Das sieht dann folgendermaßen aus (Abbildung 18):

CCS Installation: C:\ti\ccsv5\
Code Generation Tolls: C:\ti\ccsv5\tools\compiler\c2000_6.1.0\
DSP/BIOS Installation: C:\ti\bios_6_33_05_46\

Jetzt wieder ein Klick auf ***Apply*** ohne das Fenster zu schließen.

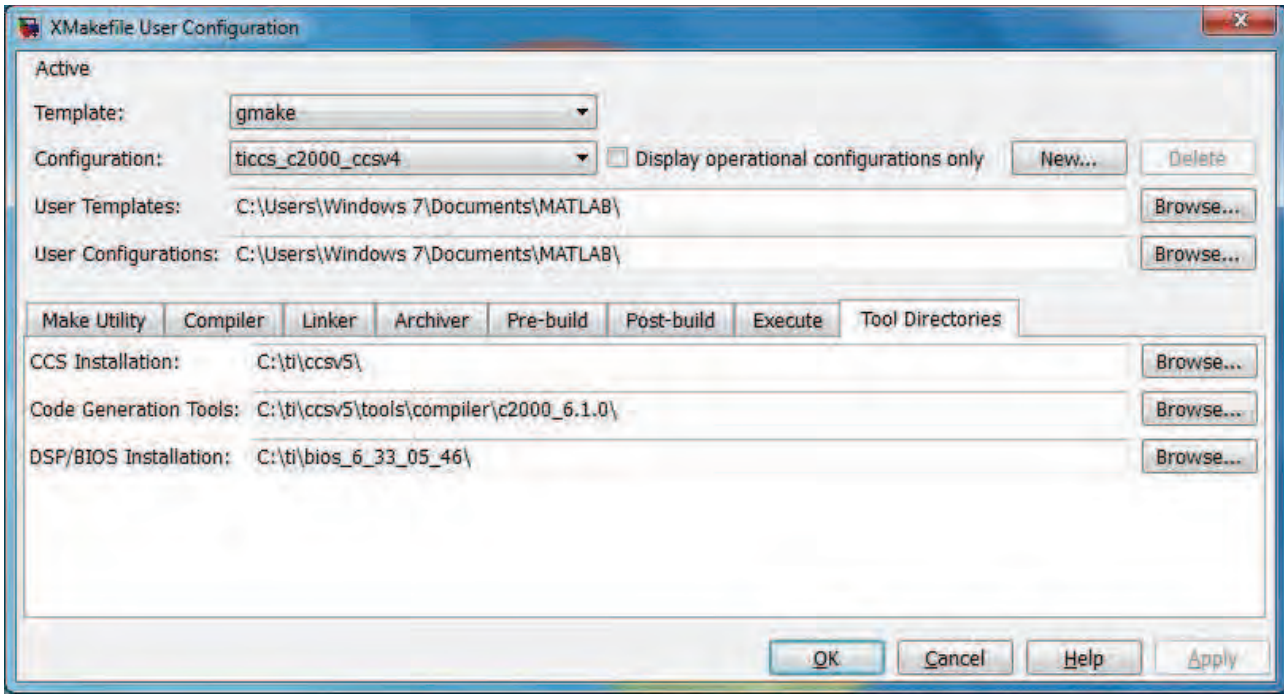


Abbildung 18: XMakefile User Configuration: Tool Directories

Der nächste Schritt wäre ein Klick auf ***New*** ganz rechts im XMakefile User Configuration Fenster. Es wird vorgeschlagen, die neue Konfiguration z.B. ***ticcs_c2000_ccsv4_clone*** zu nennen. Ein Vorschlag wäre ***ticcs_c2000_ccsv5_based_on v4*** (Abbildung 19) und anschließend auf ***OK*** klicken.

Abbildung 19: New Configuration Name

Alle Reiter in der XMakefile User Configuration sind jetzt editierbar und sollten insbesondere bei ***Compiler***, ***Linker*** und ***Archiver*** auf das aktuelle Installationsverzeichnis von CCSv5 aktualisiert werden.

Das schaut dann z.B. wie in den Abbildungen 20 - 22:

Reiter Compiler:
Compiler: C:\ti\ccsv5\tools\compiler\c2000_6.1.0\bin\cl2000
Arguments: -I"C:\ti\ccsv5\tools\compiler\c2000_6.1.0\include" -fr"[|||
MW_XMK_DERIVED_PATH_REF|||]"

Reiter Linker:
Linker: C:\ti\ccsv5\tools\compiler\c2000_6.1.0\bin\cl2000

Reiter Archiver:
Archiver: C:\ti\ccsv5\tools\compiler\c2000_6.1.0\bin\ar2000

A, Einstellungen unter ***Compiler***

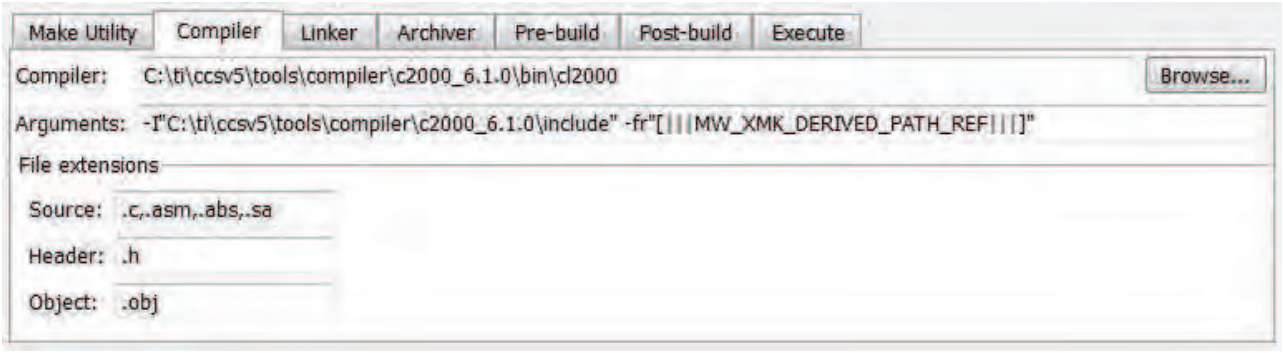


Abbildung 20: Compiler

B, Einstellungen unter ***Linker***

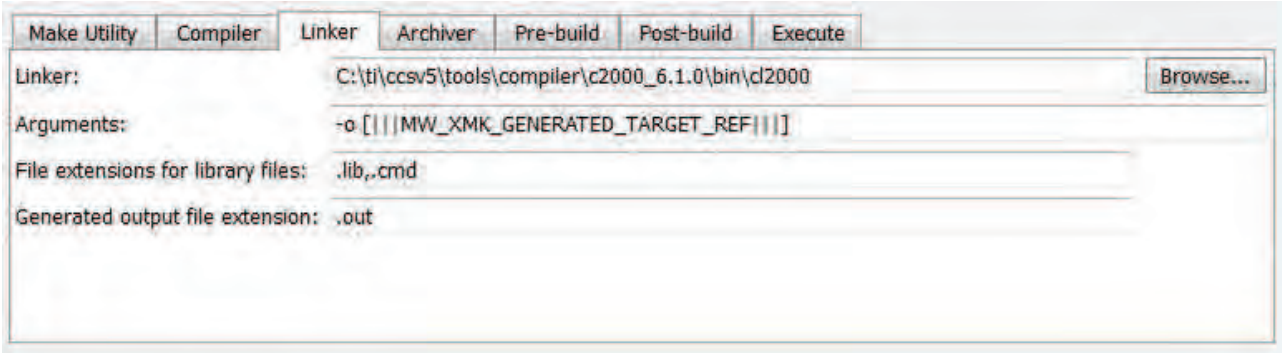


Abbildung 21: Linker

C, Einstellungen unter ***Archiver***

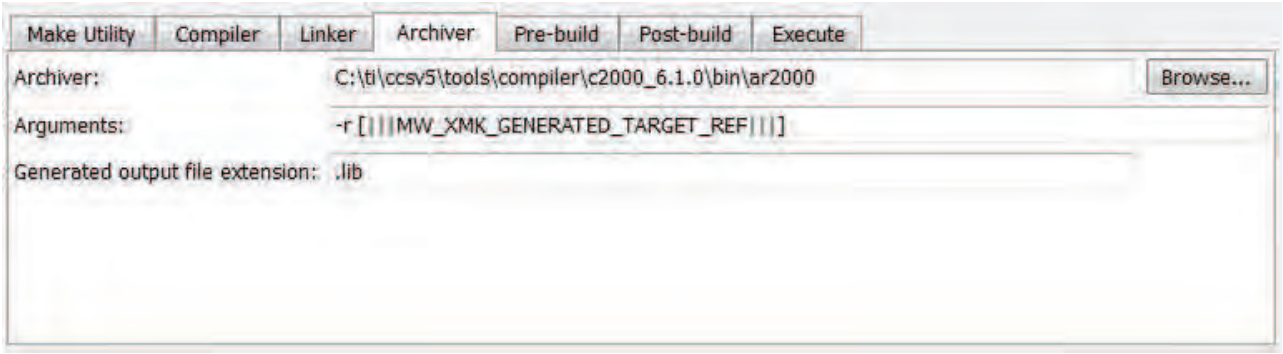


Abbildung 22: Archiver

Das waren jetzt alle Einstellungen für die Erstellung von ***makefiles*** zur Zusammenarbeit von MATLAB/Simulink mit der Entwicklungsumgebung Code Composer Studio v5. Ein Klick auf ***OK*** und das Fenster kann geschlossen werden. Nun kann mit MATLAB/Simulink gearbeitet werden.

4.2.2. CCSv5 Konfiguration mit einem Beispielprogramm

Der folgende Abschnitt beschreibt die Konfiguration in der CCSv5 Entwicklungsumgebung. Falls CCSv5 zum ersten Mal gestartet wird, folgt eine Anfrage nach dem Anlegen eines Workpace. Eine mögliche Auswahl ist ***C:\Users\Benutzername\workspace_v5_2*** (Abbildung 23).

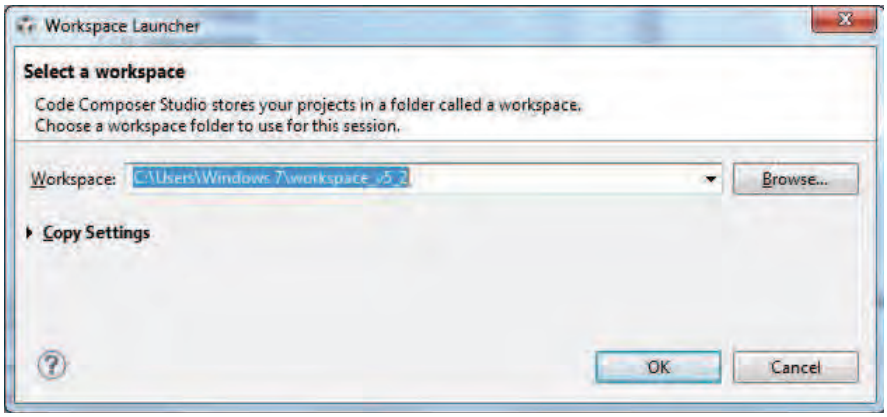


Abbildung 23: Workspace

Nachdem CCSv5 gestartet und der Workspace angelegt wurde, wird das erste Projekt angelegt. Der Aufruf dazu geht über ***File > New > CCS Project*** (Abbildung 24).

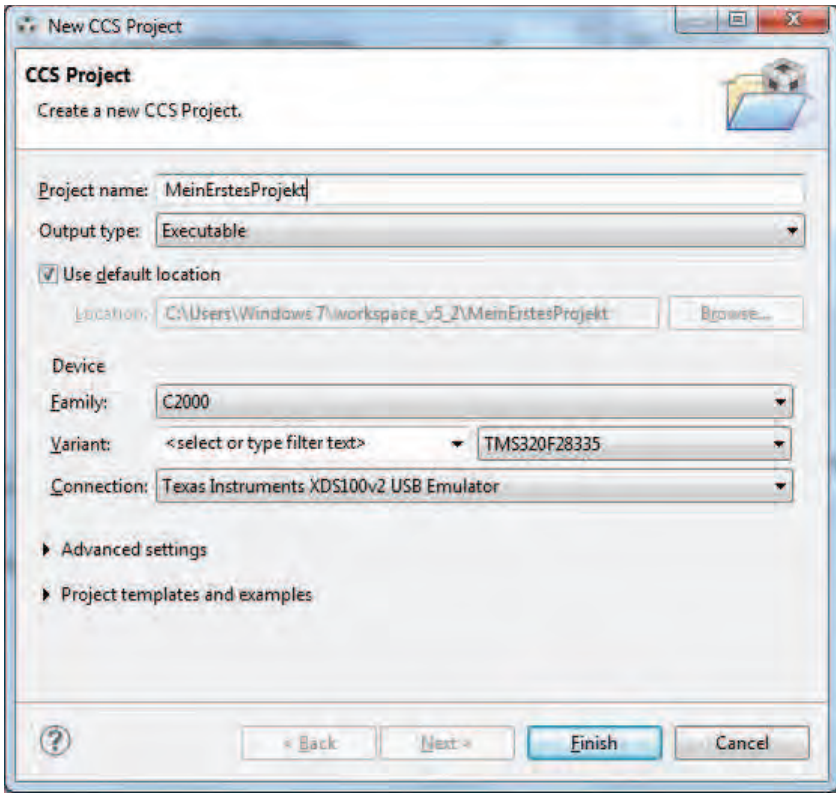


Abbildung 24: New CCS Project

Für ***Project name*** kann z.B. ***MeinErstesProjekt*** gewählt werden.
Das ***Output file*** sollte ein ***Executable*** sein und das Häckchen bei ***Use default location*** gesetzt werden.
In der Kategorie ***Device*** bitte unter ***Family*** die ***C2000 Family*** auswählen. Die verwendete ***Variante*** ist der ***TMS320F28335*** und die ***Connection*** erfolgt über ***Texas Instruments XDS100v2 USB Emulator***. Zur ***Target Configuration*** gelangt man später über ***View > Target Configuration*** und dann ein Doppelklick auf die *.ccxml-Datei im jeweiligen Projekt. Hier sollte später nochmals überprüft werden, ob die richtige Zielhardware eingestellt ist, bevor der Code übertragen wird.
Die Auswahl unter ***Advanced Settings*** und ***Project templates and examples*** kann einfach übernommen werden.
Die Einstellungen werden durch einen Klick auf ***Finish*** übernommen und das Projekt erstellt.
In Abbildung 25 ist das angelegte Projekt zu sehen.

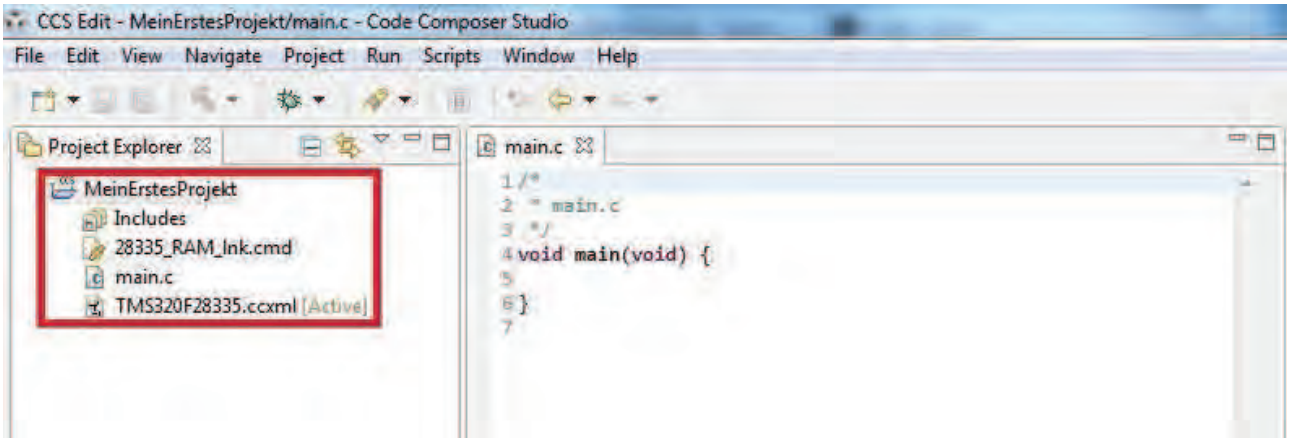


Abbildung 25: Angelegtes Projekt mit einer C-Code Vorlage von main.c

Als nächstes sollte die Größe des ***system stack*** für alle C Programme definiert werden [16]. Diese Einstellungen werden unter ***Properties*** (Klick mit der rechten Maustaste auf MeinErstesProjekt) vorgenommen. Unter Build > C2000 Linker > Basic Options >***Set C system stack size***: ***0x400*** eingeben (Abbildung 26).

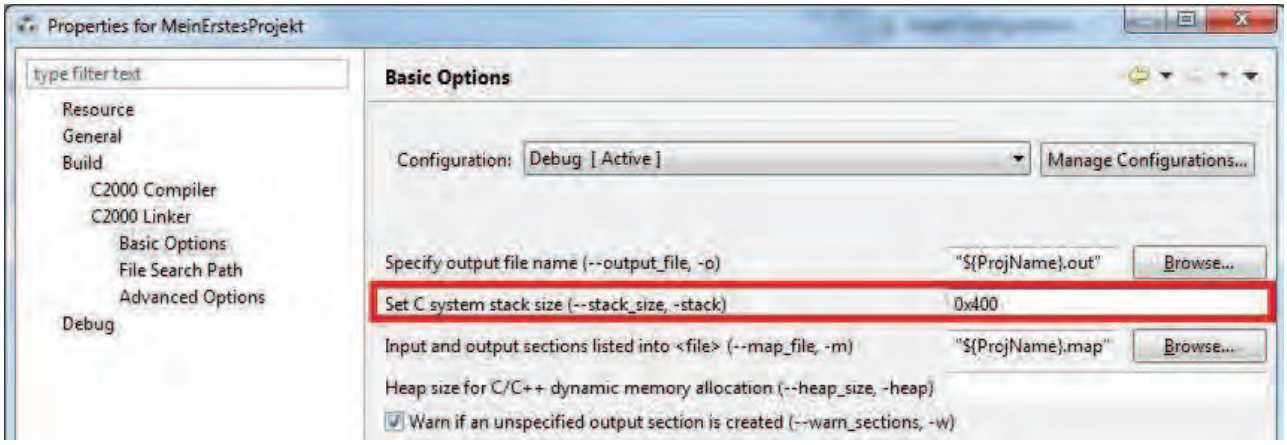


Abbildung 26: System Stack

Ein kleines Beispielprogramm wird nun in CCSv5 erstellt und anschließend auf dem Delfino DSP ausgeführt um die vorher vorgenommenen Einstellungen zu überprüfen, d.h. die Funktion der Zielhardware sicher zu stellen.

Das vorher angelegten Projekt findet sich im ***Project Explorer***. Hier kann eingesehen werden, welche Dateien automatisch von CCS angelegt sind. Im Projekt wurde eine C-Code Datei (main.c) angelegt, die durch einen Doppelklick aufgerufen wird. Der C-Code soll, wie in der Abbildung 27 zu sehen übernommen werden [16]. Damit ist ein erstes Programm erstellt.

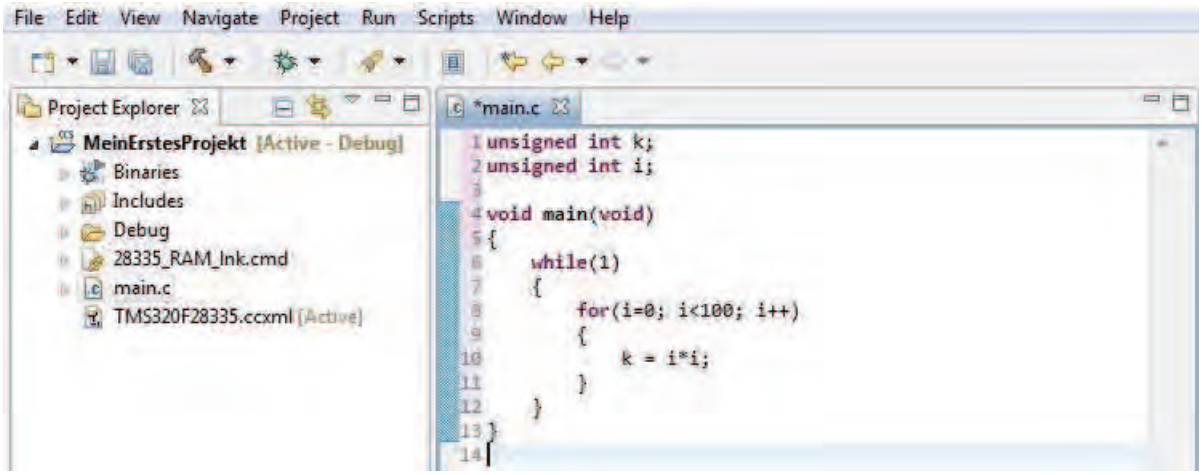


Abbildung 27: C-Code

Es ist ein sehr einfaches Programm mit zwei Variablen `i` und `k`. Das Programm besteht aus einer Endlos-Schleife (`while(1)`) in der ein for-Loop ausgeführt wird. Im for-Loop wird die Variable `i` von 0 bis 99 hochgezählt. Die Variable `k` ist dabei `i * i`.

Mit ***Project > Build Project*** wird der Code kompiliert und eine Ausgabedatei (*.out) erstellt. In der ***Console*** soll bei einem erfolgreichen Kompilieren ein ***Build Finished*** stehen.

Mit einem Klick auf ***Debug*** (Abbildung 28) werden gleich drei Schritte ausgeführt:

- Rebuild (Output-Datei erstellt)
- Connect Target (Verbindung zur Zielhardware hergestellt)
- Load Program (Programm geladen)

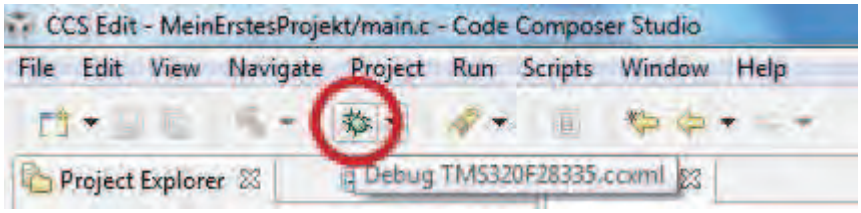


Abbildung 28: Debug

Durch den Klick auf ***Debug*** gelangt man in die ***CCS Debug Perpective***. Ein Indikator dafür, dass das Programm in diesem Fall richtig übersetzt und geladen wurde ist der blaue Pfeil, der auf die Zeile 8 (for-Loop) im Programm zeigt (Abbildung 29). Da erfolgt die erste Wertänderung. Mit ***View > Expressions*** wird ein Fenster ***Expressions*** aufgerufen in dem die Variablen i und k über ***Add nex expression*** angezeigt werden. Hier ist unter anderem der aktuellen Wert der Variablen aufgeführt.

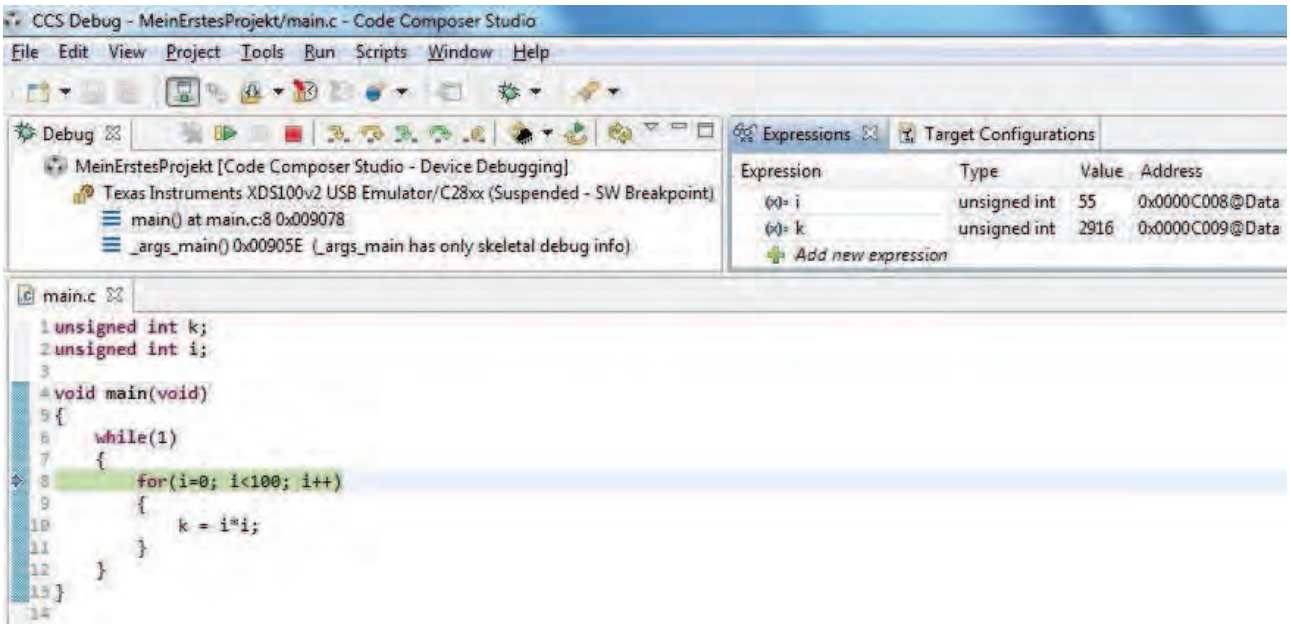


Abbildung 29: CCS Debug

Um die Wertänderung der Variablen nach jedem for-Loop Durchlauf beobachten zu können, werden Breakpoints gesetzt. So wird durch einen Doppelklick links neben der Zeile 10 (blauer Bereich) ein Breakpoint gesetzt. Durch Anklicken von ***Resume*** (alternativ F8-Taste) findet ein Durchlauf des Programms bis zum Breakpoint statt. Die damit einhergehende Wertänderung der Variablen ist unter ***Expressions*** (Abbildung 30) sichtbar.

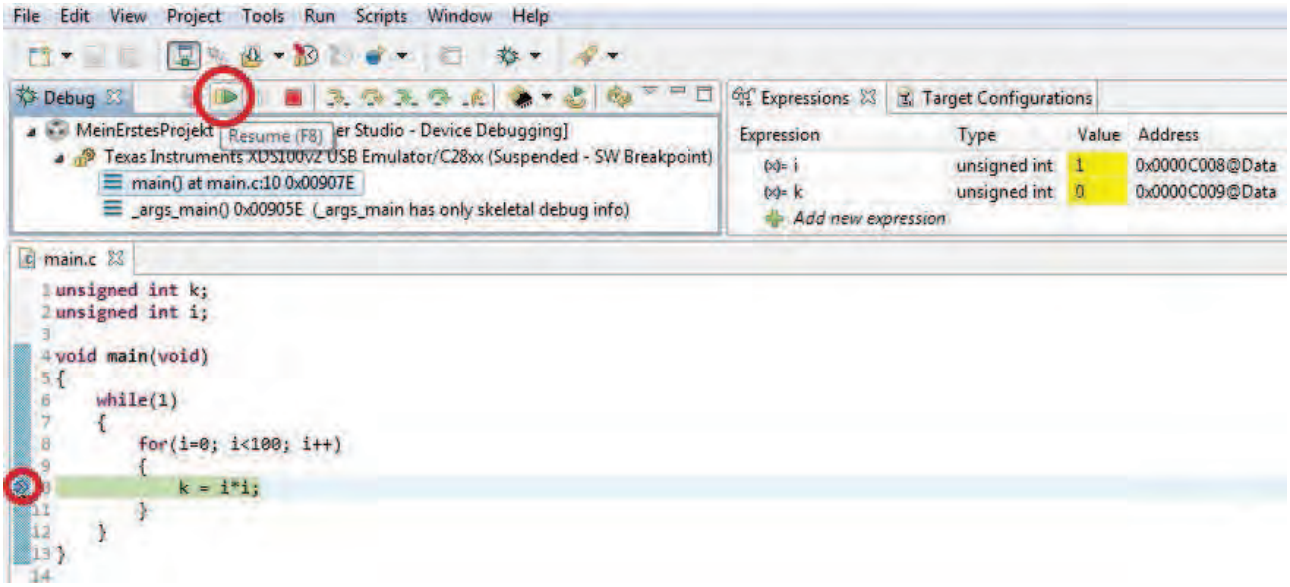


Abbildung 30: Breakpoint

4.3. Datenverarbeitung an einem Beispielmodel

Der folgende Abschnitt beschreibt die Code Erstellung (Code Generation) aus einem einfachen Beispielmodell in Simulink und den Import der erzeugten Datei in CCSv5 mit anschließender Übertragung auf die Zielhardware.

Nach dem Start von Simulink aus MATLAB sollte unter ***File > New > Model*** ein neues Fenster geöffnet werden. Als nächstes wird aus der Simulink Library der Baustein ***Target Preferences*** ausgewählt. Dieser findet sich unter ***Embedded Coder > Embedded Targets (***Abbildung 31).

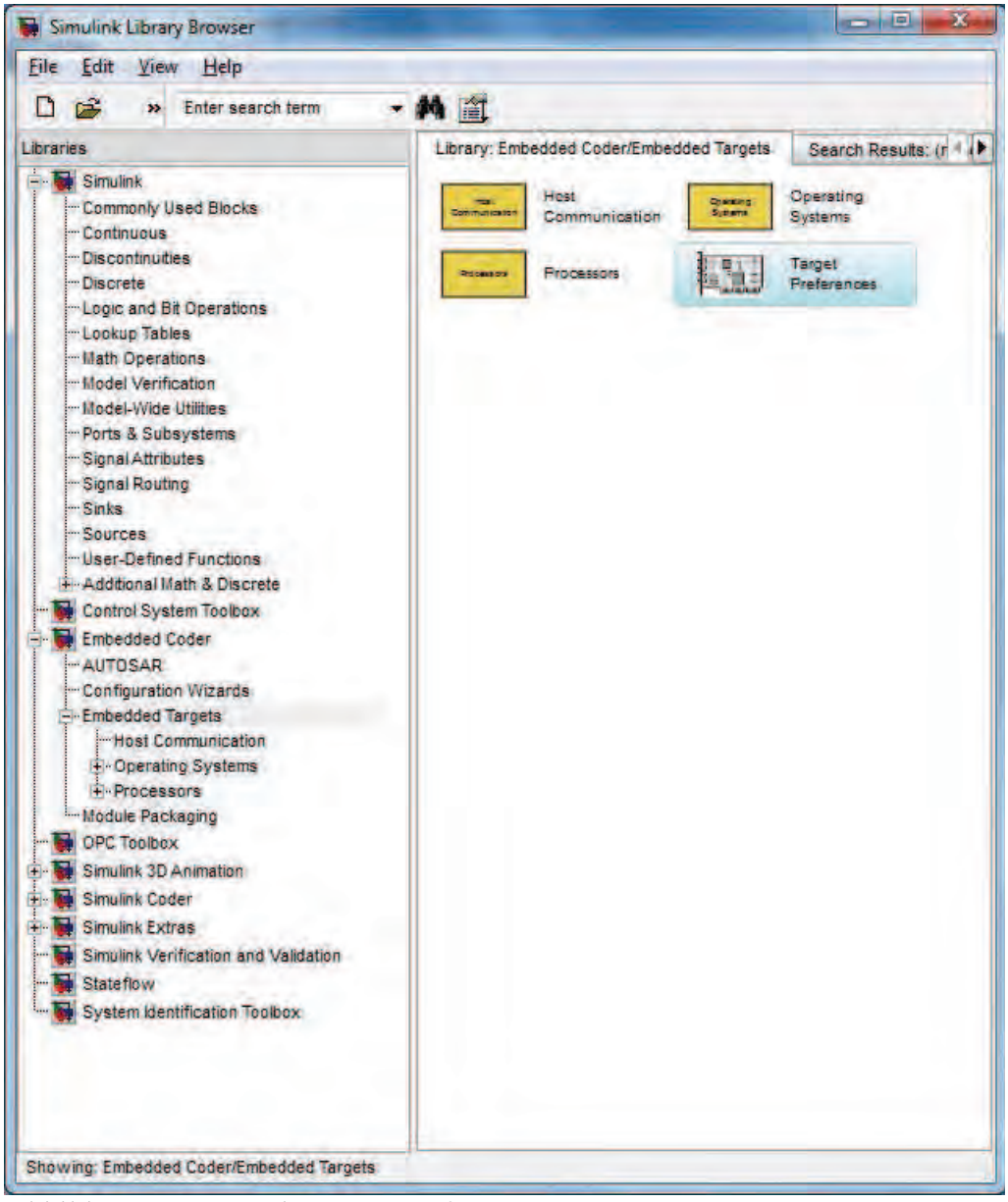


Abbildung 31: Baustein Target Preferences

Hierbei kann der Baustein einfach ins vorher neu angelegte Fenster per Drag&Drop gezogen werden. Sobald der Baustein sich im Simulink Modell befindet, sollten seine Einstellungen vorgenommen werden.

Zu den Einstellungen des Bausteins ***Target Preferences*** gelangt man bereits beim Einfügen in das Modell. Für spätere Änderungen kann man durch einen Doppelklick auf den Baustein, oder durch einen Rechtsklick mit anschließender Auswahl des Punktes ***Open Block*** die Einstellungen öffnen.

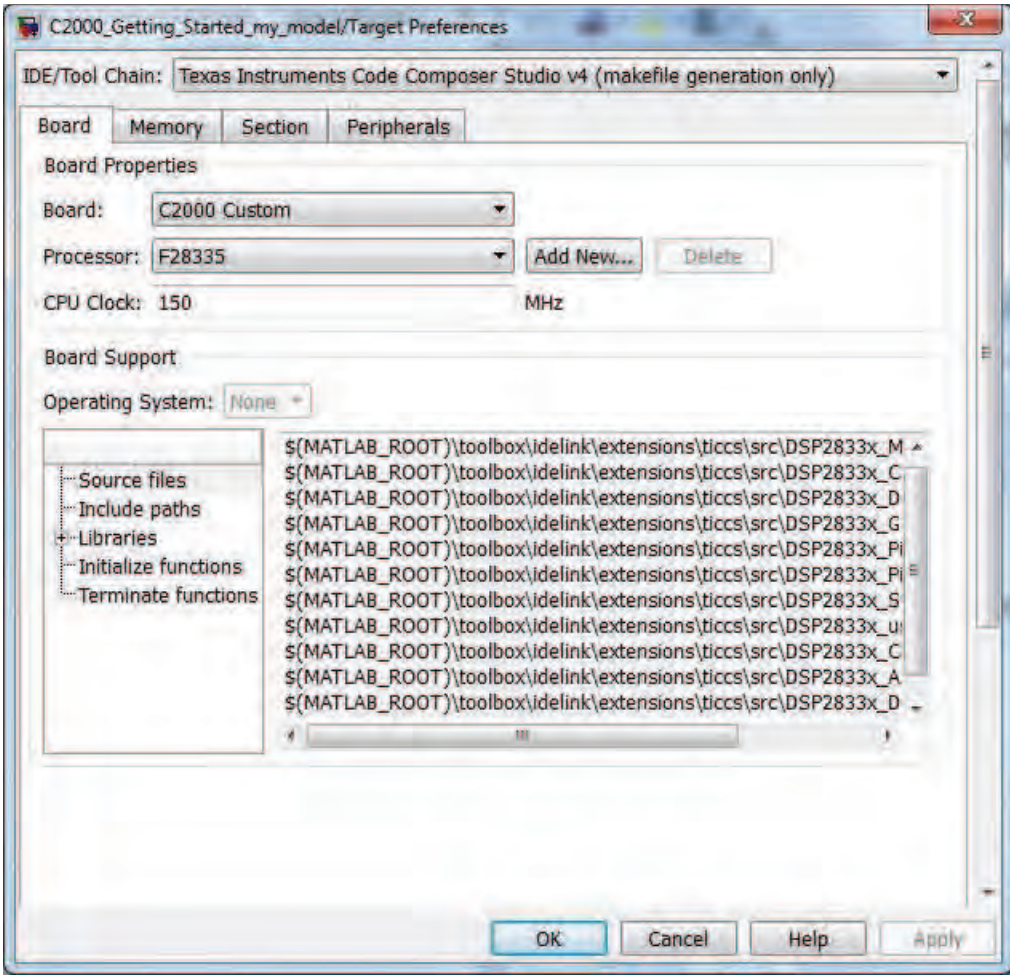


Abbildung 32: Target Preferences

Unter IDE/Tool Chain sollte Texas Instruments Code Composer Studio v4 (makefile generation only) (!) ausgewählt werden. Die Version 5 von CCS wurde noch nicht in die Menüführung einbezogen, aber die Einstellungen gelten auch für diese Version.
Nach einem Klick auf den Reiter ***Board*** sollte dort unter Board: ***C2000 Custom***, der Processor ***F28335*** und CPU Clock ***150*** ausgewählt werden.
Ein Klick auf ***Apply*** und ***OK*** beendet die Einstellungen (Abbildung 32).

Zur schnellen Veranschaulichung der Code Erstellung wird das in der folgenden Abbildung 33 verwendete einfache Modell einer blinkenden LED erstellt. Dazu wird aus der Simulink Library der Baustein ***Constant*** unter ***Simulink*** > ***Commonly Used Blocks*** und den ***Digital Output*** Baustein unter ***Embedded Coder*** > ***Embedded Targets*** > ***Processors*** > ***Texas Instruments C2000*** > ***C28x3x*** ausgewählt. Die Bausteine können einfach aus der ***Simulink Library*** per Drag&Drop in das Modellfenster gezogen werden. Um eine Verbindung zwischen den zwei Bautsteinen herzustellen, muss der erste Baustein angeklickt und anschließend bei gedrückter Strg-Taste der zweite Baustein angeklickt werden. Daraufhin sind die zwei Bausteine durch einen Pfeil verbunden.

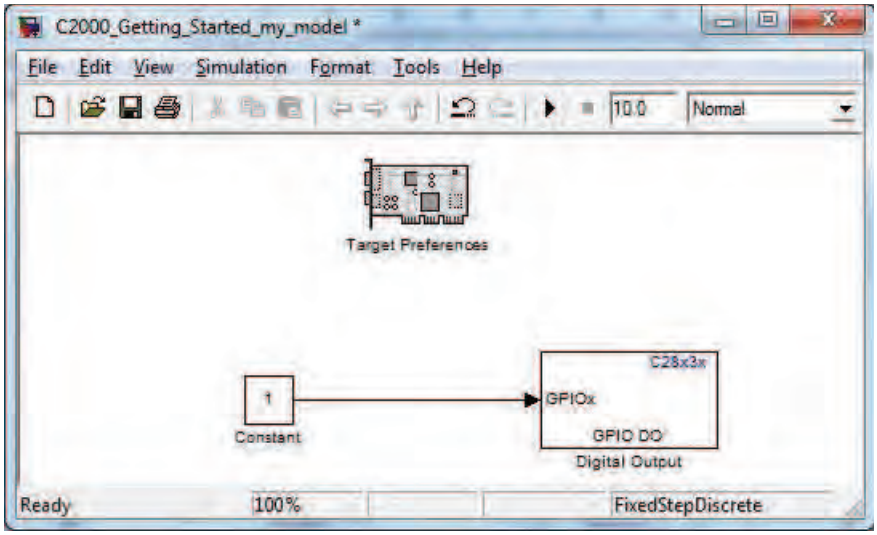


Abbildung 33: Simulink Modell

Die Bausteine ***Constant*** und ***Digital Output*** sollen noch vor dem Build des Modells konfiguriert werden. Dies kann durch einen Doppelklick auf den Baustein vorgenommen werden. Beim ***Constant*** Baustein eine ***1*** unter ***Constant Value*** eingetragen. Beim ***Digital Output*** im sich neu geöffneten Fenster (Abbildung 34) die GPIO Group: ***GPIO8~GPIO15*** auswählen und ein Häkchen bei ***GPIO9*** (LED1 liegt an diesem PIN) und ***Toggle GPIO9*** setzen. Ein Klick auf ***OK*** beendet die Konfiguration.

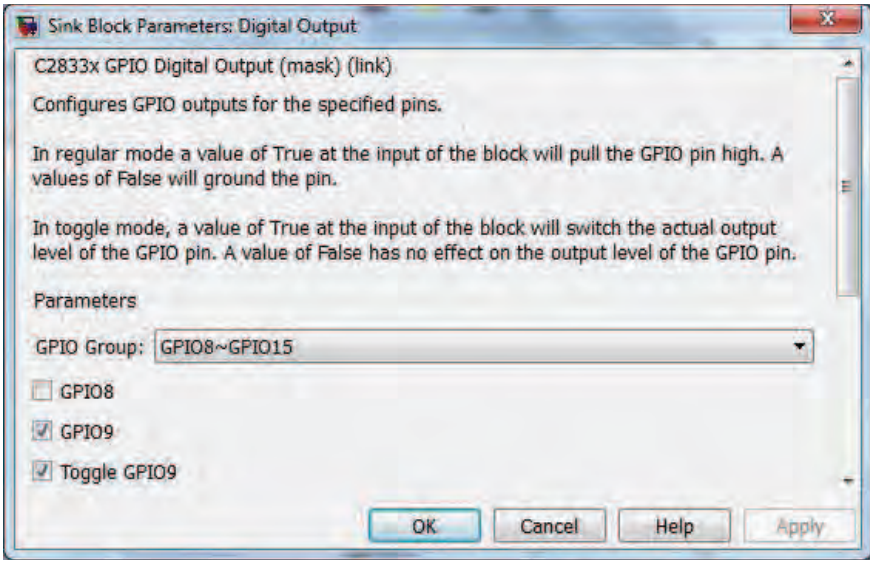


Abbildung 34: Digital Output

Sobald die Bausteine konfiguriert sind, kann aus dem Modell der Code erstellt werden. Dies erfolgt durch die Tastenkombination ***Strg + B*** oder durch einen Klick auf ***Tools > Code Generation > Build Model***. Der Build-Prozess kann im MATLAB Command Window beobachtet werden. Nach einem erfolgreichen Build steht im Command Window von MATLAB ein `Built done` und `Download done`. Der komplette Build findet sich im MATLAB Workspace in einem neuen Ordner mit dem Namen ***Modellname_ticcs***. Hier befindet sich die später verwendete *.out-Datei. In dieser Output-Datei ist die gesamte Ausgabe aus der Code Erstellung zusammengefasst. Das ist die Datei, die in CCS importiert wird. Des Weiteren kann der generierte C-Code (mehrere Dateien) separat eingesehen werden.

Das war die Code Erstellung in Simulink. Daran anknüpfend erfolgt der Import des Codes in CCSv5.

Für die Durchführung des Imports der Datei ist es erforderlich, dass die Zielhardware über ein USB-Kabel mit dem PC verbunden und eingeschaltet ist (Hauptschalter am Peripheral Explorer Kit Board).
Im CCSv5 wied unter ***View > Target Configurations*** das Fenster ***Target Configurations*** aufgerufen. In diesem Fenster befindet sich bei einem angelegten Projekt (z.B. MeinErstesProjekt) eine *.ccxml-Datei. Eine *.ccxml-Datei enthält die komplette Target Configuration. Durch einen Rechtsklick darauf und anschließend ein Klick auf ***Launch Selected Configuration*** wird die ausgewählte Konfiguration gestartet (Abbildung 35).

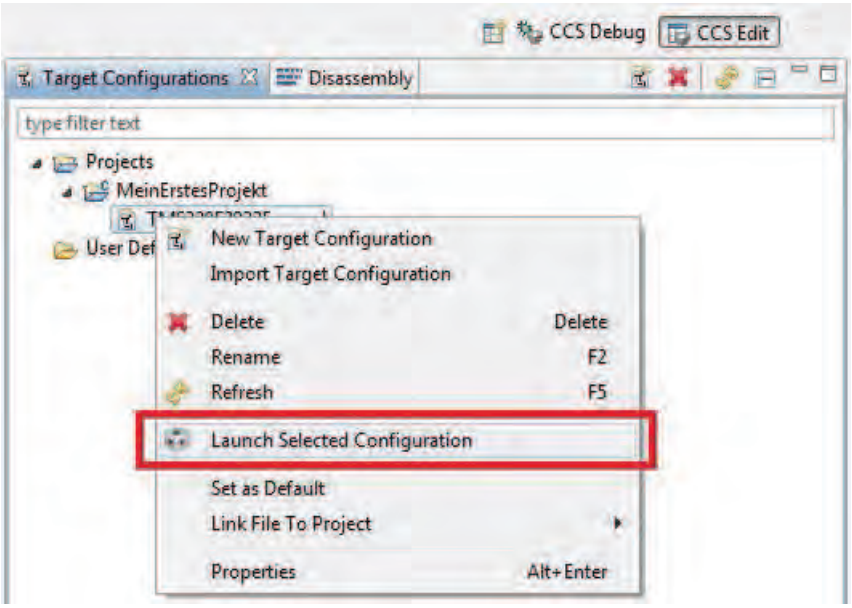


Abbildung 35: Launch Selected Configuration

Im sich neu geöffneten Fenster (Debug) befindet sich die vorhin ausgewählte *.ccxml-Datei. Nach einem Rechtsklick darauf, ***Connect Target*** auswählen (Abbildung 36). Die Verbindung ist nun mit der Zielhardware hergestellt.

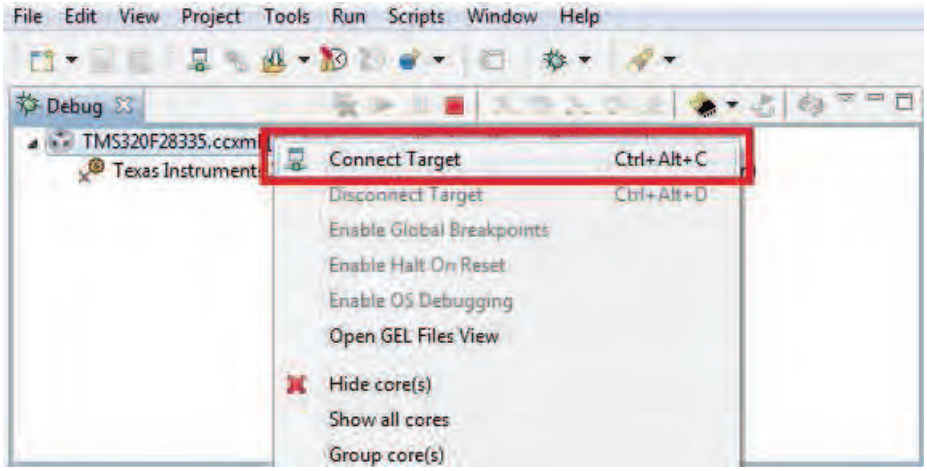


Abbildung 36: Connect Target

In der Menuleiste unter ***Run > Load > Load Program >Browse*** (Abbildung 36) die *.out-Datei auswählen. Diese befindet sich im MATLAB Workspace. Nach dem Auswählen und einem Klick auf Öffnen wird die *.out-Datei in CCSv5 geladen.

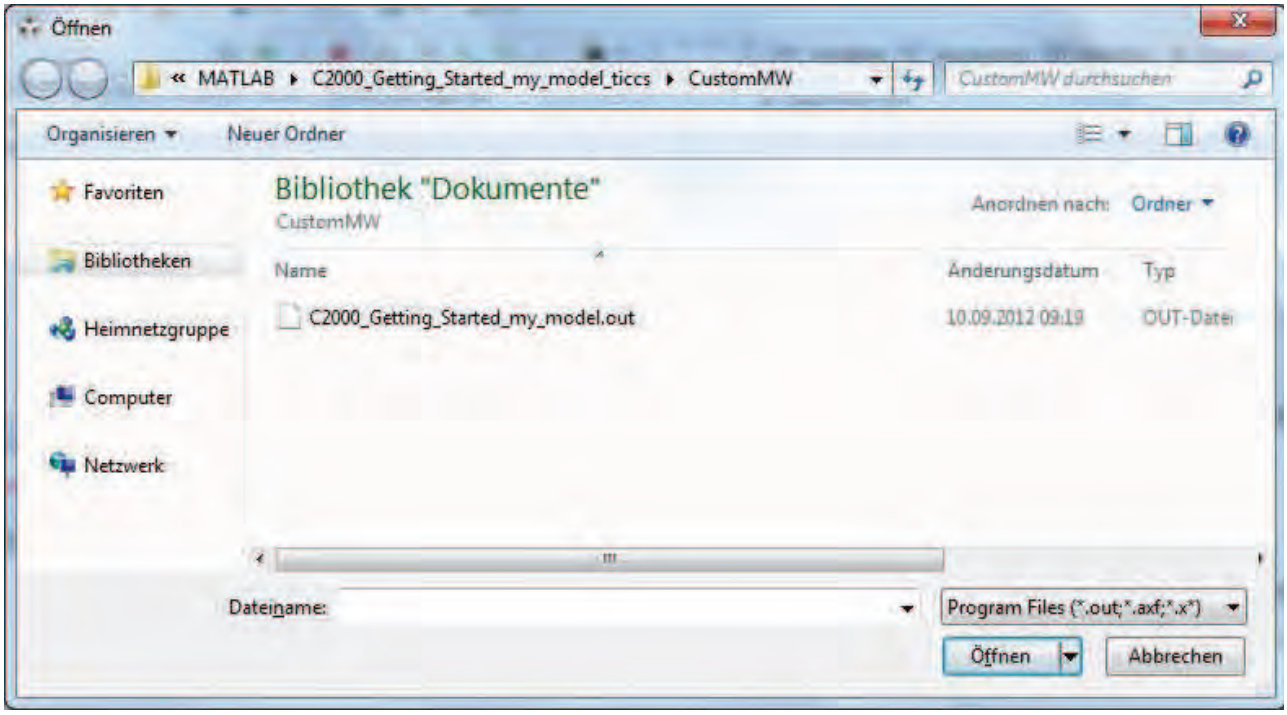


Abbildung 37: OUT-Datei Auswahl

Nach dem Drücken der Taste F8 oder dem Anklicken des ***Resume*** Buttons (Abbildung 38) in CCSv5 läuft das Programm einer blinken LED auf dem Peripheral Explorer Kit.

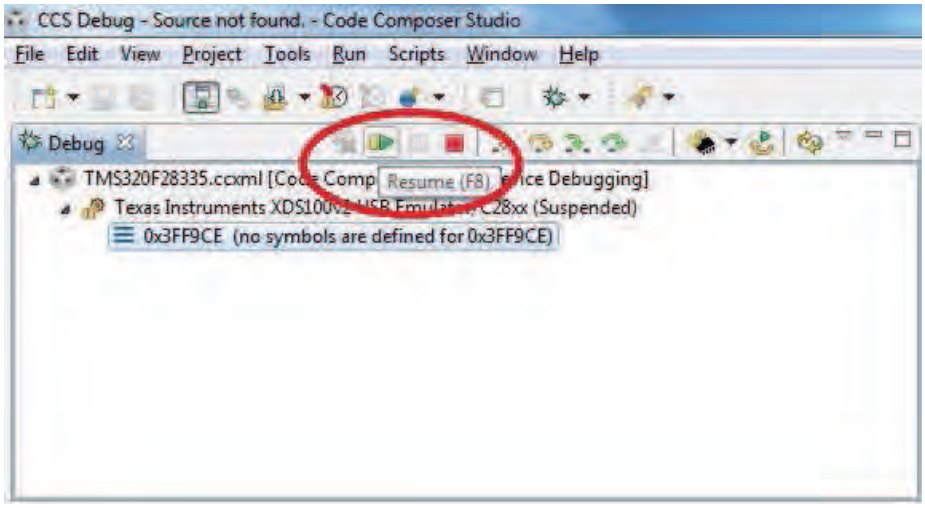


Abbildung 38: Resume

5. Demonstration der Toolchain

Für eine Beurteilung des modellbasierten Softwareentwurfs soll demonstriert werden, dass die vorgenommene Konfiguration in MATLAB und der Einsatz der damit erzeugten Output-Datei in CCSv5 auch zu einer Implementierung von Regler auf DSPs möglich ist.
Dazu wurde ein Versuch aus dem Regelungstechnik Labor der Hochschule Rosenheim im Studiengang Produktionstechnik, wie in Kapitel 3.3 beschrieben verwendet.

Die Demonstration in diesem Kapitel beginnt mit der Erstellung eines Simulink-Modells für den Praktikumsversuch Nachlaufregelung, gefolgt von der Code Erstellung aus dem Simulink-Modell, einem anschließenden Import der Output-Datei in CCSv5 sowie einer Übertragung auf den Delfino DSP auf der controlCARD.

Nach dem Start von MATLAB und dem Aufruf der Simulink Library (Abbildung 12) wird ein neues Modellfenster aufgerufen (Abbildung 39).

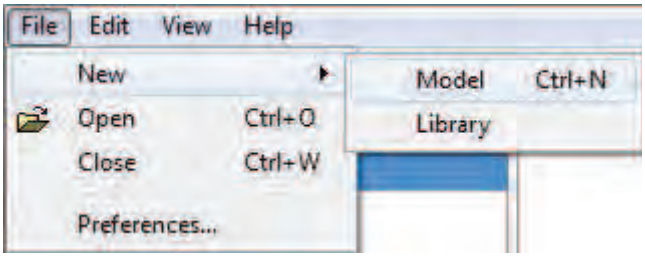


Abbildung 39: Neues Modell

Darin ist als erstes der ***Target Preference*** Baustein (Abbildung 31) eingefügt worden. Dieser Baustein ist für jede Code Erstellung aus einem Simulink-Modell für eingebettete Systeme notwendig. Nach dem Drag&Drop aus der Simulink Library in das Modellfenster wird automatisch nach den ***Konfigurations-Parameter*** (Abbildung 40) gefragt.

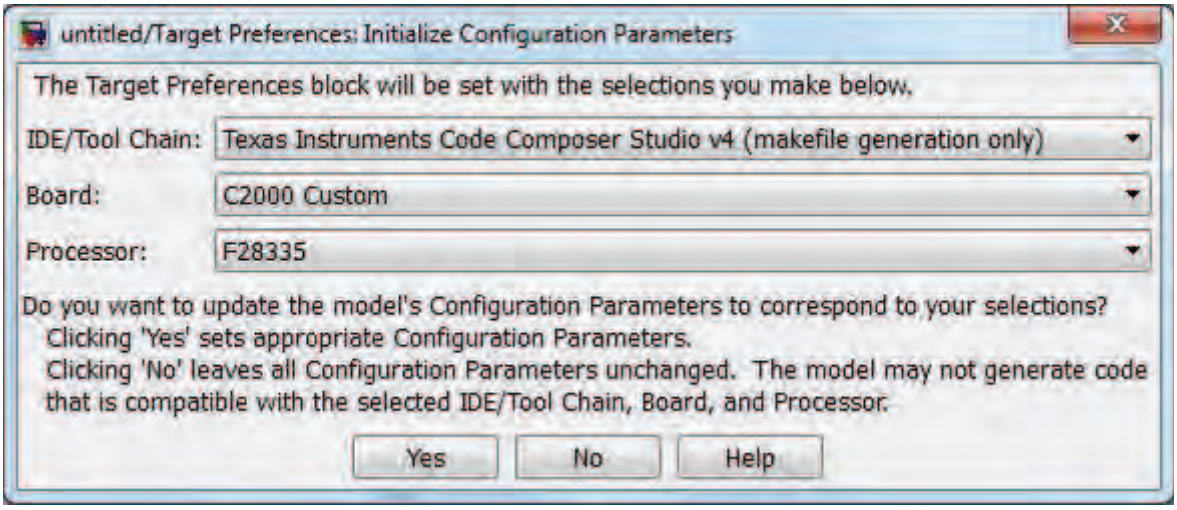


Abbildung 40: Konfigurations-Parameter

Der Abbildung 40 entsprechend sind folgende Einstellungen vorgenommen:

- IDE/Tool Chain: Texas Instruments Code Composer Studio v4 (makefile generation only)
- Board: C2000 Custom
- Processor: F28335

Für den Praktikumsversuch Nachregelung wird ein P-Regler verwendet. Die Verstärkung wurde mit dem Faktor 10 festgelegt. Der Regler hat einen mit dem Drehpotentiometer festlegbaren SOLL-Wert als Eingang und die Rückführung der Regelstrecke, also die IST-Wert Erfassung über das Schiebepotentiometer. Als Ausgang wird ein ePWM-Signal für den Motor bereitgestellt.
Die verschiedenen Bausteine des erstellten Simulink-Modells (Abbildung 41) werden anschließend im Einzelnen aufgeführt und deren Konfiguration detailliert beschrieben. Die Device-spezifischen Bausteine (ADC und ePWM) befinden sich wie in Kapitel 4.1.3 erklärt in der Simulink Library unter ***Embedded Coder > Embedded Targets > Processors > Texas Instruments C2000 > C28x3x*** (Abbildung 14). Alle anderen sind aus ***Simulink > Commonly Used Blocks*** (Abbildung 13).

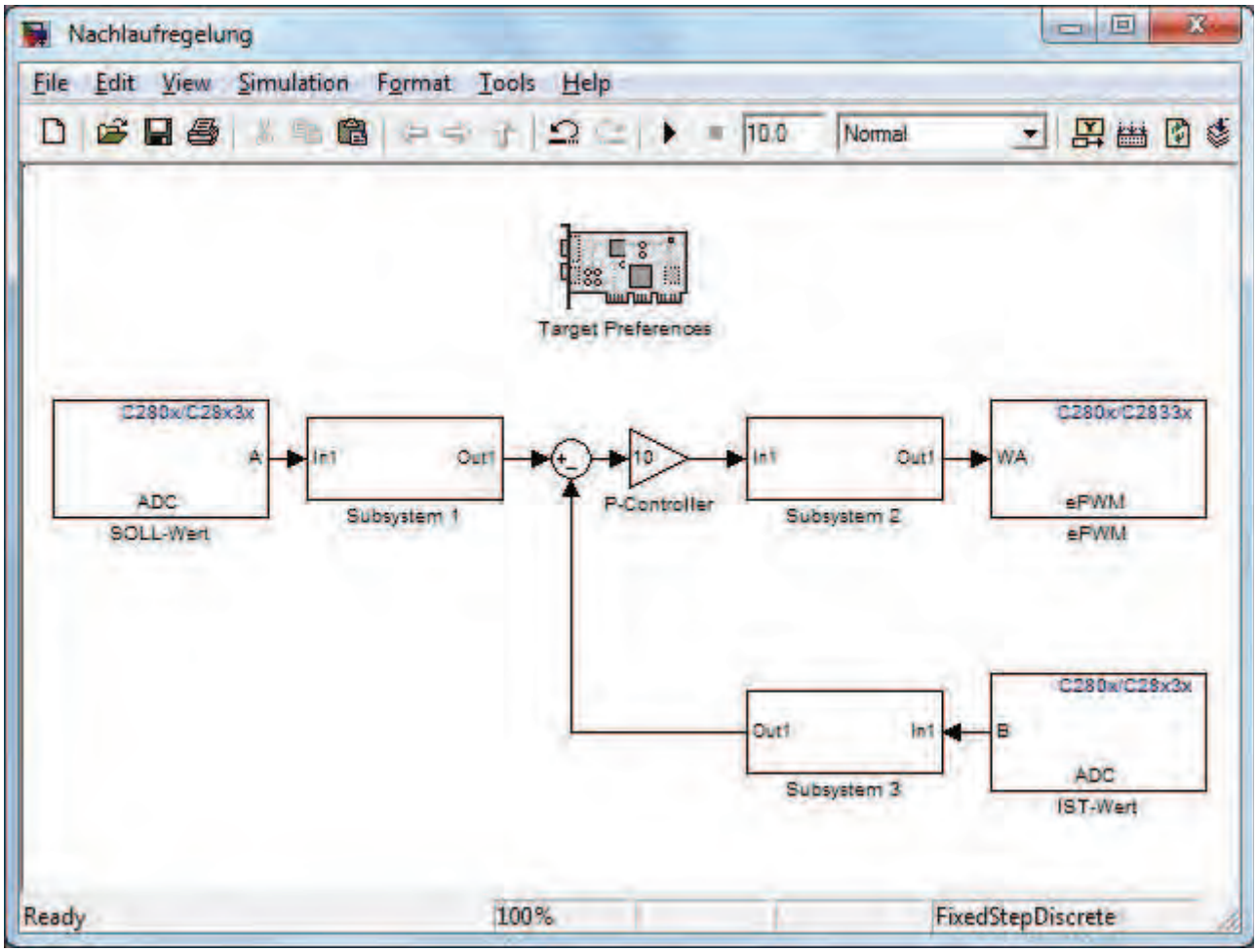


Abbildung 41: Simulink-Modell

Der SOLL-Wert wird mit dem Drehpotentiometer bestimmt, welches beim Peripheral Explorer Kit am analogen Eingang A1 liegt. Der gelieferte Wert ist vom Typ unit16. Die Einstellungen für den Baustein ADC sind in Abbildung 42 und 43 zu sehen getroffen worden.

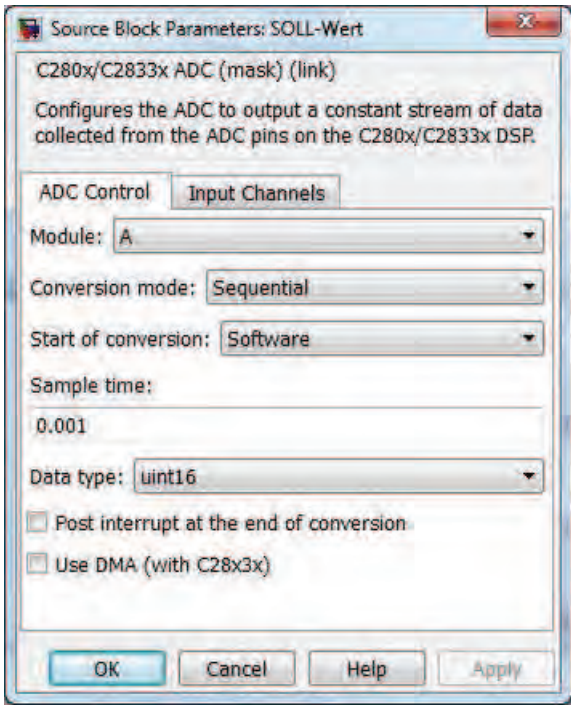


Abbildung 42: SOLL-Wert ADC Control

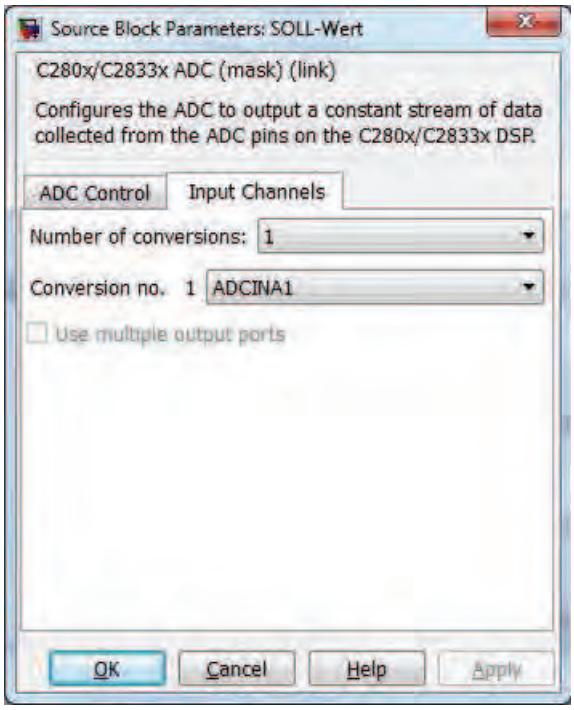


Abbildung 43: SOLL-Wert Input Channels

Der IST-Wert wird mit dem Schiebepotentiometer bestimmt, welches beim Peripheral Explorer Kit am analogen Eingang B2 angelegt wurde. Der gelieferte Wert ist vom Typ unit16. Die Einstellungen für den Baustein ADC sind in Abbildung 44 und 45 zu sehen.

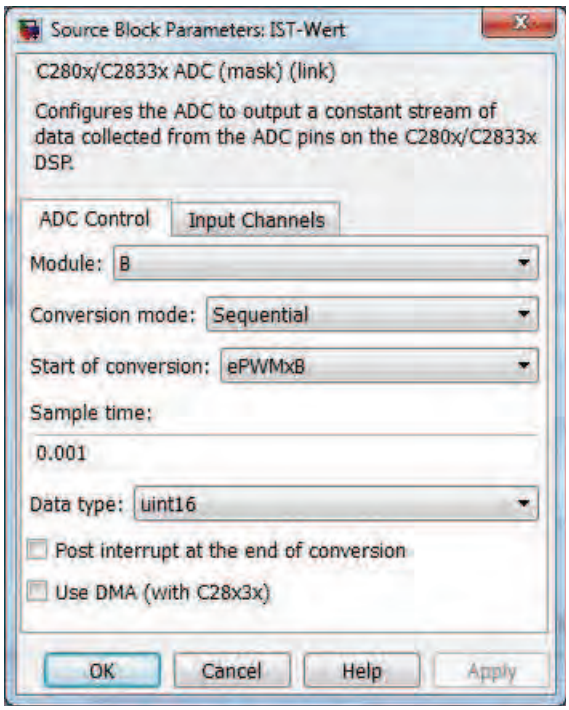


Abbildung 44: IST-Wert ADC Control

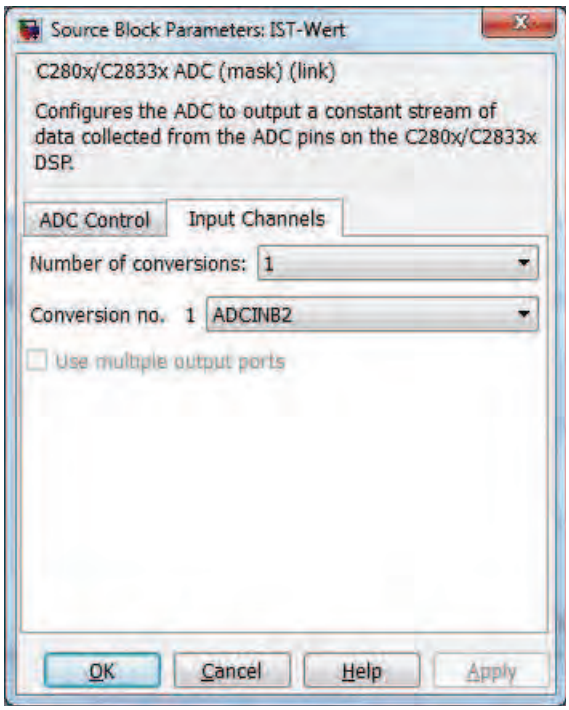


Abbildung 45: IST-Wert Input Channels

Die Konfiguration des ePWM Blocks erfolgt unter dem Reiter General. Die ***Timer period*** ist auf ***4095*** festgelegt. Das ist erforderlich, da der Drehpotentiometer eine 12-Bit Auflösung aufweist und infolgedessen eine PWM nur im Bereich zwischen 0 und 4095 realisierbar ist.

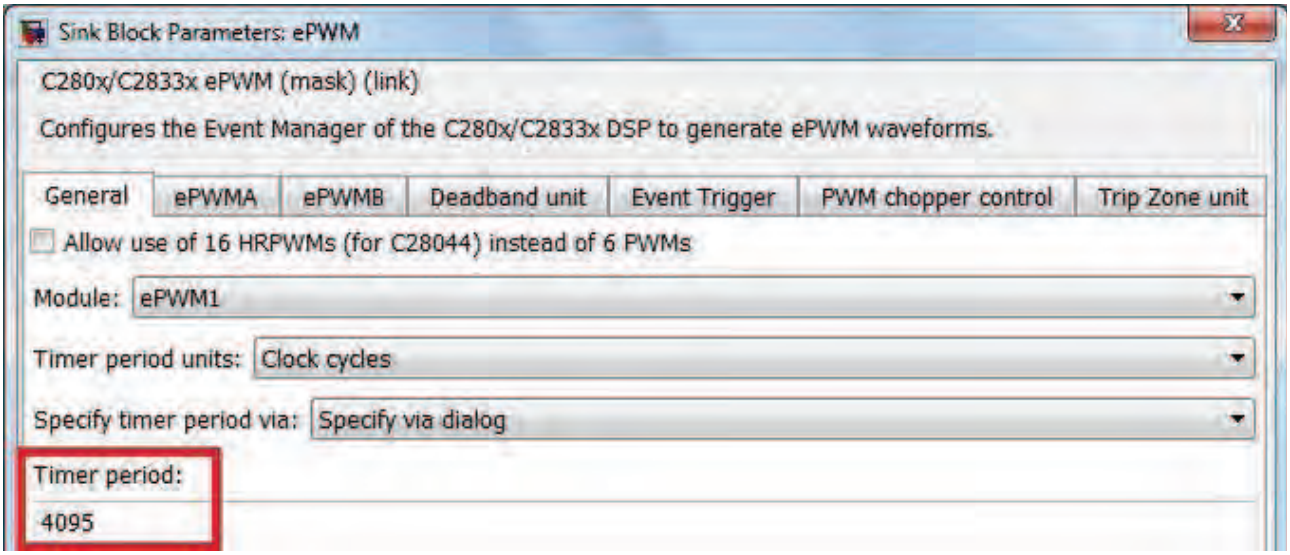


Abbildung 46: ePWM General

Um vom P-Regler den gelieferten Wert in ein PWM-Signal umzuwandeln, wird unter dem Reiter ***ePWMA*** die ***Input port*** Einstellung getroffen sowie der Anfangswert auf ***0*** gesetzt.

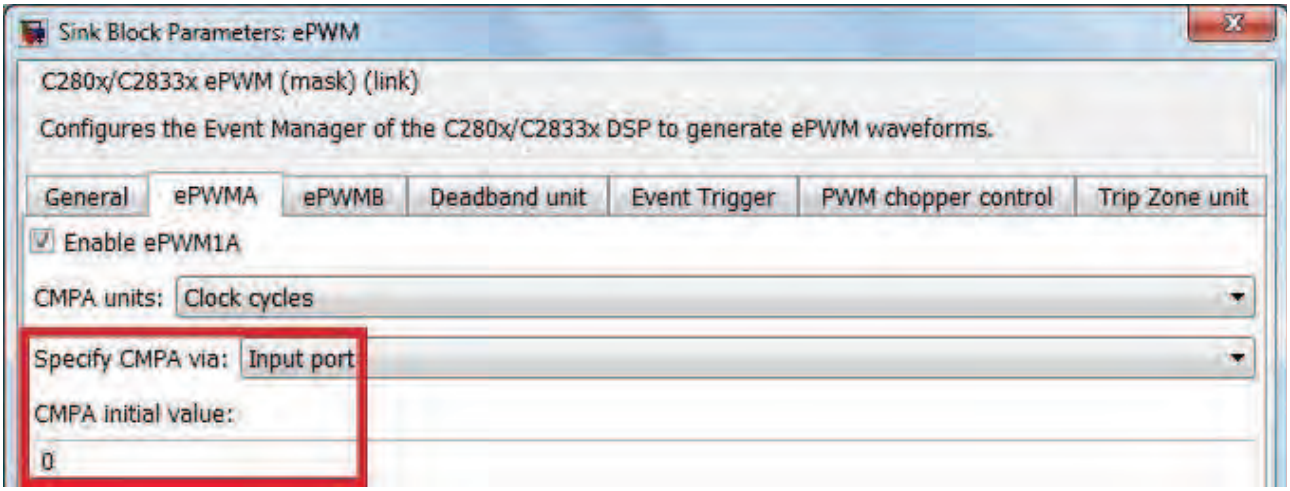


Abbildung 47: ePWMA

Die Subsysteme im Simulink-Modell (Abbildung 41) dienen der Umrechnung der von den Device-spezifischen Blöcken gelieferten Werte. Sie sind das „Software-Pendant“ zu den OPV-Schaltungen, die zwischen dem Peripheral Explorer Kit und den Ein- bzw. Ausgang an der Linearachse verschaltet sind (Kapitel 3.3). Damit wird im Modell so gerechnet, wie der P-Regler ausgelegt ist.

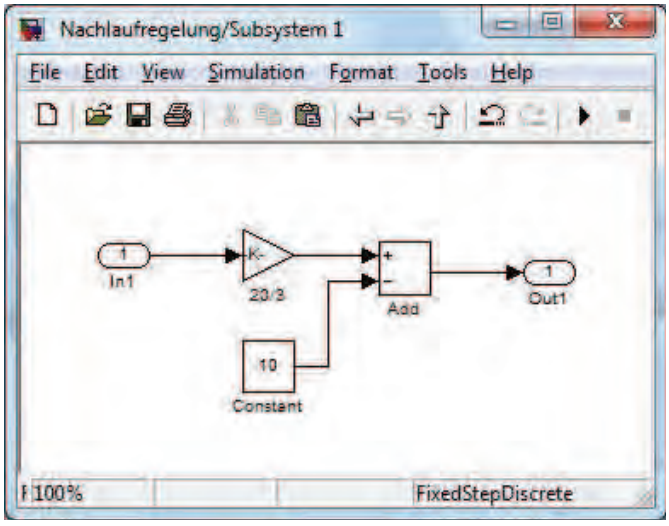


Abbildung 48: Subsystem 1

Die Bereichsanpassung aus dem ersten Subsystem führt aus dem Eingangswert der OPV zum Drehpotentiometer. Der Eingangswert der zwischen 0 und 3V liegt, wird in einen Ausgangswert zwischen -10V und 10V überführt.

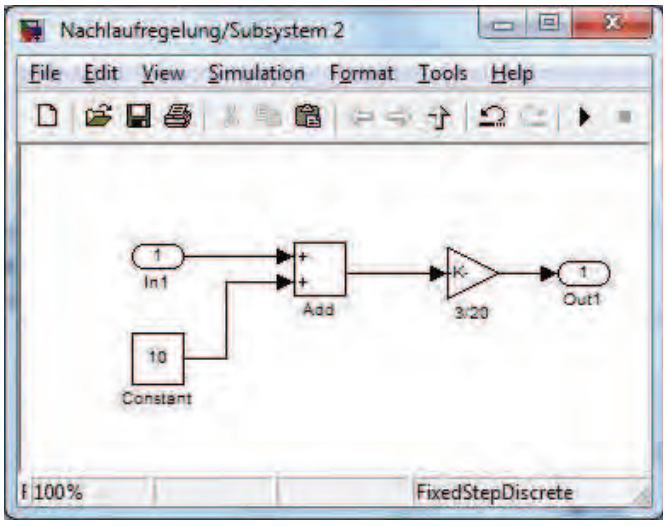


Abbildung 49: Subsystem 2

Im zweiten Subsystem wird aus der Ausgabe des P-Reglers eine Bereichsanpassung für den Ausgang auf 0 bis 3V vorgenommen. Dabei wird zuerst der Bereich von -10V bis 10V auf 0 bis 20V verschoben und dann über den Faktor 3/20 auf 0 bis 3V angepasst. Dies liefert den Eingang für den ePWM Block.

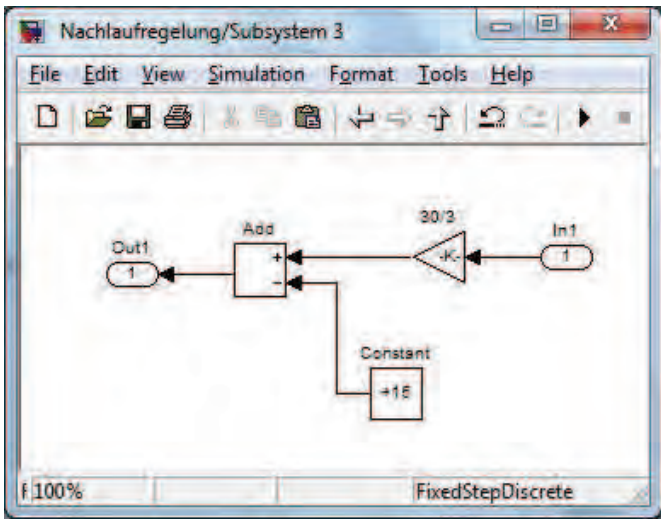


Abbildung 50: Subsystem 3

Das dritte Subsystem wandelt wiederum aus dem Eingangswert der OPV nach dem Schiebepotentiometer (0 bis 3V), einen für den Regler benötigten Bereich zwischen -15 und 15V durch.

Das Simulink-Modell für die Nachregelung ist erstellt und alle Parameter eingegeben. Dem folgend kommt die Code Erstellung. Dies wird ebenfalls in Simulink ausgeführt.

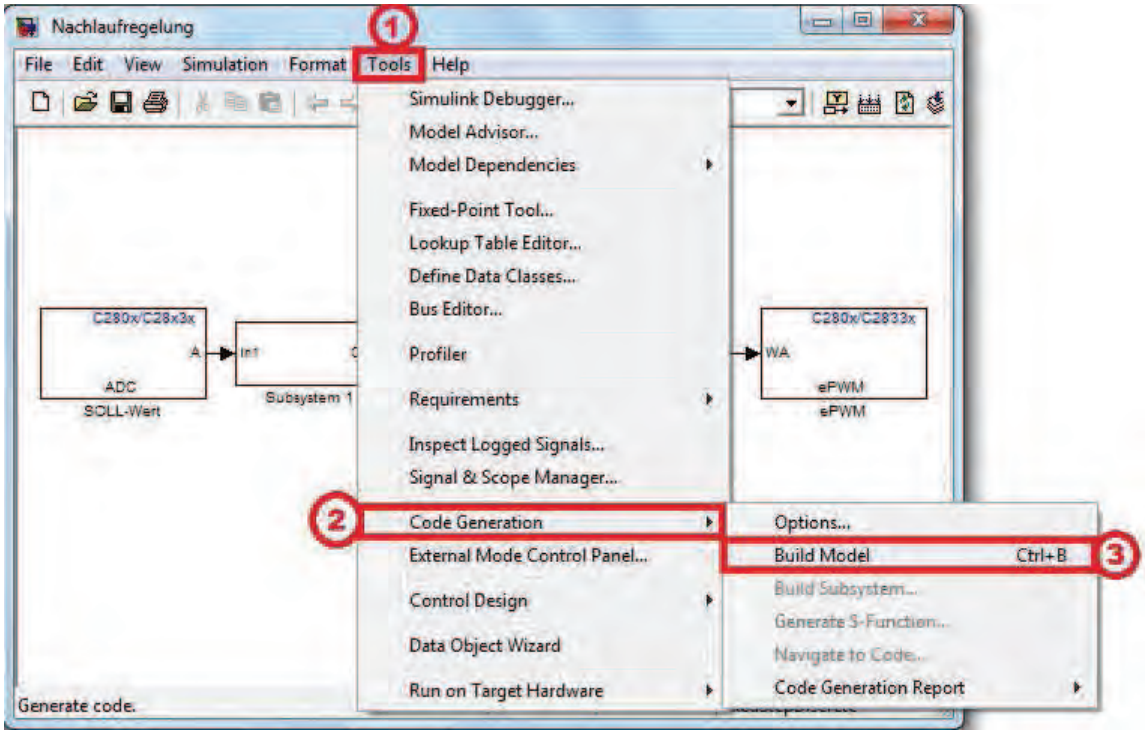


Abbildung 51: Build Model

Dazu wird in der Menüleiste unter ***Tools > Code Generation > Build Model*** (Abbildung 51) ausgewählt und der gesamte ***Build*** kann im Command Window von MATLAB verfolgt werden. Am Ende dieser Prozedur steht im Command Window ein ***Build done*** und ***Download done***.
Im Workspace von MATLAB findet sich umgehend im Projektordner eine Output-Datei, die im nächsten Schritt Verwendung findet.

Nach dem Start von CCSv5, dem Verbinden und Einschalten des Peripheral Explorer Kit über das USB-Kabel mit dem PC wird die aktuelle Zielhardware-Konfiguration ausgewählt (Launch Selected Configuration) und somit eine Verbindung hergestellt (Connect Target), wie in Kapitel 4.3 Abbildung 35 und 36.
Der Import der *.out-Datei erfolgt über die Menüleiste (***Run > Load > Load Program***). Nach der erfolgreichen Ausführung auf dem Peripheral Explorer Kit, kann mit dem Drehpotentiometer 1 die Sollwertvorgabe durchgeführt werden und der Schlitten wird vom Regler in die gewünschte Richtung nachgeführt.

6. Diskussion und Zusammenfassung

Abschließend werden die aus der modellbasierten Softwareentwicklung gewonnenen Erkenntnisse und Vorteile diskutiert, sowie Schwierigkeiten angesprochen, die bei der Umsetzung der Toolchain auftraten.
Da das Thema „modellbasierter Softwareentwurf“ neu ist und erst jetzt zunehmende Verwendung findet, ist die dafür notwendige Toolchain nicht vollständig und inkonsistent. Auf Seiten der Hersteller sollte die Zusammenarbeit der Software mit den aktuellen Versionen verbessert werden. Von The Mathworks, Inc. dem Hersteller von MATLAB/Simulink, werden die letzten zwei Versionen von Code Composer Studio (4 und 5) nicht vollständig unterstützt. Mit „nicht vollständig unterstützt“ ist gemeint, dass nach einem erfolgreichen Erstellen z.B. eines Simulink-Modells, nach der Code Erstellung der Import der Output-Datei in CCS manuell getätigt werden muss. Dies war bei der Version 3 von CCS nicht erforderlich, weil diese noch eine Eigenentwicklung von Texas Instruments war und nicht wie jetzt auf Ecplise basiert. Des weiteren ist in der kompletten Menüführung von MATLAB/Simulink die aktuellste Version 5 von CCS nicht berücksichtigt und somit konnte eine Kompatibilität nur durch sorgfältiges Testen bestätigt werden.

Die wichtigste Erkenntnis aus der Arbeit ist, dass die gesamte Programmierung auf einer höheren Ebene stattfindet. Das lauffähige Programm, welches sich auf dem Delfino DSP befindet, wird nur im Simulink-Modell editiert. Im erzeugten Quelltext wird nichts mehr hinzugefügt oder geändert. Dies wird einen weiteren Schritt im Entwicklungsprozess darstellen und soll durch den modellbasierten Ansatz vermieden werden.
Die Qualität der Software kann sichergestellt werden, da die Code Erstellung aus den Modellen standardisiert und zertifiziert werden kann und die Anwender sich somit hauptsächlich auf ihr Modelle konzentrieren können. Das ist ein großer Vorteil in Projekten, an denen Regelungstechniker beteiligt sind die ihre Regler mit MATLAB/Simulink gut umsetzen und sich nicht durch Tooling ablenken lassen wollen.

Einerseits kann die Qualität der Software durch die vorher erwähnte standardisierte und zertifizierte Code Erstellung aus den Modellen sichergestellt werden. Auf der anderen Seite muss ebenfalls der Code auf der Hardware Tests unterzogen werden. Dafür gibt es Verfahren wie Hardware-in-the-Loop, bei dem die zu testende Hardware an einem Tester, dem HiL-Simulator angeschlossen wird und deren Ein- und Ausgänge mit vorher erzeugten Testparameter beaufschlagt werden. Im Rahmen der Bachelorarbeit wurde ein Headerboard für den HILEVEL-Tester entwickelt, welches damit eine Schnittstelle für die controlCARD liefert.

Die Studie zur Toolchain des modellbasierten Softwareentwurf mit MATLAB/Simulink bereitete große Freude, da das Thema sehr spannend ist. Die damit gewonnenen Einblicke in die Theorie sowie der praktischen Umsetzung sind wichtig für die Zukunft.

7. Quellenangabe

[1] Berns, Schürmann, Trapp (2010): *Eingebettete Systeme - Systemgrundlagen und Entwicklung eingebetteter Systeme*, Vieweg+Teubner Verlag, 1.Auflage

[2] Stahl, Völter, Efftinge, Haase (2007): *Modellgetriebene Softwareentwicklung*, dpunkt.Verlag , 2. Auflage

[3] Pietruszka (2012): *MATLAB und Simulink in der Ingenieurspraxis - Modellbildung, Berechnung und Simulation*, Vieweg+Teubner Verlag, 3.Auflage

[4] Das V-Modell für die Systementwicklung http://www.cio.bund.de/DE/Architekturen-und-Standards/V-Modell-XT/vmodell_xt_node.html Stand: 28.09.2012

[5] Kalix, Bunzel (2005): *Integration modellbasierter Entwurfsverfahren in Softwareverifikation und -entwicklungsprozesse*

[6] TI C2000 Peripheral Explorer Kit, http://www.ti.com/tool/tmdsprex28335 Stand: 28.09.2012

[7] TMS320F28335 controlCARD, http://www.ti.com/tool/tmdscncd28335 Stand: 28.09.2012

[8] TMS320F28335, http://www.ti.com/product/tms320f28335 Stand: 28.09.2012

[9] Peripheral Explorer Kit Overview Quick Start (Rev.A), http://www.ti.com/litv/pdf/sprugm2a Stand: 28.09.2012

[10] Andreas Bernhard, PraktikumNachlaufregREV20120201.pdf im Regelungstechnik Labor

[11] TMS320F28335 Data Manual, 4.7 ADC-Module, http://www.ti.com/litv/pdf/sprs439m Stand: 28.09.2012

[12] http://www.mathworks.de/products/matlab/ Stand: 28.09.2012

[13] http://www.mathworks.de/products/simulink/ Stand: 28.09.2012

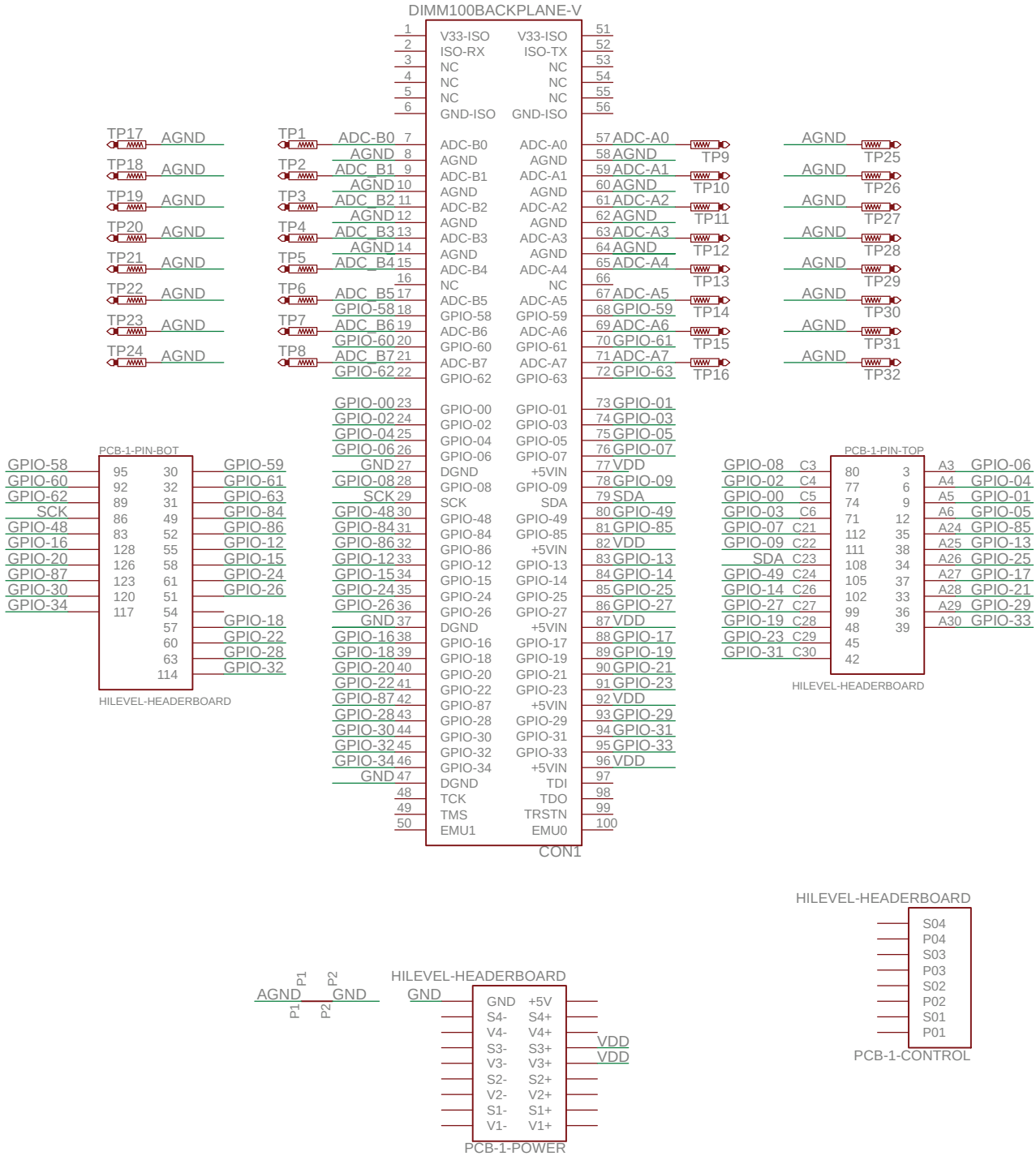
[14] http://www.mathworks.de/products/embedded-coder/ Stand: 28.09.2012

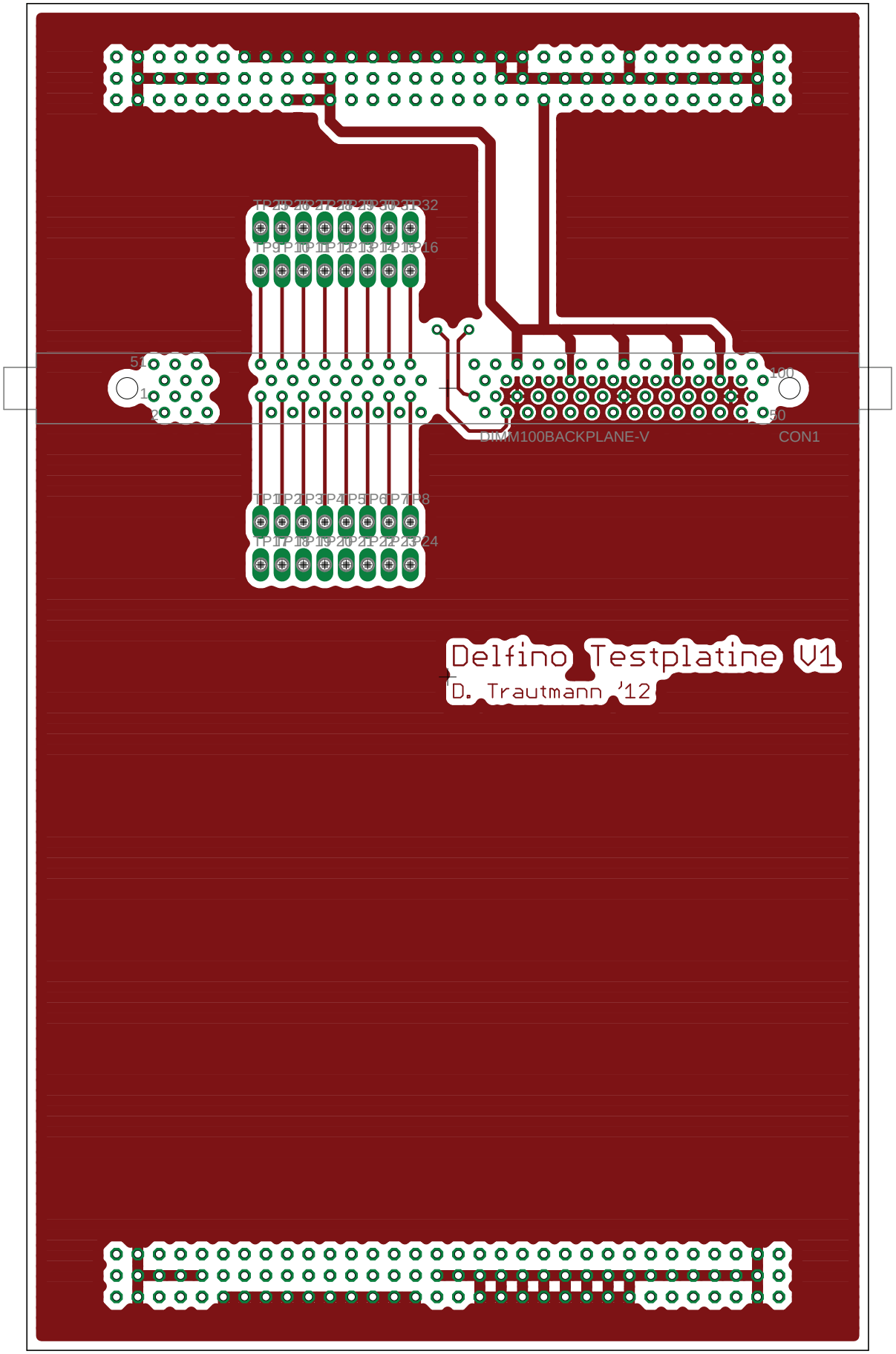
[15] http://processors.wiki.ti.com/index.php/Category:Code_Composer_Studio_v5 Stand: 28.09.2012

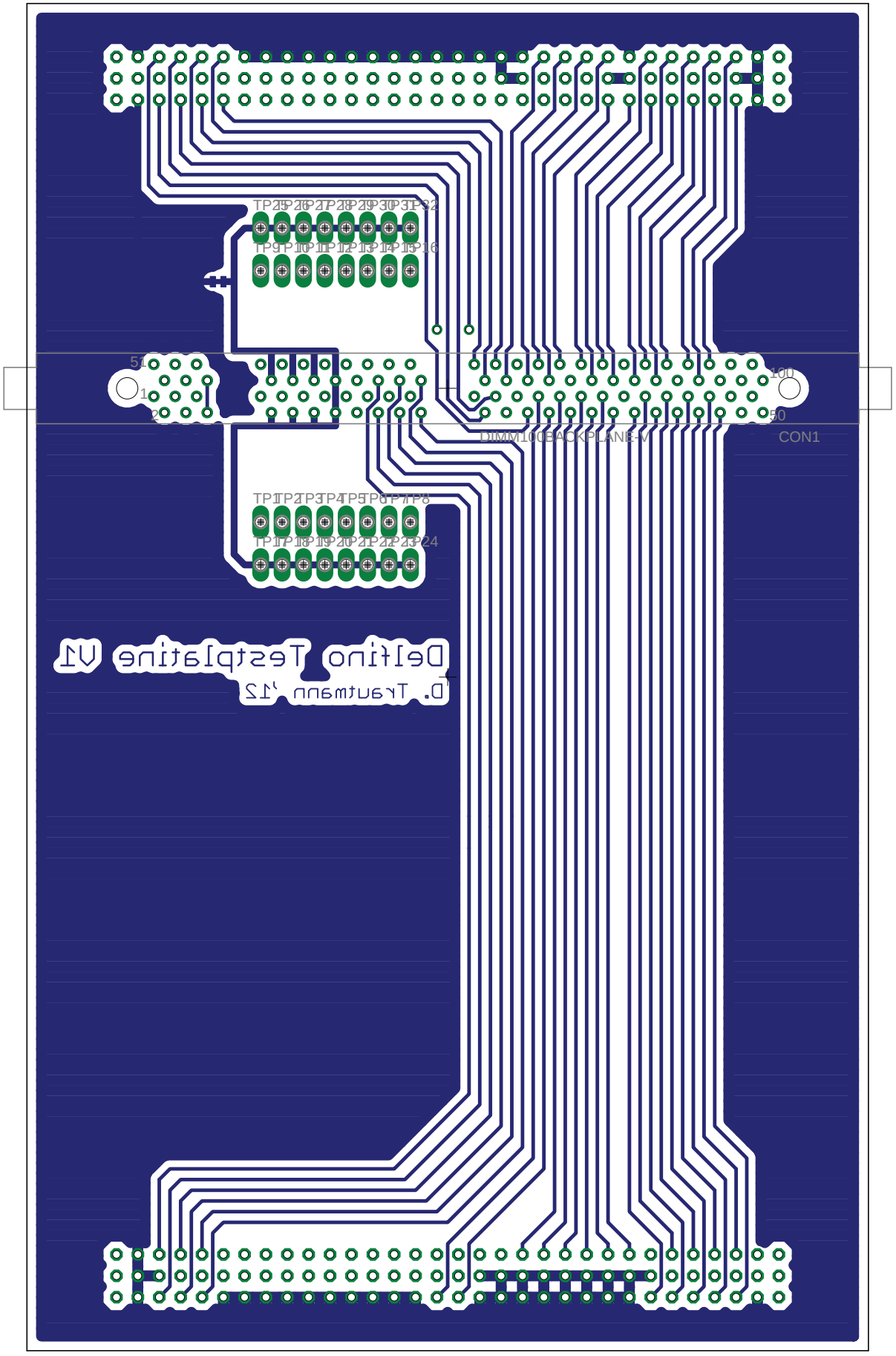
[16] C2000 Teaching Materials, Tutorials and Applications, Schulungs CD von TI, Module 3: F2833x Program Development Tools S. 13 / S. 14

A. Anhang Headerboard (Schaltplan + Layout)

Die für Hardware-in-the-Loop am HILEVEL-Tester benötigte Verbindung zwischen dem Tester und der controlCARD wurde im Rahmen der Bachelorarbeit „Studie zur Toolchain des modellbasierten Softwareentwurfs und Test eines C2000 Mikrocontroller mit MATLAB/Simulink“ entworfen. Der Schaltplan und das Layout (Vorder- und Rückseite) der Platine befinden sich im Anhang.


DIMM100BACKPLANE-V
V33-ISO
ISO-RX
NC
NC
NC
GND-ISO
V33-ISO
ISO-TX
NC
NC
NC
GND-ISO
ADC-B0
AGND
ADC-B1
AGND
ADC-B2
AGND
ADC-B3
AGND
ADC-B4
NC
ADC-B5
GPIO-58
ADC-B6
GPIO-60
ADC-B7
GPIO-62
ADC-A0
AGND
ADC-A1
AGND
ADC-A2
AGND
ADC-A3
AGND
ADC-A4
NC
ADC-A5
GPIO-59
ADC-A6
GPIO-61
ADC-A7
GPIO-63
GPIO-00
GPIO-02
GPIO-04
GPIO-06
DGND
GPIO-08
SCK
GPIO-48
GPIO-84
GPIO-86
GPIO-12
GPIO-15
GPIO-24
GPIO-26
DGND
GPIO-16
GPIO-18
GPIO-20
GPIO-22
GPIO-87
GPIO-28
GPIO-30
GPIO-32
GPIO-34
DGND
TCK
TMS
EMU1
GPIO-01
GPIO-03
GPIO-05
GPIO-07
+5VIN
GPIO-09
SDA
GPIO-49
GPIO-85
+5VIN
GPIO-13
GPIO-14
GPIO-25
GPIO-27
+5VIN
GPIO-17
GPIO-19
GPIO-21
GPIO-23
+5VIN
GPIO-29
GPIO-31
GPIO-33
+5VIN
TDI
TDO
TRSTN
EMU0
CON1
TP17 AGND
TP18 AGND
TP19 AGND
TP20 AGND
TP21 AGND
TP22 AGND
TP23 AGND
TP24 AGND
TP1 ADC-B0
TP2 ADC_B1
TP3 ADC_B2
TP4 ADC_B3
TP5 ADC_B4
TP6 ADC_B5
TP7 ADC_B6
TP8 ADC_B7
TP9 AGND
TP10 AGND
TP11 AGND
TP12 AGND
TP13 AGND
TP14 GPIO-59
TP15 GPIO-61
TP16 GPIO-63
AGND TP25
AGND TP26
AGND TP27
AGND TP28
AGND TP29
AGND TP30
AGND TP31
AGND TP32
PCB-1-PIN-BOT
GPIO-58 GPIO-59
GPIO-60 GPIO-61
GPIO-62 GPIO-63
SCK GPIO-84
GPIO-48 GPIO-86
GPIO-16 GPIO-12
GPIO-20 GPIO-15
GPIO-87 GPIO-24
GPIO-30 GPIO-26
GPIO-34 GPIO-18
GPIO-22
GPIO-28
GPIO-32
HILEVEL-HEADERBOARD
PCB-1-PIN-TOP
GPIO-08 C3 A3 GPIO-06
GPIO-02 C4 A4 GPIO-04
GPIO-00 C5 A5 GPIO-01
GPIO-03 C6 A6 GPIO-05
GPIO-07 C21 A24 GPIO-85
GPIO-09 C22 A25 GPIO-13
SDA C23 A26 GPIO-25
GPIO-49 C24 A27 GPIO-17
GPIO-14 C26 A28 GPIO-21
GPIO-27 C27 A29 GPIO-29
GPIO-19 C28 A30 GPIO-33
GPIO-23 C29
GPIO-31 C30
HILEVEL-HEADERBOARD
AGND P1
GND P2
P1
P2
HILEVEL-HEADERBOARD
GND
GND +5V
S4- S4+
V4- V4+
S3- S3+
V3- V3+
S2- S2+
V2- V2+
S1- S1+
V1- V1+
VDD
VDD
PCB-1-POWER
HILEVEL-HEADERBOARD
S04
P04
S03
P03
S02
P02
S01
P01
PCB-1-CONTROL
TITLE: TestplatineV1
Document Number:
REV:
Date: 06.09.2012 08:05:17
Sheet: 1/1


Delfino Testplatine V1
D. Trautmann '12
DIMM100BACKPLANE-V
CON1


TP25 TP26 TP27 TP28 TP29 TP30 TP31 TP32
TP9 TP10 TP11 TP12 TP13 TP14 TP15 TP16
51
1
2
100
50
DIMM100BACKPLANE-V
CON1
TP1 TP2 TP3 TP4 TP5 TP6 TP7 TP8
TP17 TP18 TP19 TP20 TP21 TP22 TP23 TP24
Delfino Testplatine V1
D. Trautmann '12

B. Anhang Praktikumsanleitung

Im Rahmen der Bachelorarbeit „Studie zur Toolchain des modellbasierten Softwareentwurfs und Test eines C2000 Mikrocontroller mit MATLAB/Simulink“ wurde eine Anleitung für das Regelungstechnik Labor der Hochschule Rosenheim erstellt.

Anleitung für den modellbasierten Softwareentwurf mit MATLAB/Simulink und Code Composer Studio v5

1

2

von Trautmann, Dietrich

im Studiengang Mechatronik

am 28.09.2012

1 Quelle: http://www.mathworks.com
2 Quelle: http://processors.wiki.ti.com/index.php/Category:Code_Composer_Studio_v5

Inhaltsverzeichnis

1. Einleitung..... 1

2. Verwendete Hardware..... 2
2.1. TI C2000 Peripheral Explorer Kit..... 2
2.2. Prakikumsversuch Nachlaufregelung..... 4

3. Verwendete Software …..... 7
3.1. MATLAB..... 7
3.2. Simulink..... 9
3.3. Embedded Coder..... 10
3.4. Code Composer Studio v5..... 11

4. Einstellungen..... 12
4.1. MATLAB Konfiguration..... 12
4.2. CCSv5 Konfiguration..... 16

5. Demonstration der Toolchain am Praktikumsversuch..... 20

6. Schluss..... 27

7. Quellenangabe..... 28

Abbildungsverzeichnis

Abbildung 1 : Peripheral Explorer Kit mit der controlCARD..... 2
Abbildung 2 : Peripherie [4]..... 3
Abbildung 3 : Linearachse..... 4
Abbildung 4 : Blockschaltbild Linearachse [5]..... 4
Abbildung 5 : Schaltplan Teil 1..... 6
Abbildung 6 : Schaltplan Teil 2..... 6
Abbildung 7 : MATLAB Oberfläche..... 7
Abbildung 8 : MATLAB und Toolboxen Version..... 8
Abbildung 9 : Simulink Aufruf in MATLAB..... 9
Abbildung 10 : Simulink Library Browser..... 9
Abbildung 11 : Embedded Coder in der Simulink Library..... 10
Abbildung 12 : CCS Oberfläche..... 11
Abbildung 13 : Command Window Befehleingabe..... 12
Abbildung 14 : XMakefile User Configuration..... 12
Abbildung 15 : XMakefile User Configuration: Tool Directories..... 13
Abbildung 16 : New Configuration Name..... 14
Abbildung 17 : Compiler..... 15
Abbildung 18 : Linker..... 15
Abbildung 19 : Archiver..... 15
Abbildung 20 : Workspace..... 16
Abbildung 21 : New CCS Project..... 16
Abbildung 22 : Angelegtes Projekt mit einer C-Code Vorlage von main.c... 17
Abbildung 23 : System Stack..... 17
Abbildung 24 : C-Code..... 18

Abbildung 25 : Debug..... 18
Abbildung 26 : CCS Debug..... 19
Abbildung 27 : Breakpoint..... 19
Abbildung 28 : Neues Modell..... 20
Abbildung 29 : Konfigurations-Parameter..... 20
Abbildung 30 : Simulink Modell..... 21
Abbildung 31 : SOLL-Wert ADC Control..... 22
Abbildung 32 : SOLL-Wert Input Channels..... 22
Abbildung 33 : IST-Wert ADC Control..... 22
Abbildung 34 : IST-Wert Input Channles..... 22
Abbildung 35 : ePWM General..... 23
Abbildung 36 : ePWMA..... 23
Abbildung 37 : Subsystem 1..... 24
Abbildung 38 : Subsystem 2..... 24
Abbildung 39 : Subsystem 3..... 24
Abbildung 40 : Build Model..... 25
Abbildung 41 : Launch Selected Configuration..... 26
Abbildung 42 : Build Model..... 26

1. Einleitung

Die Anleitung für den modellbasierte Softwareentwurf mit MATLAB/Simulink und Code Composer Studio v5 wurde im Rahmen der Bachelorarbeit „Studie zur Toolchain des modellbasierte Softwareentwurf und Test eines C2000 Mikrocontroller mit MATLAB/Simulink“ von Dietrich Trautmann verfasst.

Sie sollte Studenten der Hochschule Rosenheim einen einfachen Einstieg in den modellbasierte Softwareentwurf ermöglichen. Mit dem modellbasierten Softwareentwurf können Regelungstechniker ihre mit MATLAB/Simulink erstellten Regler auf eine Zielhardware übertragen und ausführen.

Die Erstellung der Simulink-Modelle und die anschließende Übersetzung in eine auf der Zielhardware läuffähige Datei, erfolgt dabei mit dem Programm MATLAB/Simulink von The MathWorks, Inc. Der Embedded Coder, ebenfalls vom gleichen Hersteller, liefert alle dafür notwendigen Zielhardware-spezifischen Komponenten.
Mit der Entwicklungsumgebung Code Composer Studio Version 5 (CCSv5) von Texas Instruments (TI) wird der verwendete Mikrocontroller (Delfino TMS320F28335) konfiguriert und anschließend die vorher erzeugte Datei in die Entwicklungsumgebung geladen und auf den µC übertragen.

Dazu wird die verwendete Hardware, die benötigten Programme und ihre Konfigurationen beschrieben. Eine abschließende Demonstration der Code Erstellung und Ausführung an einem Praktikumsversuch rundet diese Anleitung ab.

Studenten der Hochschule Rosenheim sollten nach dieser Anleitung in der Lage sein, den modellbasierten Softwareentwurf umsetzen zu können.

2. Verwendete Hardware

Bei der Erstellung der Anleitung ist die nachfolgend beschriebene Hardware verwendet worden. Dabei wurde das TI C2000 Peripheral Explorer Kit gekauft und der Praktikumsversuch „Nachregelung (Linearachse)“ aus dem Regelungstechnik Labor der FH Rosenheim im Studiengang Produktionstechnik verwendet.

2.1. TI C2000 Peripheral Explorer Kit

Das Board des TI C2000 Peripheral Explorer Kit [1] enthält Peripherie z.B. LEDs, Drehpotentiometer, Temperatur Sensor, Anschlüsse für Audio-In und Headphone-Out und viel mehr für einen schnellen Einstieg in die Programmierung eines C2000 Mikrocontroller und zum Experimentieren. Über den 100-pin DIMM[3] Sockel können verschiedene controlCARDs eingesteckt werden. Die verwendete controlCARD [2] enthält den C2000 TMS320F28335 Mikrocontroller [3].

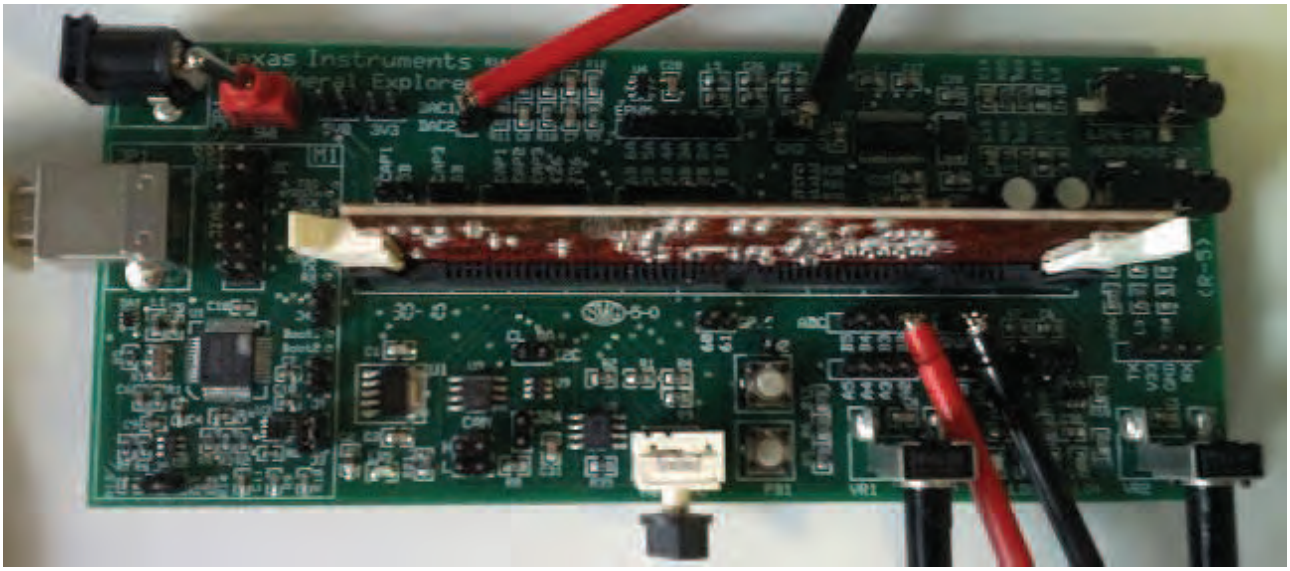


Abbildung 1: Peripheral Explorer Kit mit der controlCARD

In der Abbildung 1 ist das verwendete Peripheral Explorer Kit Board mit der eingesteckten controlCARD zu sehen.

3 DIMM: Dual In-Line Memory Module

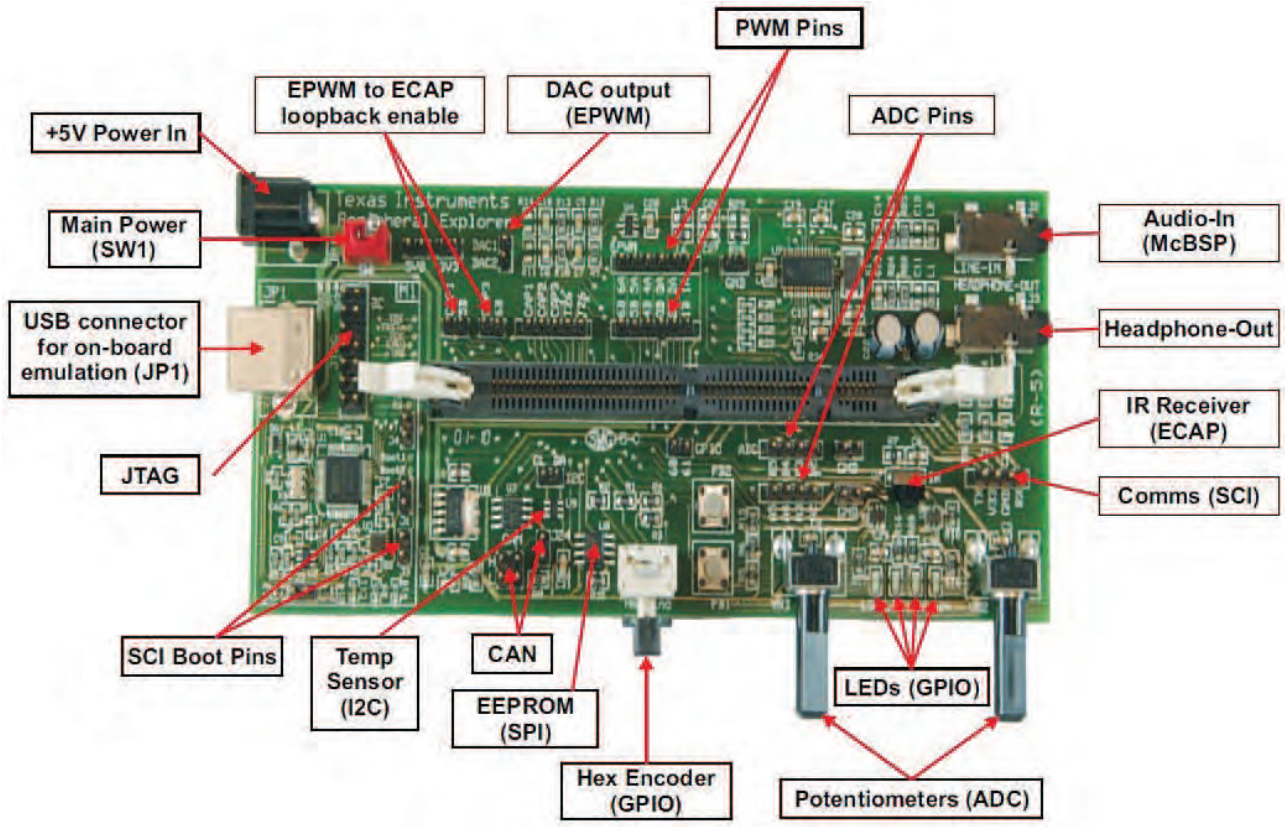


Abbildung 2: verfügbare Peripherie [4]

Die Abbildung 2 stellt das Peripheral Explorer Kit Board dar und beschreibt die verfügbare Peripherie. Für die ersten Tests der Toolchain wurde einer der beiden Potentiometer, die ADC[4] Pins und die PWM[5] Pins verwendet. Die genaue Umsetzung und Verwendung wird im Kapitel 5 beschrieben.

4 ADC: Analog-to-Digital Converter
5 PWM: Puls-Width Modulation

2.2 Praktikumsversuch: Nachlaufregelung

Der Praktikumsversuch Nachlaufregelung besteht aus einer Linearachse mit der eine gewünschte Postion angefahren werden sollte. Der Aufbau der Linearachse ist in der Abbildung 3 zu sehen.

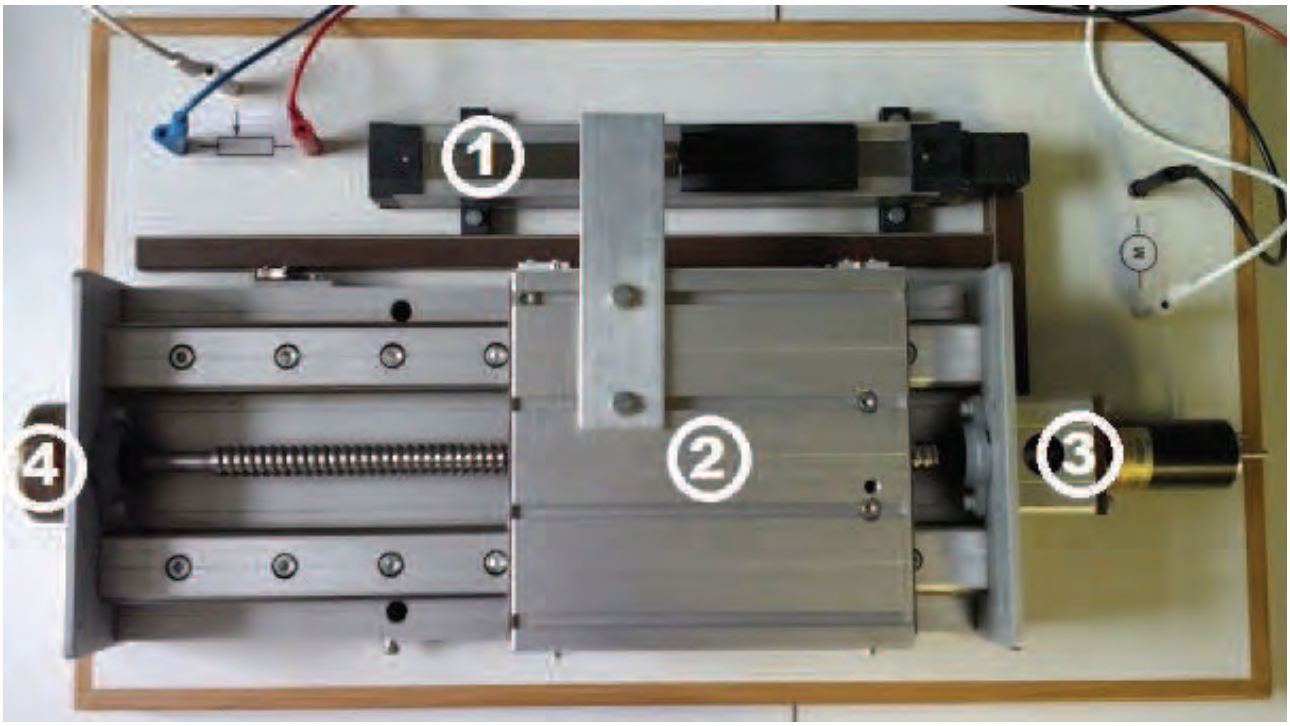


Abbildung 3: Linearachse

Die eingezeichneten Markierung stellen folgendes dar:

1 Schiebepotentiometer (Positionsbestimmung, Streckeneingang)
2 Schlitten
3 Gleichstrommotor (über Leistungsverstärker, Streckenausgang)
4 Schwungmasse

In Abbildung 4 ist die Linearachse in einem Blockschaltbild dargestellt.

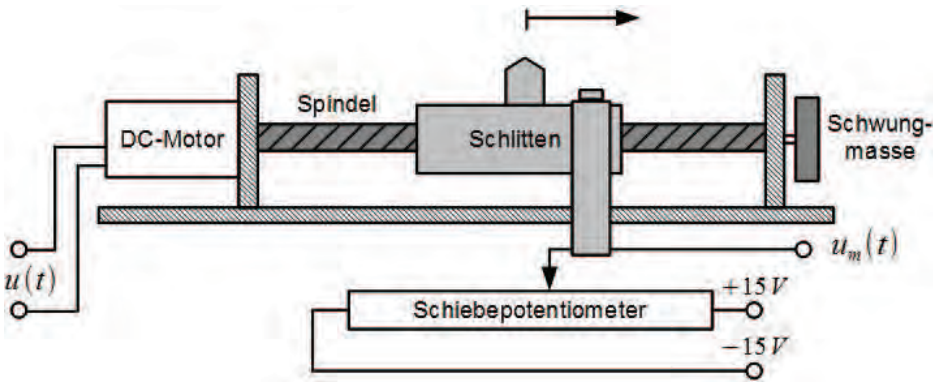


Abbildung 4: Blockschaltbild Linearachse [5]

Funktion der Linearachse:
Beim Anlegen einer Spannung am Gleichstrommotor dreht sich die Spindel und der Schlitten wird je nach Drehrichtung nach links oder rechts bewegt. Der am Schlitten befestigte Schieberegler des Schiebepotentiometers wird dabei mitbewegt. Somit erfährt die am Schiebepotentiometer gemessene Spannung eine proportionale Wertänderung. Durch diesen Vorgang wird eine Positionsbestimmung ermöglicht. Die Linearachse hat die Spannung für den Gleichstrommotor als einen Eingang und liefert mit dem Schiebepotentiometer eine Spannung.

Nachlaufregelung:
Der Regler hat die eingestellte Spannung am Drehpotentiometer des TI C2000 Peripheral Explorer Kit Boards und die gemessene Spannung am Schiebepotentiometer der Linearachse als Eingang. Mit dem Potentiometer des Boards wird die Sollwertvorgabe durchgeführt.

Potentiometer Drehung nach Links	→	Schlitten fährt nach Links
Potentiometer Drehung nach Rechts	→	Schlitten fährt nach Rechts

Es kann jede beliebige Position zwischen den Endwerten angefahren werden.

Als Ausgang muss der auf dem Delfino DSP[6] implementierte Regler einen analogen Wert für den Gleichstrom bereitstellen. Da die verwendete controlCARD keinen analogen Ausgang hat, wird ein ePWM[7] Signal erzeugt. Die controlCARD mit dem Delfino TMS320F28335 kann für die verwendeten PIN nur Spannungen im Bereich von 0-3V verarbeiten. Aufgrund dessen wurde eine Schaltung entworfen die den Wertebereich anpasst.

Vorverschaltung:
Die Schaltung aus der Abbildung 5 erzeugt aus dem Wertebereich (-15V bis +15V) des Schiebepotentiometer, ein für den analogen Eingang am Peripheral Explorer Kit Board gültigen Wertebereich (0 bis 3V).
Der erste am Schiebepotentiometer angeschlossene invertierende Operationsverstärker hat eine Verstärkung von etwa -1/7 was den Wertebereich auf ungefähr -2V bis 2V verkleinert. Der daran anknüpfende invertierende Operationsverstärker ist ein Addierer. Dabei bestimmt U_1 um wie viel der Bereich verschoben werden soll. Es wurde ein Wert von -2V für U_1 eingestellt. Damit ergibt sich ein Wertebereich von 0 bis 4V. Da der Sollwert über das Drehpotentiometer am Board nur im Bereich zwischen 0 und 3,3V eingestellt werden kann, wird vom Schiebepotentiometer auch nur ein Wert in diesem Bereich erwartet.

6 DSP: Digital Signal Processor
7 ePWM: Enhanced Pulse-Width Modulation

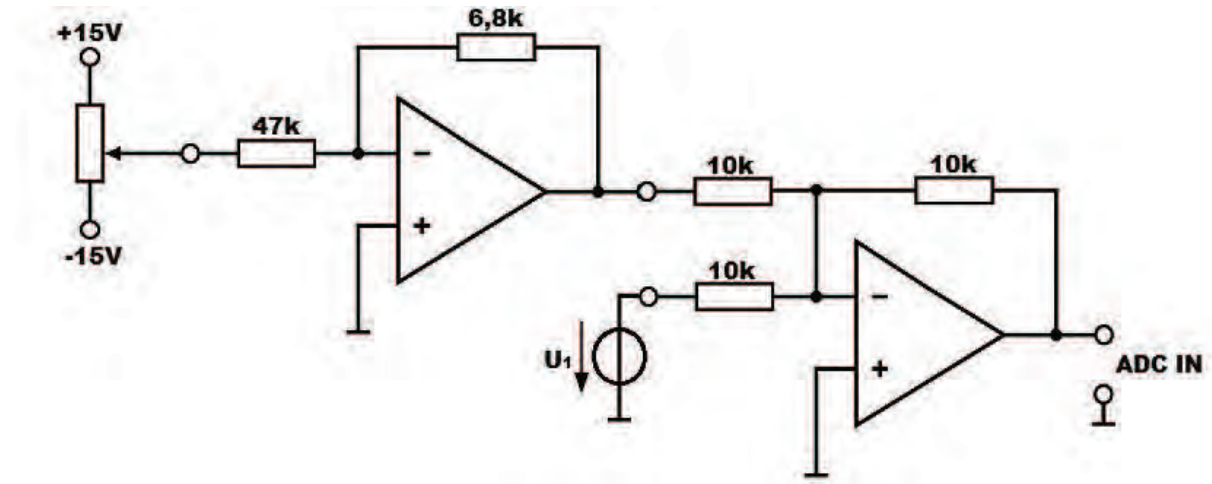


Abbildung 5: Schaltplan Teil 1

Die Schaltung in der Abbildung 6 macht aus dem Wertebereich (0 bis 3,3V) des ausgegebenen ePWM-Signal ein für die Ansteuerung des Motors benötigten Wertebereich (-10V bis 10V).
An den ePWM PIN des Boards ist ein invertierender Operationsverstärker angeschlossen. Mit diesem Addierer wird durch U_2 bestimmt, wie der Eingangswert verschoben wird. Da das ausgegebene ePWM-Signal nur zwischen 0 und 3,3V liegt, wird bei der Wahl von U_2 = -1,65V ein symmetrischer Wertebereich von -1,65V bis +1,65V erzeugt. Durch den folgenden invertierenden Operationsverstärker mit einer Verstärkung von etwa 6, wird der Wertebereich auf -10V bis 10V erweitert. Mit dem vor dem Motor geschalteten Leistungsverstärker ist die Schaltung komplett.

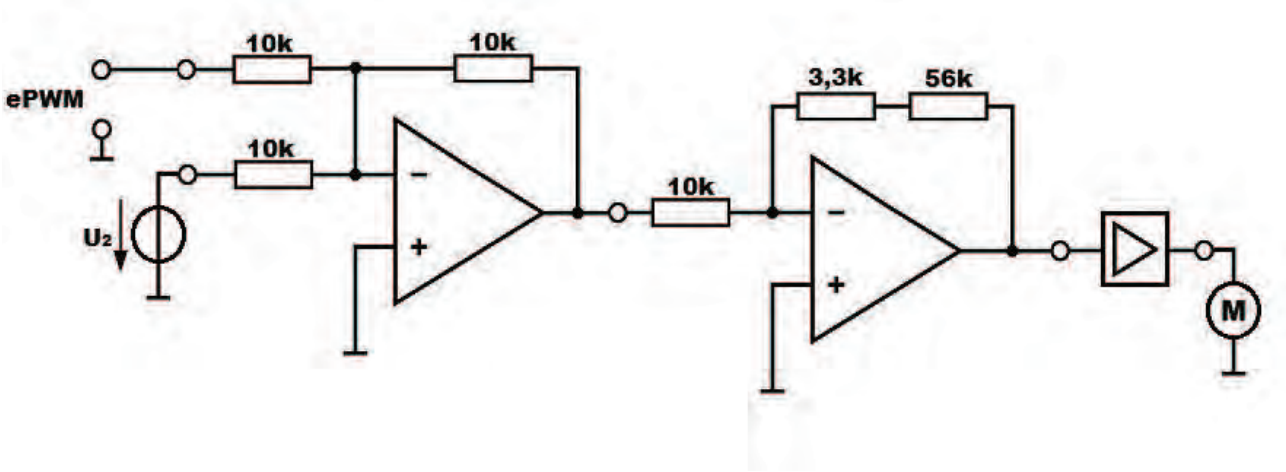


Abbildung 6: Schaltplan Teil 2

3. Verwendete Software

3.1. MATLAB

MATLAB [7] ist ein Programm und eine Hochsprache. Damit können Berechnungen durchgeführt und anschließend visuell dargestellt werden. Die Software ist ein beliebtes Tool in der Industrie sowie an Hochschulen und wird unter anderem für regelungstechnische Aufgaben eingesetzt. Die grafische Oberfläche ist in der Abbildung 7 gezeigt.

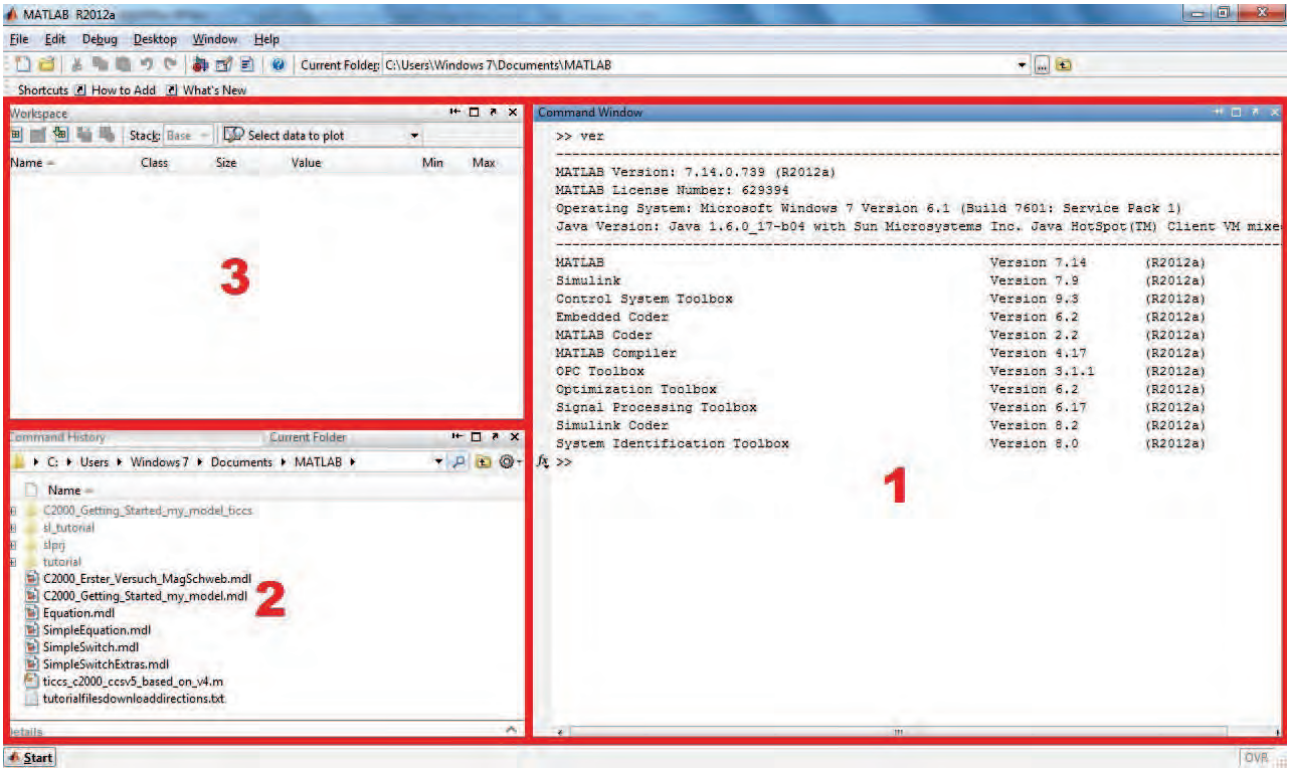


Abbildung 7: MATLAB Oberfläche

Die eingezeichneten Markierungen stellen folgendes dar:

1 Command Window
2 Current Folder und Command History
3 Workspace

Im Command Window werden alle MATLAB Befehle eingegeben und alle Berechnungen, Warnungen sowie Informationen ausgegeben. Der Current Folder zeigt das aktuelle Verzeichnis an und hier können weitere MATLAB Modelle geladen werden. Die Command History listet alle im Workspace eingegebenen Befehle auf. Im Workspace werden alle in der aktuellen MATLAB-Sitzung vorhandenen Variablen mit den entsprechenden Werten hinterlegt.

Nach eingeben des MATLAB Befehls `ver` im Command Window, wird die aktuelle MATLAB Version mit allen installierten Toolboxen angezeigt. Für die Durchführung dieser Anleitung wird Simulink sowie der Embedded Coder benötigt.

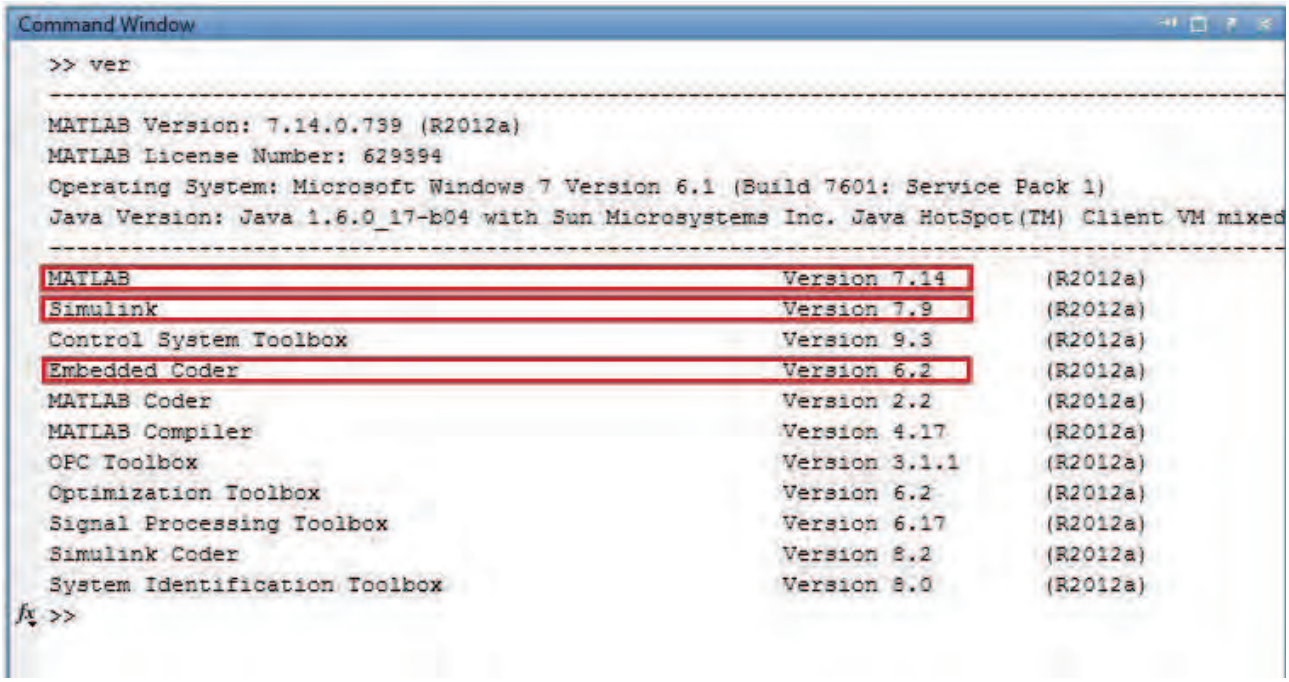


Abbildung 8: MATLAB und Toolboxen Version

Durch das Eingeben des Befehls `ver` soll überprüft werden ob die für die Durchführung der Anleitung benötigten Toolboxen (Abbildung 8) installiert sind.

MATLAB	Version 7.14
Simulink	Version 7.9
Embedded Coder	Version 6.2

3.2. Simulink

Mit Simulink [8] kann das modellbasierte Design und verschiedene Simulationen durchgeführt werden. Da hier mit Modellen (grafische Programmiersprache) gearbeitet wird und nicht mit Befehlen, erlaubt Simulink einen einfachen Überblick über den Sachverhalt. Die Simulink Library kann aus MATLAB wie in der Abbildung 12 zu sehen, aufgerufen werden.

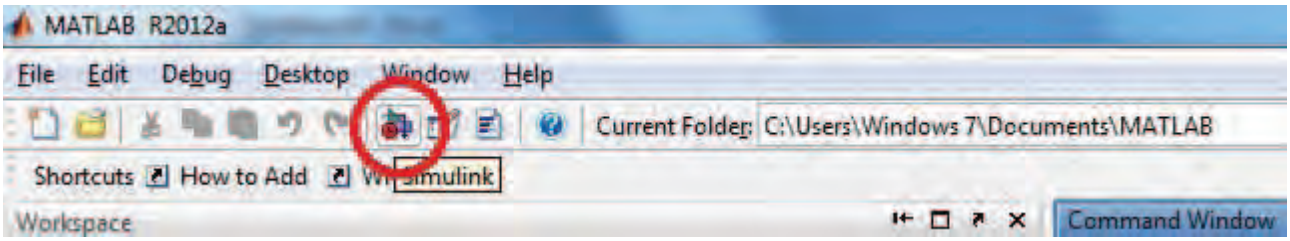


Abbildung 9: Simulink Aufruf in MATLAB

Die daraufhin erscheinende Simulink Library (Abbildung 10) erlaubt das Erstellen eines neuen Modells über *File* > *New* > *Model*. In dem sich geöffneten noch leeren Modellfenster werden die benötigten Blöcke aus der Simulink Library eingefügt.

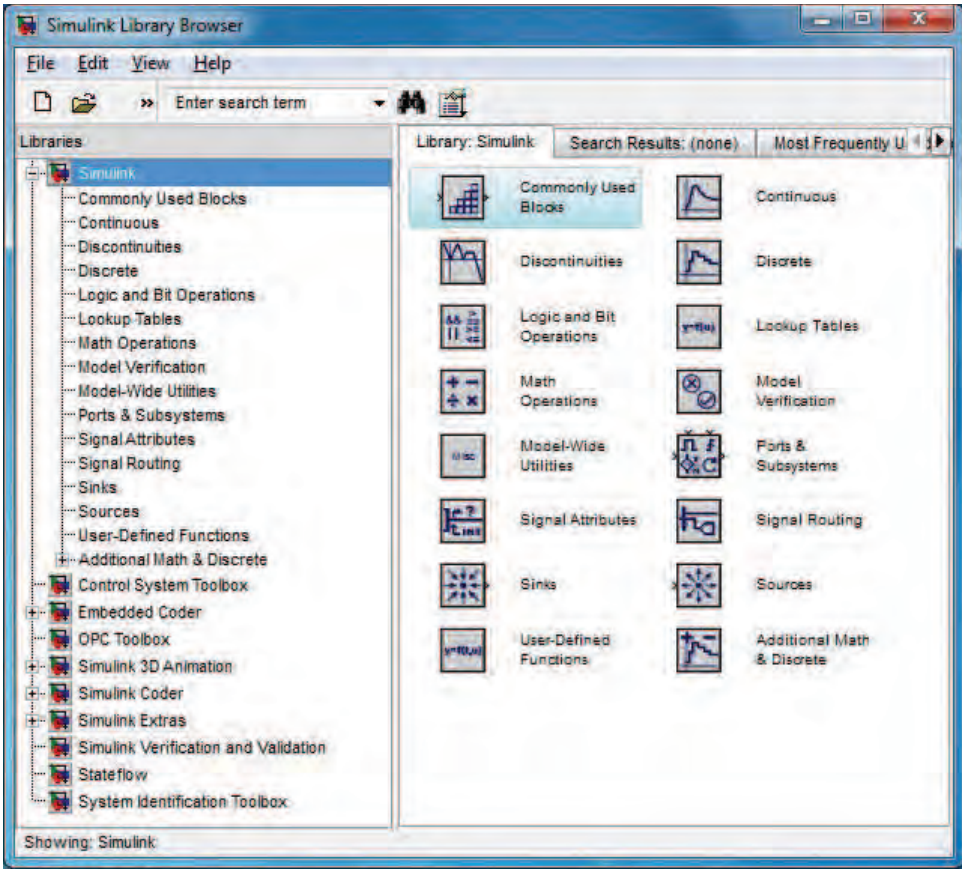


Abbildung 10: Simulink Library Browser

3.3. Embedded Coder

Eine während dieser Anleitung verwendete Toolbox in MATLAB ist der Embedded Coder [9]. Damit kann Code für verschiedene DSPs erzeugt werden. Im Simulink Library Browser befindet sich eine Kategorie „Embedded Coder“ in der für die jeweilige Zielhardware bestimmte Blöcke abgelegt sind.

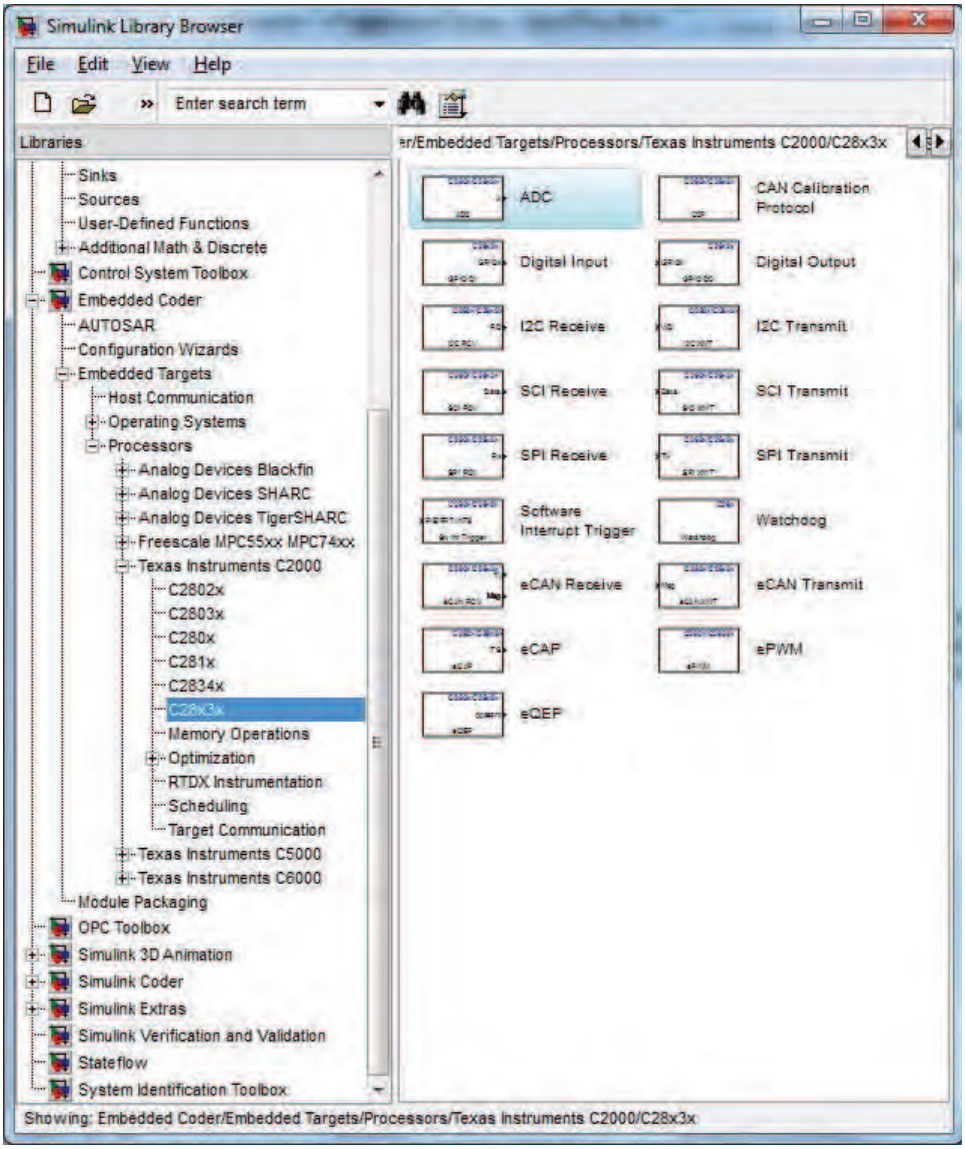


Abbildung 11: Embedded Coder in der Simulink Library

Für das verwendete TI C2000 Peripheral Explorer Kit mit der controlCARD und dem TMS320F28335 DSP finden sich die Blöcke unter *Embedded Coder > Embedded Targets > Processors > Texas Instruments C2000 > C28x3x* (Abbildung 11).
Der Target Preference Baustein, der bei Simulink Modellen für Embedded Systeme vorhanden sein muss, findet sich unter *Embedded Coder > Embedded Targets > Target Preferences*.

3.4. Code Composer Studio v5

Die von Texas Instruments im Rahmen vieler Anwendungen kostenlos zur Verfügung gestellte Entwicklungsumgebung CCS [15] basiert auf einer ehemals für Java entwickelten quelloffenen integrierten Entwicklungsumgebung. Durch die vielen Erweiterungen ist es aber mittlerweile möglich für nahezu alle Programmiersprachen zu entwickeln. Mit CCS kann nicht nur der C2000 programmiert werden sondern alle von TI angebotenen Mikrocontroller. Die grafische Benutzeroberfläche wird in der Abbildung 12 dargestellt.

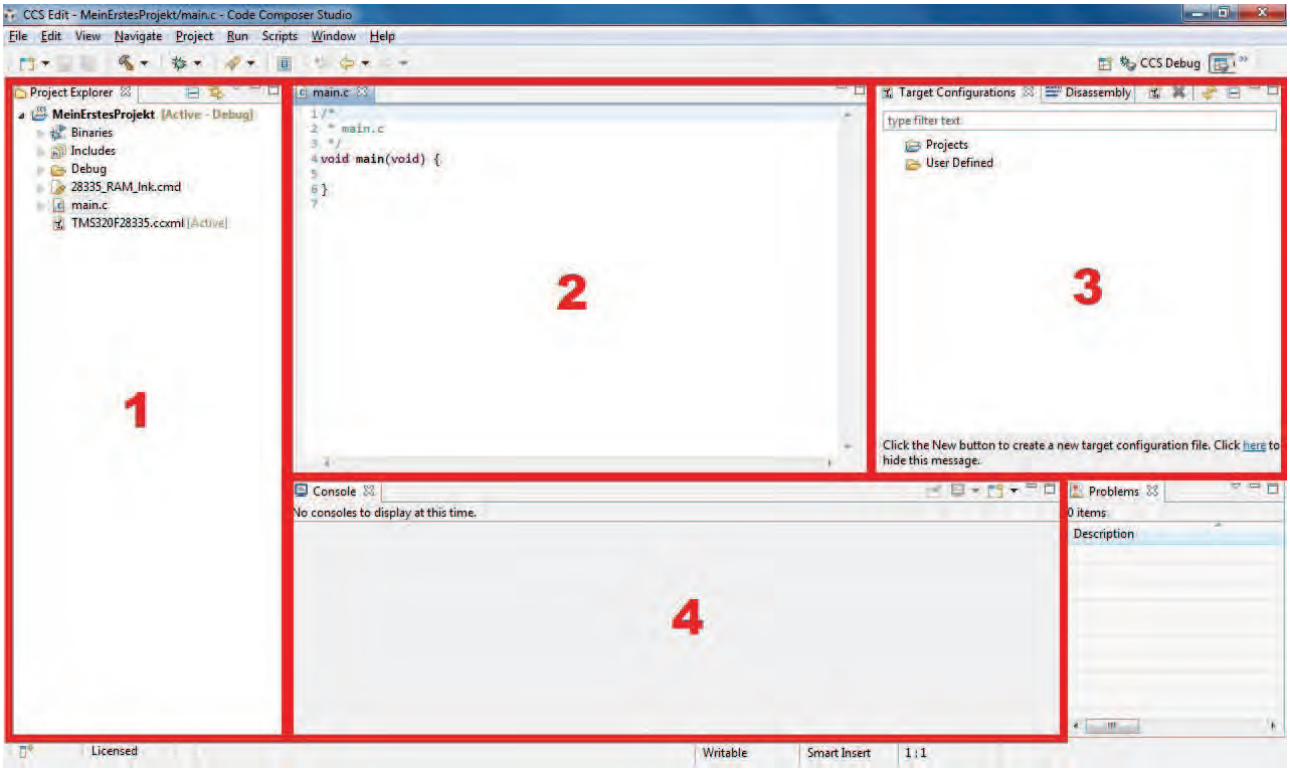


Abbildung 12: CCS Oberfläche

Die eingezeichneten Markierungen stellen folgendes dar:

1 Project Explorer (Überblick der angelegten Projekte)
2 Quellcode (Hier kann der Quellcode erstellt und editiert werden)
3 Target Configuration (Einstellungen der Zielhardware)
4 Console (Ausgabe zu einem Prozess ist hier einsehbar)

Die Fenster können innerhalb der Entwicklungsumgebung frei konfiguriert werden, d.h. verschoben, geschlossen und über die Menüleiste unter *View* wieder geöffnet werden.

4. Einstellungen

4.1. MATLAB Konfiguration

Der folgende Abschnitt beschreibt die Konfiguration in MATLAB.

Im Command Window von MATLAB bitte folgenden Befehl eintippen: `xmakefilesetup` (Abbildung 13)

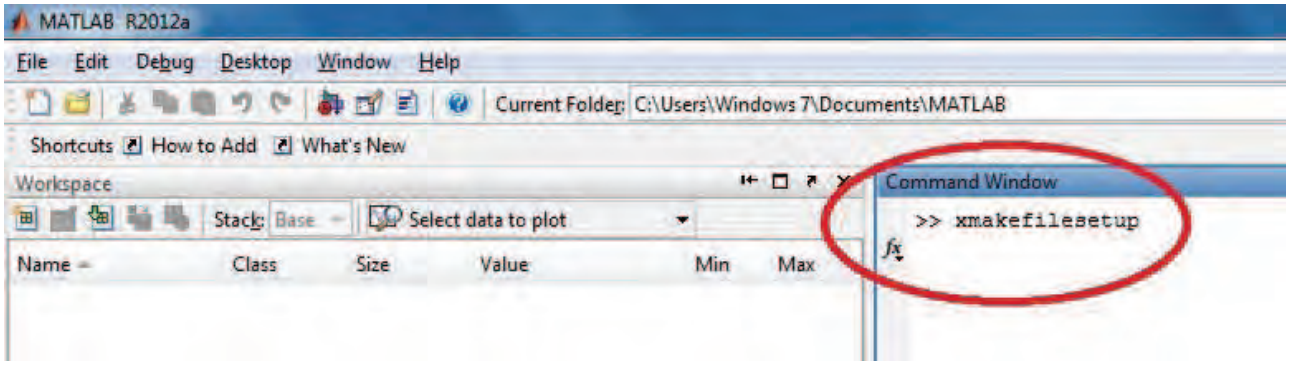


Abbildung 13: Command Window Befehleingabe

Daraufhin erscheint ein neues Fenster *XMakefile User Configuration* (Abbildung 14) in dem alle folgenden Einstellungen vorgenommen werden.

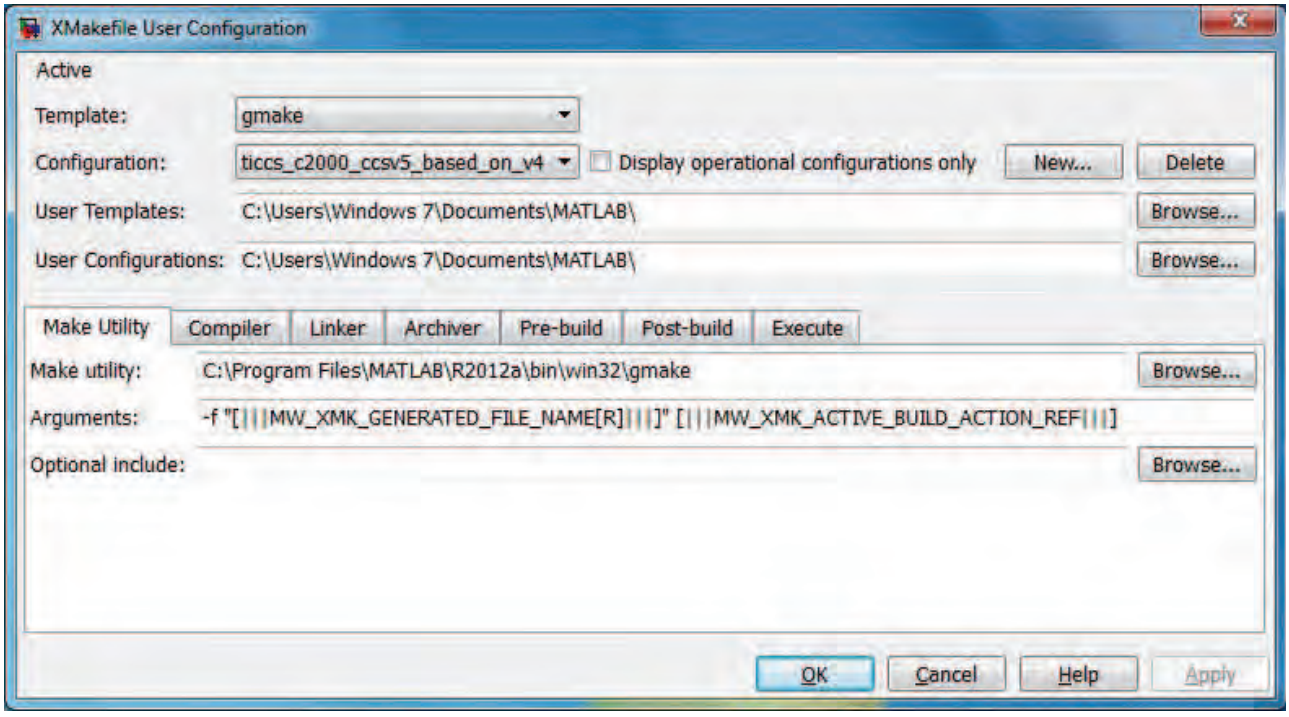


Abbildung 14: XMakefile User Configuration

Unter *Template* wird *gmake* ausgewählt und das Häckchen bei *Display operational configurations only* entfernen. Desweiteren sollte unter *Configuration*, *ticcs_c2000_ccsv4*(!) ausgewählt werden. Dann einen Klick auf *Apply*, ohne das Fenster zu schließen.

Falls der Aufruf der XMakefile User Configuration zum ersten Mal erfolgt, so wird man nach dem Installationspfad für Code Composer Studio v4(!) gefragt. Es wird nach CCSv4 gefragt, da die neueste Version CCSv5 erst vor kurzem veröffentlicht wurde. Hier bitte den Installationspfad der verwendeten Version von Code Composer Studio angeben. Das sieht dann zum Beispiel so aus: *C:\ti\ccsv5* bei der Verwendung von CCSv5.

Wenn der Aufruf schon bei der Konfiguration von CCSv4 erfolgte, dann sollen die Angaben für CCSv5 unter *Tool Directories* editiert werden.

Das sieht dann folgendermaßen aus (Abbildung 15):

CCS Installation:	C:\ti\ccsv5\
Code Generation Tolls:	C:\ti\ccsv5\tools\compiler\c2000_6.1.0\
DSP/BIOS Installation:	C:\ti\bios_6_33_05_46\

Jetzt wieder ein Klick auf *Apply* ohne das Fenster zu schließen.

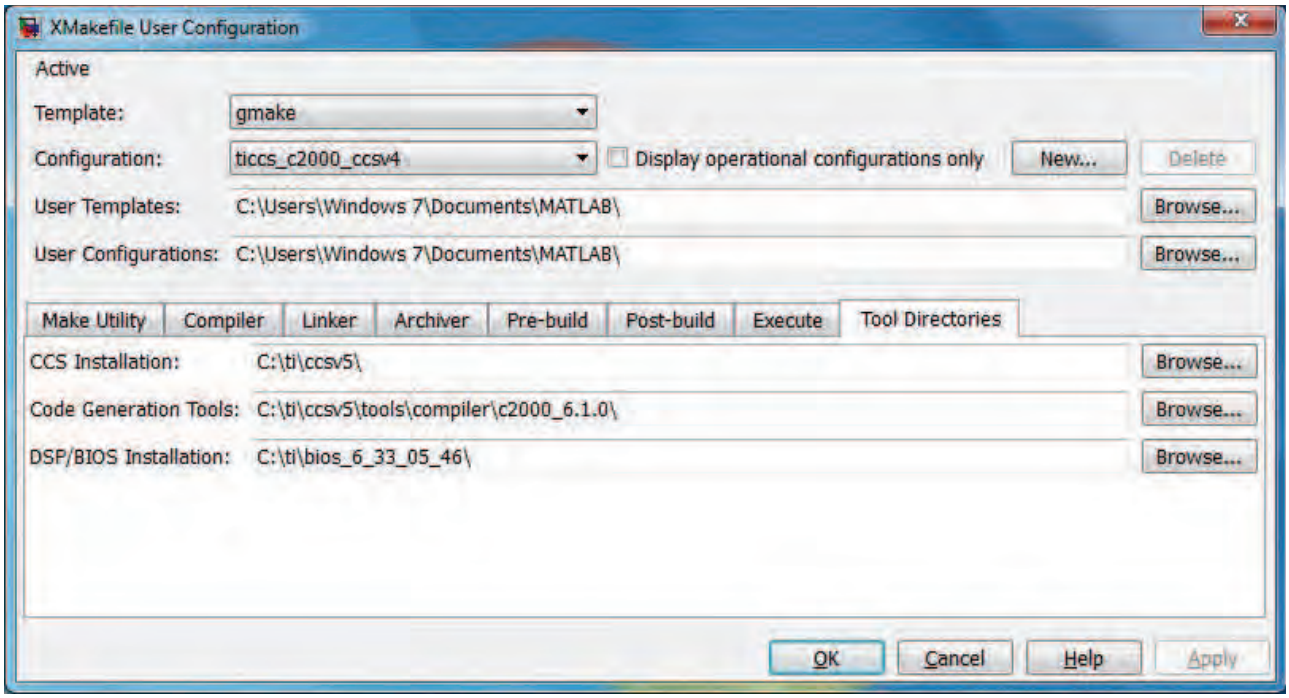


Abbildung 15: XMakefile User Configuration: Tool Directories

Der nächste Schritt wäre ein Klick auf *New* ganz rechts im XMakefile User Configuration Fenster. Es wird vorgeschlagen, die neue Konfiguration z.B. *ticcs_c2000_ccsv4_clone* zu nennen. Ein Vorschlag wäre *ticcs_c2000_ccsv5_based_on v4* (Abbildung 16) und anschließend auf *OK* klicken.

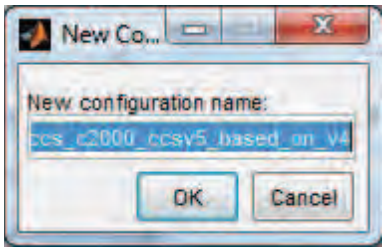


Abbildung 16: New Configuration Name

Alle Reiter in der XMakefile User Configuration sind jetzt editierbar und sollten insbesondere bei *Compiler*, *Linker* und *Archiver* auf das aktuelle Installationsverzeichnis von CCSv5 aktualisiert werden.

Das schaut dann z.B. wie in den Abbildungen 17 – 19 aus:

Reiter Compiler:
Compiler: C:\ti\ccsv5\tools\compiler\c2000_6.1.0\bin\cl2000
Arguments: -I"C:\ti\ccsv5\tools\compiler\c2000_6.1.0\include" -fr"[|||
MW_XMK_DERIVED_PATH_REF|||]"

Reiter Linker:
Linker: C:\ti\ccsv5\tools\compiler\c2000_6.1.0\bin\cl2000

Reiter Archiver:
Archiver: C:\ti\ccsv5\tools\compiler\c2000_6.1.0\bin\ar2000

A, Einstellungen unter *Compiler*

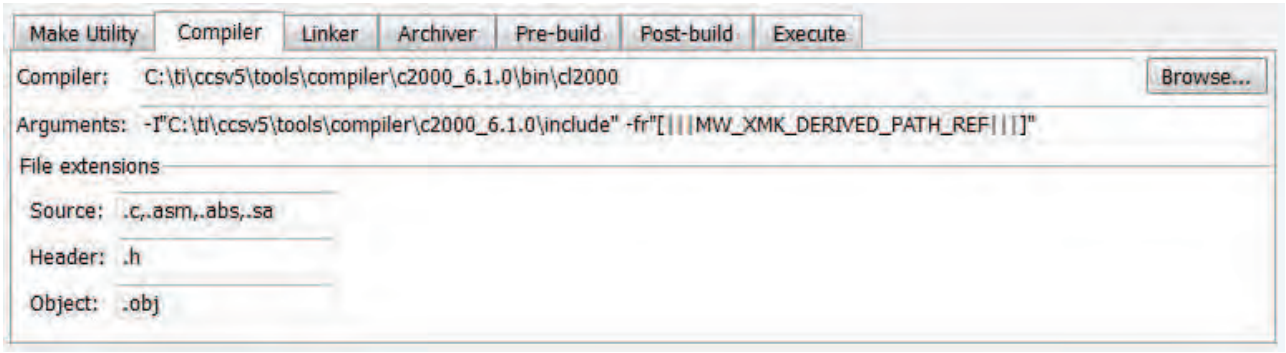


Abbildung 17: Compiler

B, Einstellungen unter *Linker*

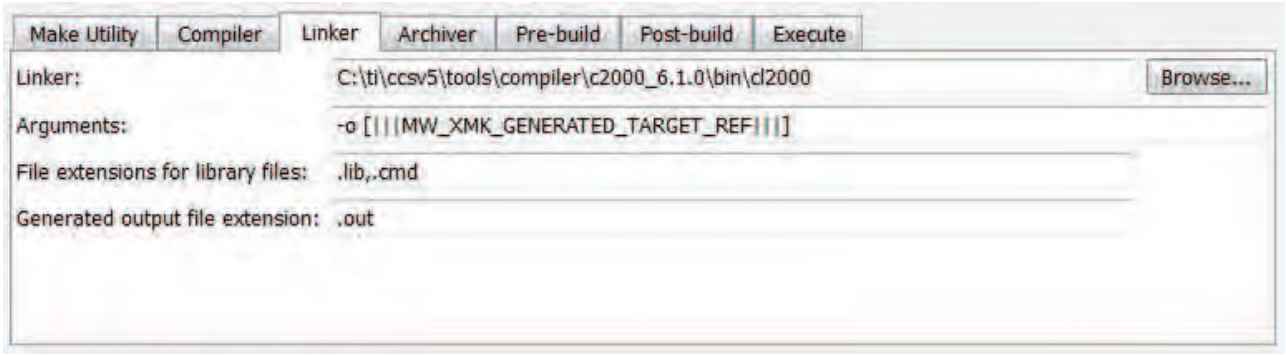


Abbildung 18: Linker

C, Einstellungen unter *Archiver*

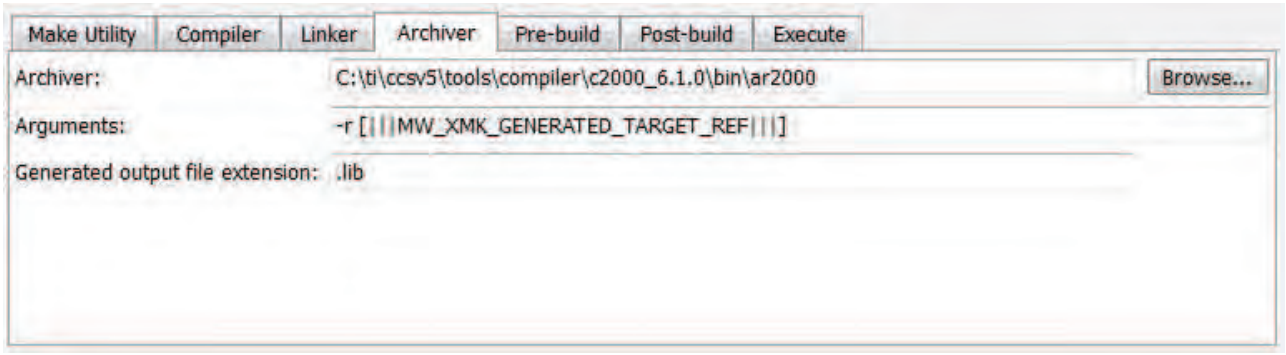


Abbildung 19: Archiver

Das waren jetzt alle Einstellungen für die Erstellung von *makefiles* zur Zusammenarbeit von MATLAB/Simulink mit der Entwicklungsumgebung Code Composer Studio v5. Ein Klick auf *OK* und das Fenster kann geschlossen werden. Nun kann mit MATLAB/Simulink gearbeitet werden.

4.2. CCSv5 Konfiguration

Der folgende Abschnitt beschreibt die Konfiguration in der CCSv5 Entwicklungsumgebung. Falls CCSv5 zum ersten Mal gestartet wird, folgt eine Anfrage nach dem Anlegen eines Workpace. Eine mögliche Auswahl ist *C:\Users\Benutzername\workspace_v5_2* (Abbildung 20).

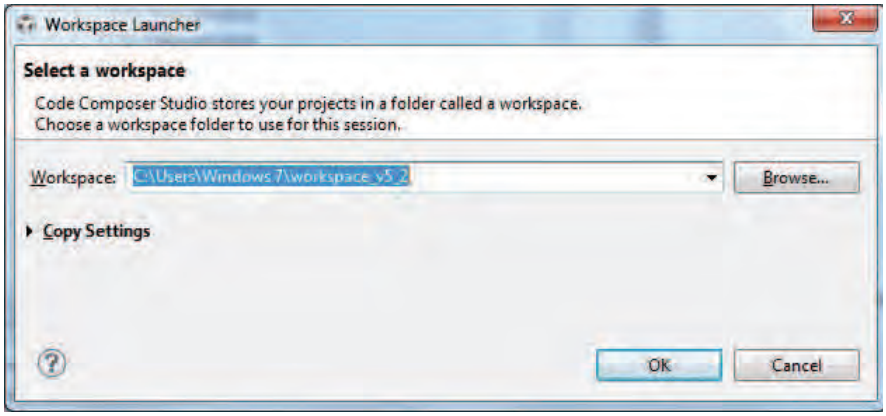


Abbildung 20: Workspace

Nachdem CCSv5 gestartet und der Workspace angelegt wurde, wird das erste Projekt angelegt. Der Aufruf dazu geht über *File > New > CCS Project* (Abbildung 21).

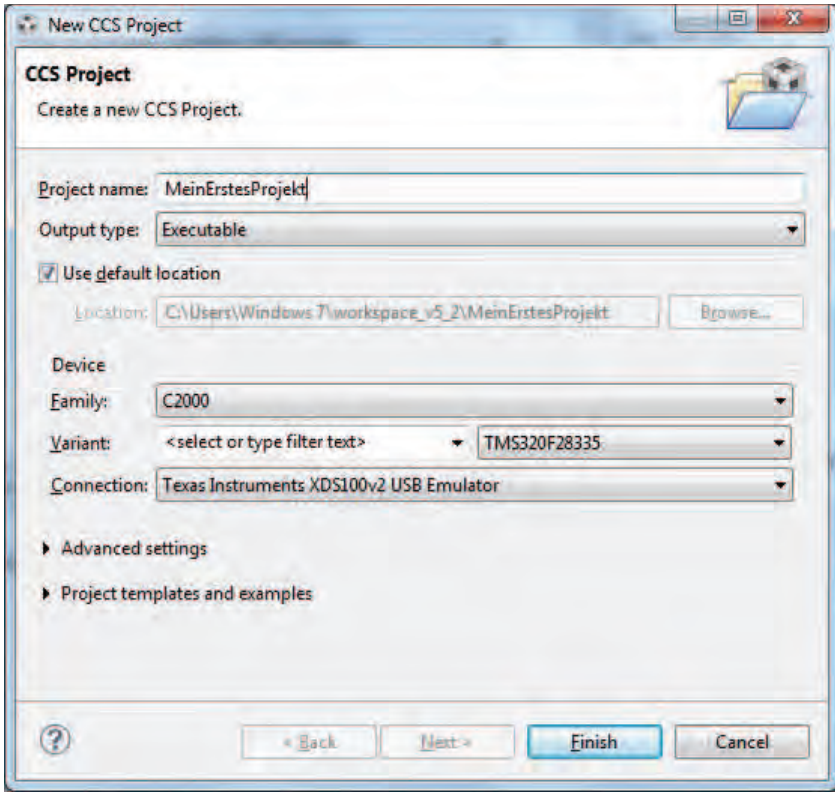


Abbildung 21: New CCS Project

Für *Project name* kann z.B. *MeinErstesProjekt* gewählt werden.
Das *Output file* sollte ein *Executable* sein und das Häckchen bei *Use default location* gesetzt werden.
In der Kategorie *Device* bitte unter *Family* die *C2000 Family* auswählen. Die verwendete *Variante* ist der *TMS320F28335* und die *Connection* erfolgt über *Texas Instruments XDS100v2 USB Emulator*. Zur *Target Configuration* gelangt man später über *View* > *Target Configuration* und dann ein Doppelklick auf die *.ccxml-Datei im jeweiligen Projekt. Hier sollte später nochmals überprüft werden, ob die richtige Zielhardware eingestellt ist, bevor der Code übertragen wird.
Die Auswahl unter *Advanced Settings* und *Project templates and examples* kann einfach übernommen werden.
Die Einstellungen werden durch einen Klick auf *Finish* übernommen und das Projekt erstellt.
In Abbildung 22 ist das angelegte Projekt zu sehen.

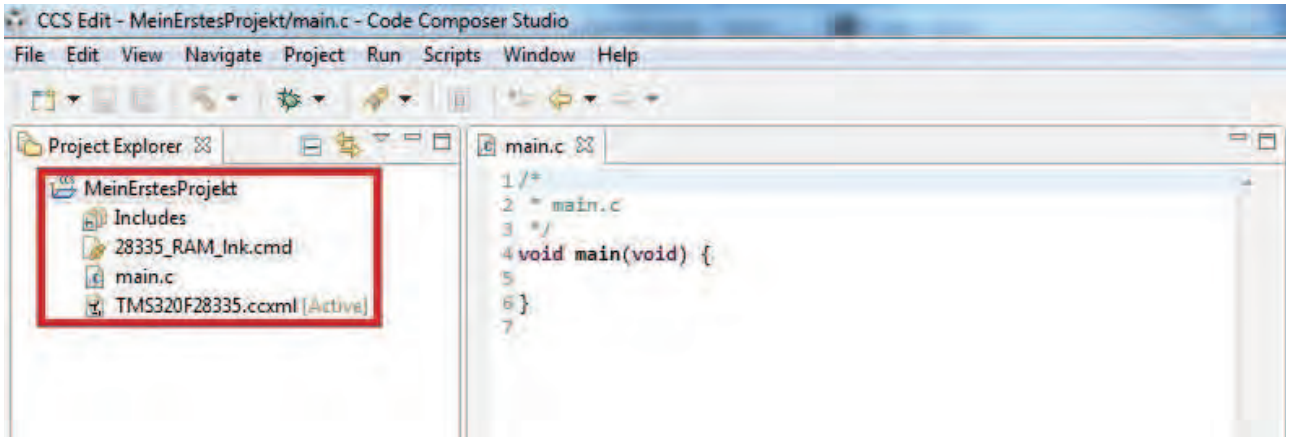


Abbildung 22: Angelegtes Projekt mit einer C-Code Vorlage von main.c

Als nächstes sollte die Größe des *system stack* für alle C Programme definiert werden [16]. Diese Einstellungen werden unter *Properties* (Klick mit der rechten Maustaste auf MeinErstesProjekt) vorgenommen. Unter Build > C2000 Linker > Basic Options >*Set C system stack size*: *0x400* eingeben (Abbildung 23).

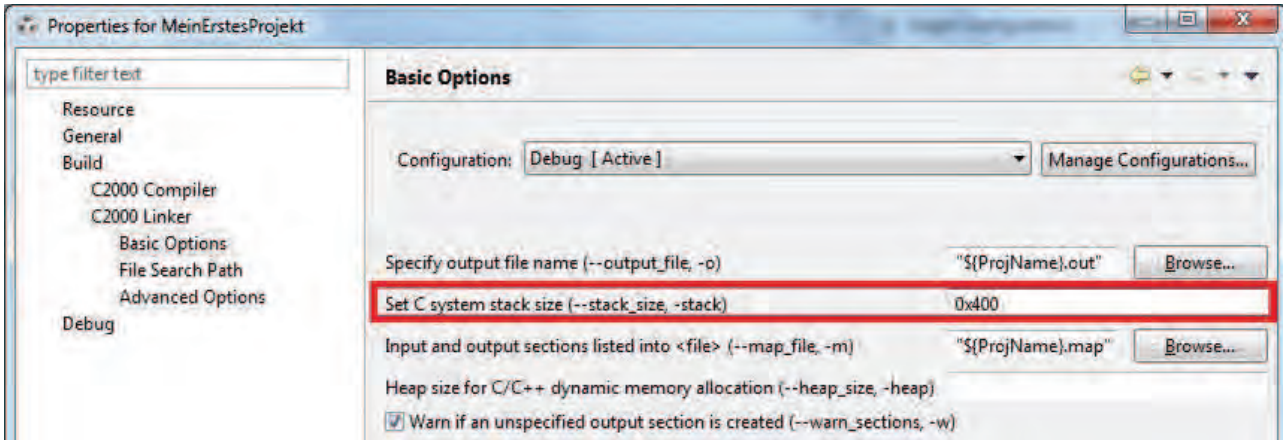


Abbildung 23: System Stack

Ein kleines Beispielprogramm wird nun in CCSv5 erstellt und anschließend auf dem Delfino DSP ausgeführt um die vorher vorgenommenen Einstellungen zu überprüfen, d.h. die Funktion der Zielhardware sicher zu stellen.

Das vorher angelegten Projekt findet sich im *Project Explorer*. Hier kann eingesehen werden, welche Dateien automatisch von CCS angelegt sind. Im Projekt wurde eine C-Code Datei (main.c) angelegt, die durch einen Doppelklick aufgerufen wird. Der C-Code soll, wie in der Abbildung 24 zu sehen übernommen werden. Damit ist ein erstes Programm erstellt.

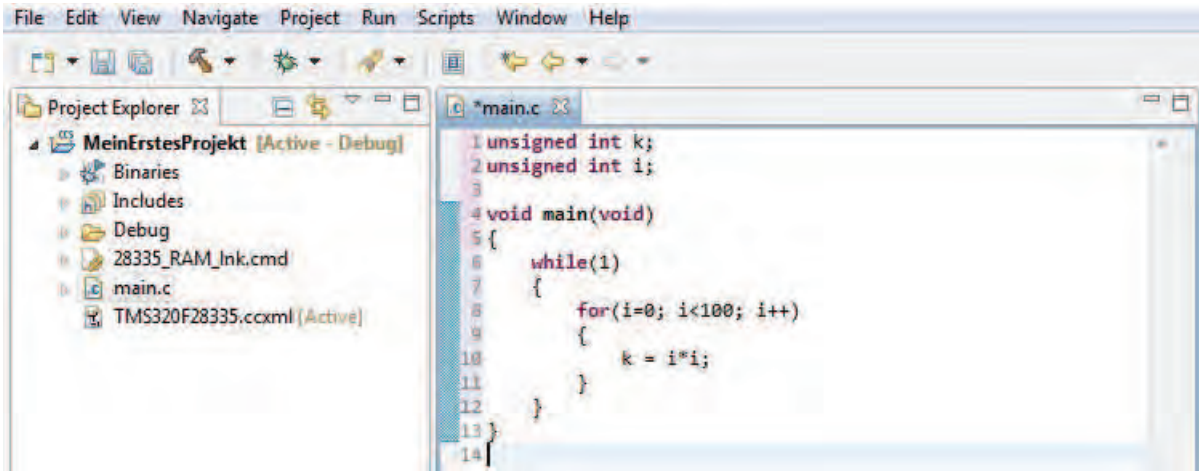


Abbildung 24: C-Code

Es ist ein sehr einfaches Programm mit zwei Variablen `i` und `k`. Das Programm besteht aus einer Endlos-Schleife (`while(1)`) in der ein for-Loop ausgeführt wird. Im for-Loop wird die Variable `i` von 0 bis 99 hochgezählt. Die Variable `k` ist dabei `i * i`.

Mit *Project* > *Build Project* wird der Code kompiliert und eine Ausgabedatei (*.out) erstellt. In der *Console* soll bei einem erfolgreichen Kompilieren ein *Build Finished* stehen.

Mit einem Klick auf *Debug* (Abbildung 25) werden gleich drei Schritte ausgeführt:

- Rebuild (Output-Datei erstellt)
- Connect Target (Verbindung zur Zielhardware hergestellt)
- Load Program (Programm geladen)

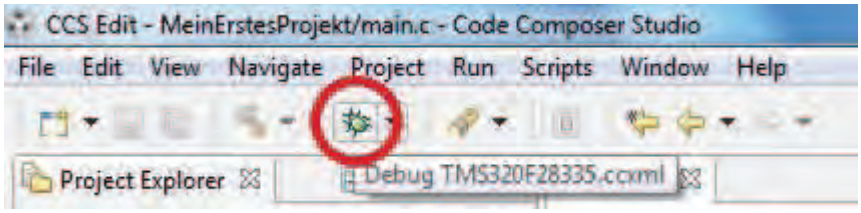


Abbildung 25: Debug

Durch den Klick auf *Debug* gelangt man in die *CCS Debug Perpective*. Ein Indikator dafür, dass das Programm in diesem Fall richtig übersetzt und geladen wurde ist der blaue Pfeil, der auf die Zeile 8 (for-Loop) im Programm zeigt (Abbildung 26). Da erfolgt die erste Wertänderung. Mit *View > Expressions* wird ein Fenster *Expressions* aufgerufen in dem die Variablen i und k über *Add nex expression* angezeigt werden. Hier ist unter anderem der aktuellen Wert der Variablen aufgeführt.

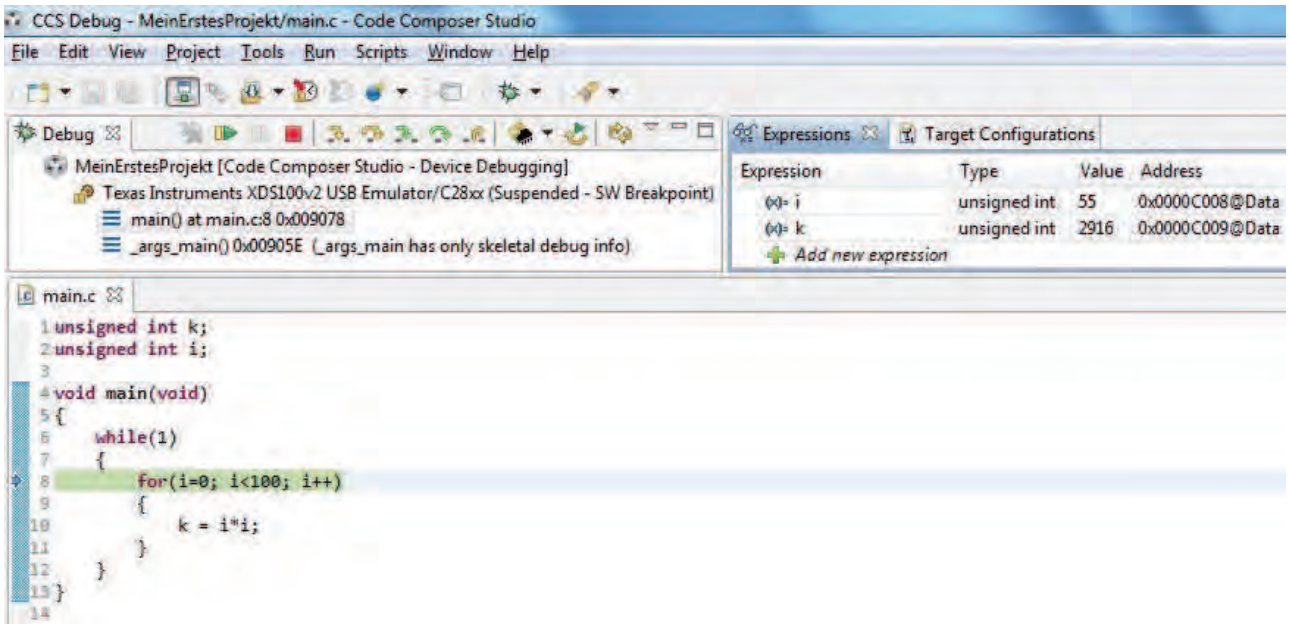


Abbildung 26: CCS Debug

Um die Wertänderung der Variablen nach jedem for-Loop Durchlauf beobachten zu können, werden Breakpoints gesetzt. So wird durch einen Doppelklick links neben der Zeile 10 (blauer Bereich) ein Breakpoint gesetzt. Durch Anklicken von *Resume* (alternativ F8-Taste) findet ein Durchlauf des Programms bis zum Breakpoint statt. Die damit einhergehende Wertänderung der Variablen ist unter *Expressions* (Abbildung 27) sichtbar.

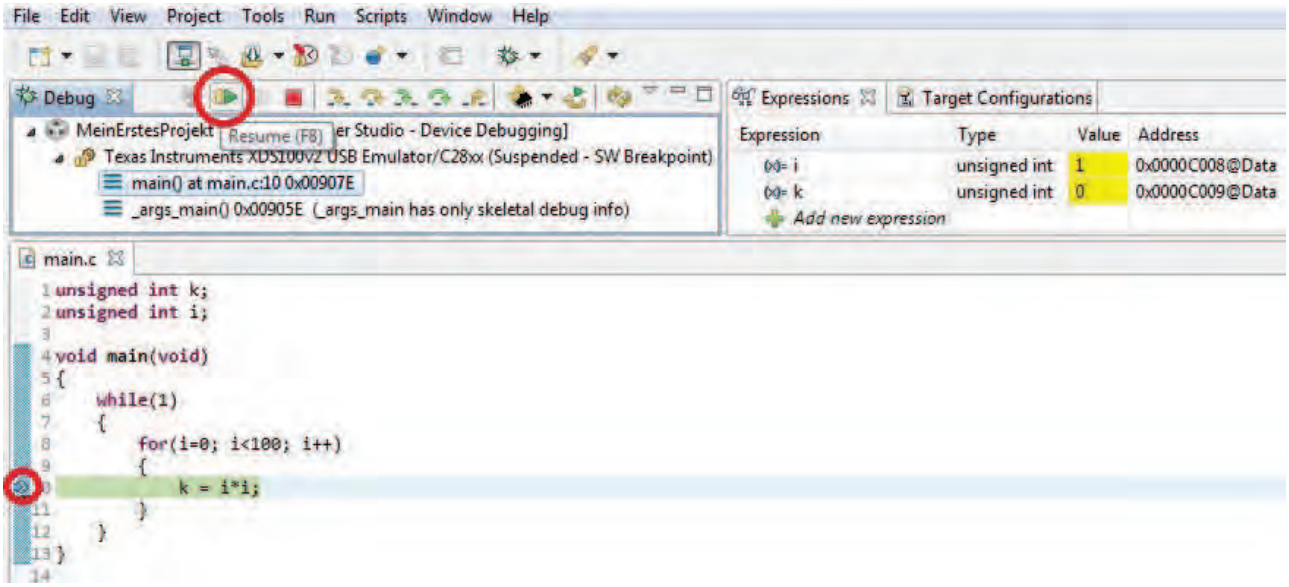


Abbildung 27: Breakpoint

5. Demonstration der Toolchain

Für eine Beurteilung des modellbasierten Softwareentwurfs soll demonstriert werden, dass die vorgenommene Konfiguration in MATLAB und der Einsatz der damit erzeugten Output-Datei in CCSv5 auch zu einer Implementierung von Regler auf DSPs möglich ist.
Dazu wurde ein Versuch aus dem Regelungstechnik Labor der Hochschule Rosenheim im Studiengang Produktionstechnik, wie in Kapitel 2.2 beschrieben verwendet.

Die Demonstration in diesem Kapitel beginnt mit der Erstellung eines Simulink-Modells für den Praktikumsversuch Nachlaufregelung, gefolgt von der Code Erstellung aus dem Simulink-Modell, einem anschließenden Import der Output-Datei in CCSv5 sowie einer Übertragung auf den Delfino DSP auf der controlCARD.

Nach dem Start von MATLAB und dem Aufruf der Simulink Library (Abbildung 9) wird ein neues Modellfenster aufgerufen (Abbildung 28).

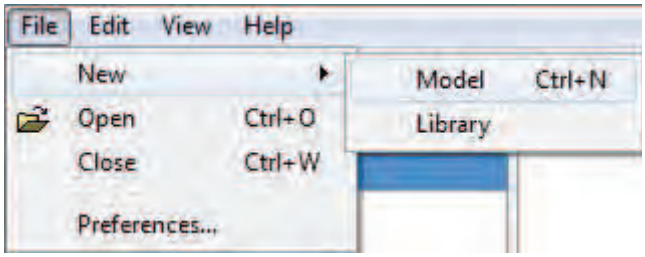


Abbildung 28: Neues Modell

Darin ist als erstes der *Target Preference* Baustein (siehe 3.3) eingefügt worden. Dieser Baustein ist für jede Code Erstellung aus einem Simulink-Modell für eingebettete Systeme notwendig. Nach dem Drag&Drop aus der Simulink Library in das Modellfenster wird automatisch nach den *Konfigurations-Parameter* (Abbildung 29) gefragt.

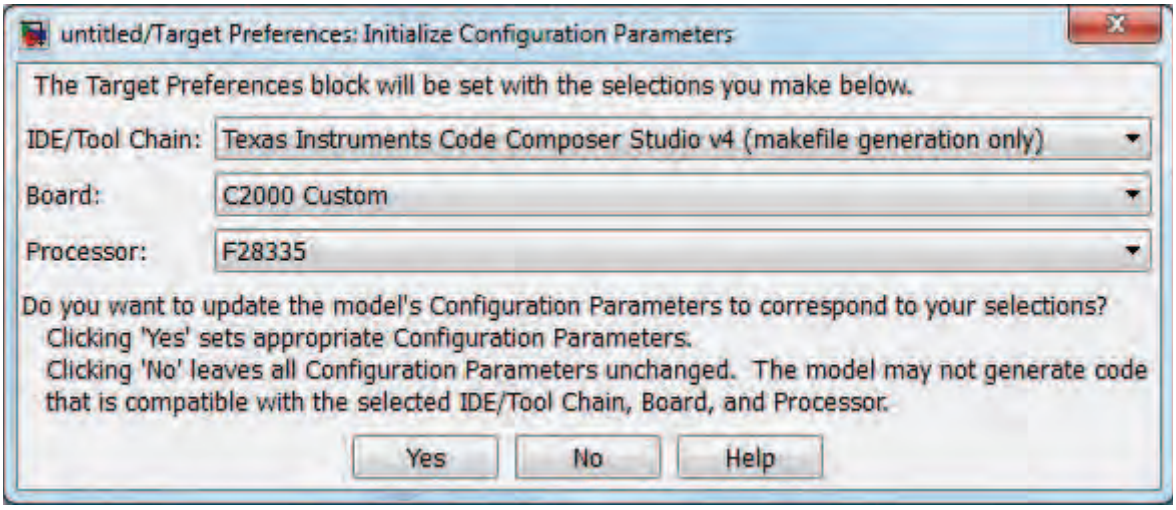


Abbildung 29: Konfigurations-Parameter

Der Abbildung 29 entsprechend sind folgende Einstellungen vorgenommen:

- IDE/Tool Chain: Texas Instruments Code Composer Studio v4 (makefile generation only)
- Board: C2000 Custom
- Processor: F28335

Für den Praktikumsversuch Nachregelung wird ein P-Regler verwendet. Die Verstärkung wurde mit dem Faktor 10 festgelegt. Der Regler hat einen mit dem Drehpotentiometer festlegbaren SOLL-Wert als Eingang und die Rückführung der Regelstrecke, also die IST-Wert Erfassung über das Schiebepotentiometer. Als Ausgang wird ein ePWM-Signal für den Motor bereitgestellt.
Die verschiedenen Bausteine des erstellten Simulink-Modells (Abbildung 30) werden anschließend im Einzelnen aufgeführt und deren Konfiguration detailliert beschrieben. Die Device-spezifischen Bausteine (ADC und ePWM) befinden sich wie in Kapitel 3.3 erklärt in der Simulink Library unter *Embedded Coder > Embedded Targets > Processors > Texas Instruments C2000 > C28x3x* (Abbildung 11). Alle anderen sind aus *Simulink > Commonly Used Blocks* (Abbildung 10).

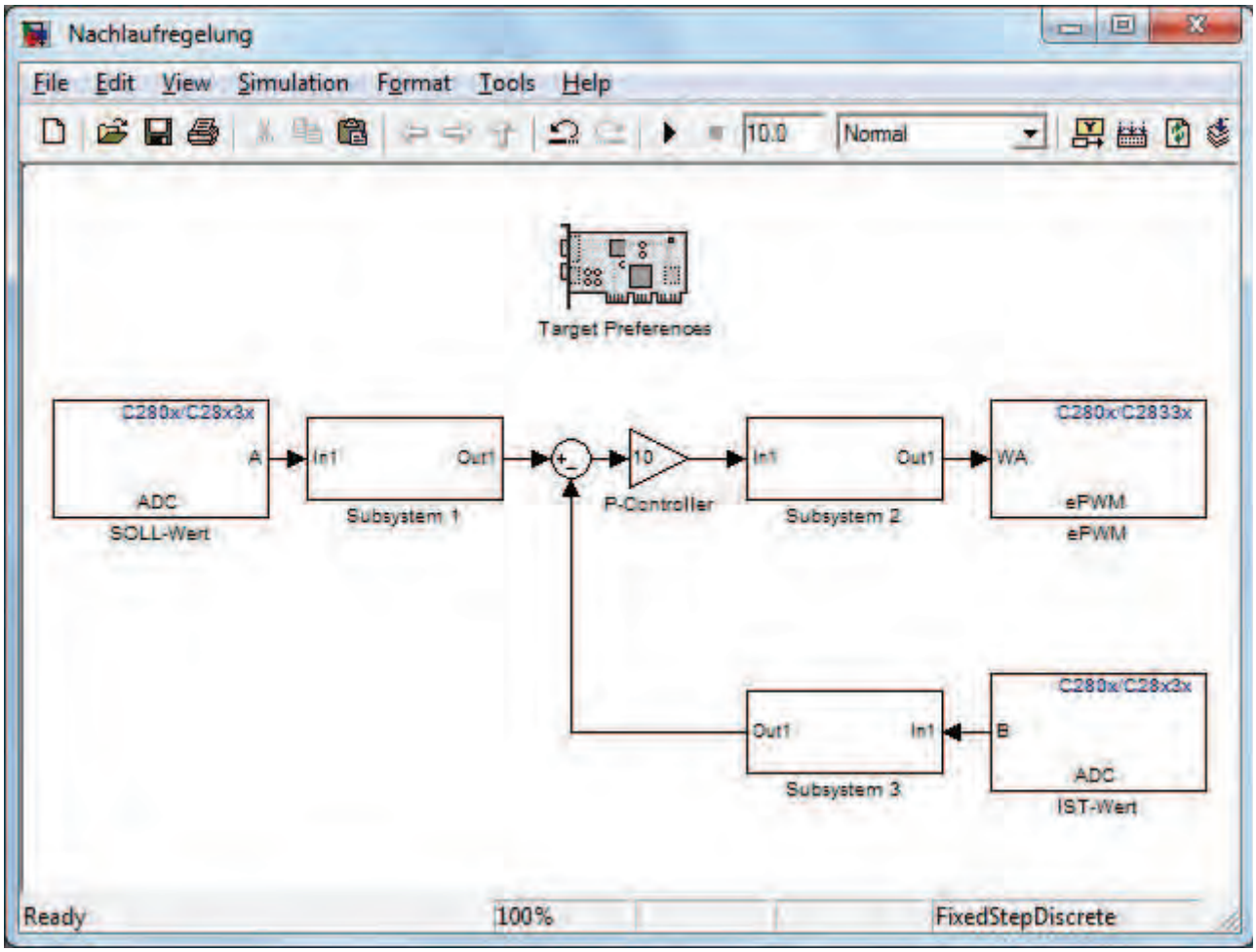


Abbildung 30: Simulink-Modell

Der SOLL-Wert wird mit dem Drehpotentiometer bestimmt, welches beim Peripheral Explorer Kit am analogen Eingang A1 liegt. Der gelieferte Wert ist vom Typ unit16. Die Einstellungen für den Baustein ADC sind in Abbildung 31 und 32 zu sehen getroffen worden.

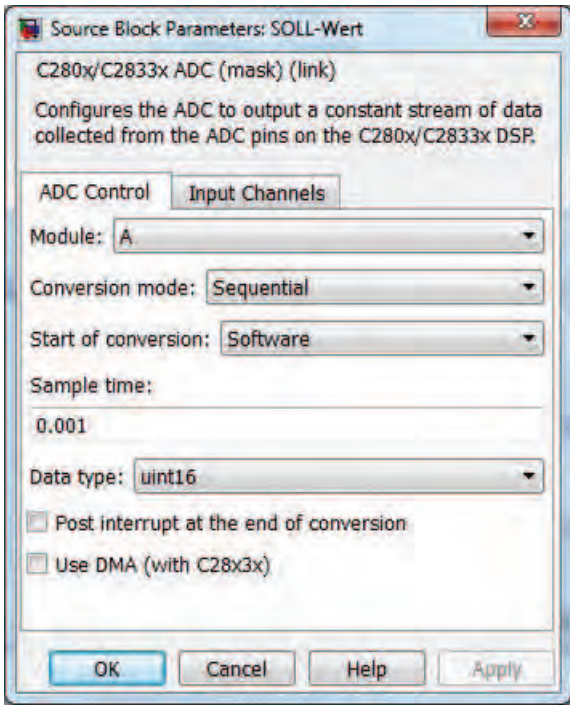


Abbildung 31: SOLL-Wert ADC Control

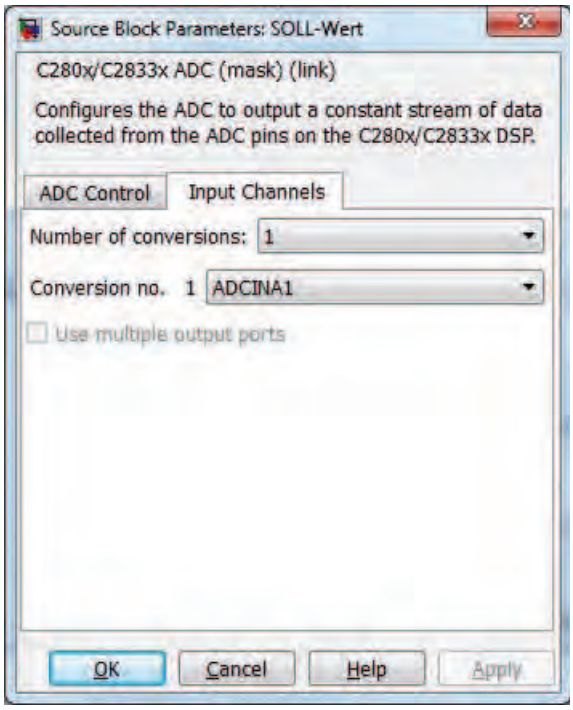


Abbildung 32: SOLL-Wert Input Channels

Der IST-Wert wird mit dem Schiebepotentiometer bestimmt, welches beim Peripheral Explorer Kit am analogen Eingang B2 angelegt wurde. Der gelieferte Wert ist vom Typ unit16. Die Einstellungen für den Baustein ADC sind in Abbildung 33 und 34 zu sehen.

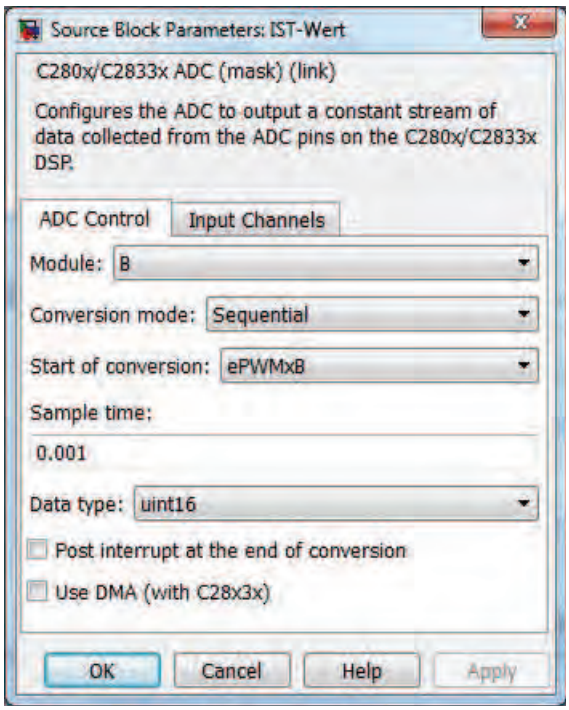


Abbildung 33: IST-Wert ADC Control

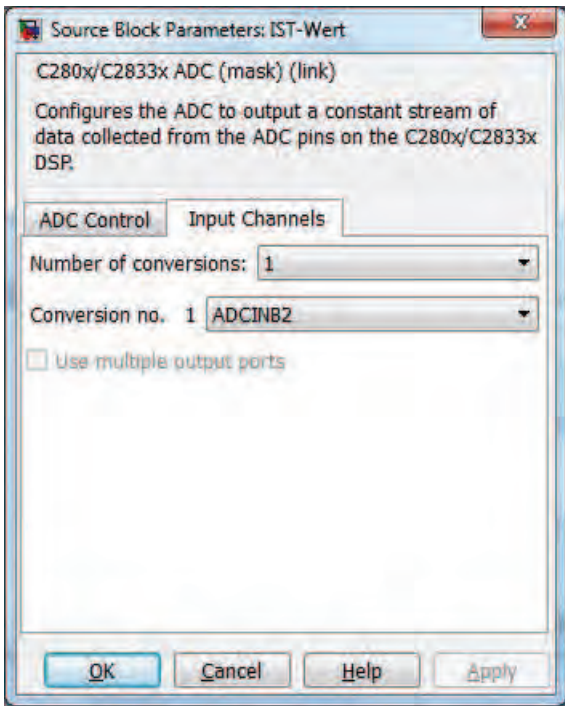


Abbildung 34: IST-Wert Input Channels

Die Konfiguration des ePWM Blocks erfolgt unter dem Reiter General. Die *Timer period* ist auf *4095* festgelegt. Das ist erforderlich, da der Drehpotentiometer eine 12-Bit Auflösung aufweist und infolgedessen eine PWM nur im Bereich zwischen 0 und 4095 realisierbar ist.

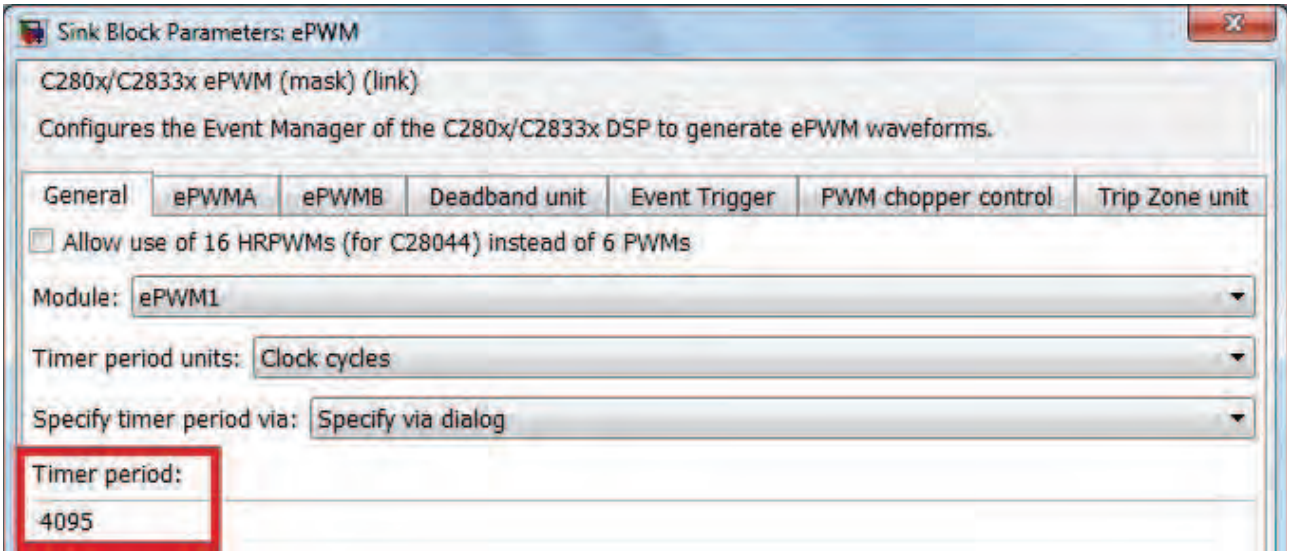


Abbildung 35: ePWM General

Um vom P-Regler den gelieferten Wert in ein PWM-Signal umzuwandeln, wird unter dem Reiter *ePWMA* die *Input port* Einstellung getroffen sowie der Anfangswert auf *0* gesetzt.

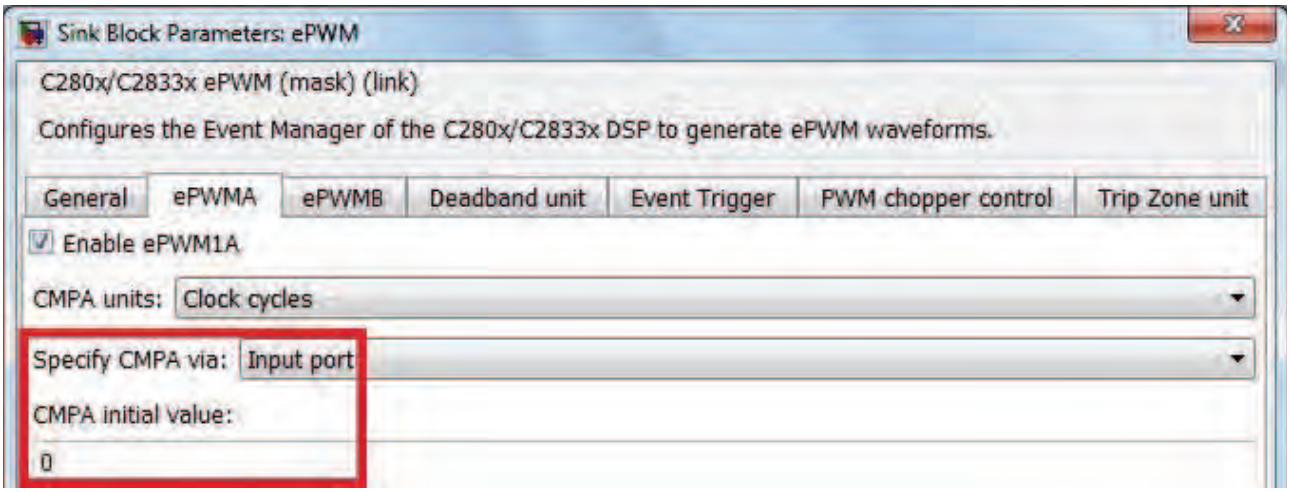


Abbildung 36: ePWMA

Die Subsysteme im Simulink-Modell (Abbildung 30) dienen der Umrechnung der von den Device-spezifischen Blöcken gelieferten Werte. Sie sind das „Software-Pendant“ zu den OPV-Schaltungen, die zwischen dem Peripheral Explorer Kit und den Ein- bzw. Ausgang an der Linearachse verschaltet sind (Kapitel 2.2). Damit wird im Modell so gerechnet, wie der P-Regler ausgelegt ist.

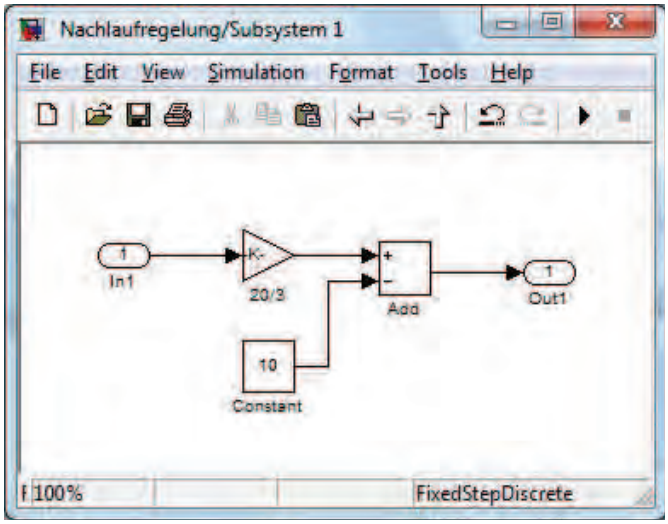


Abbildung 37: Subsystem 1

Die Bereichsanpassung aus dem ersten Subsystem führt aus dem Eingangswert der OPV zum Drehpotentiometer. Der Eingangswert der zwischen 0 und 3V liegt, wird in einen Ausgangswert zwischen -10V und 10V überführt.

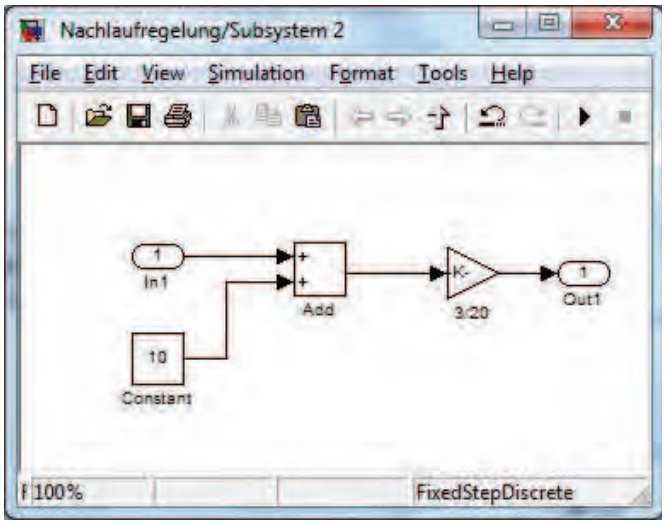


Abbildung 38: Subsystem 2

Im zweiten Subsystem wird aus der Ausgabe des P-Reglers eine Bereichsanpassung für den Ausgang auf 0 bis 3V vorgenommen. Dabei wird zuerst der Bereich von -10V bis 10V auf 0 bis 20V verschoben und dann über den Faktor 3/20 auf 0 bis 3V angepasst. Dies liefert den Eingang für den ePWM Block.

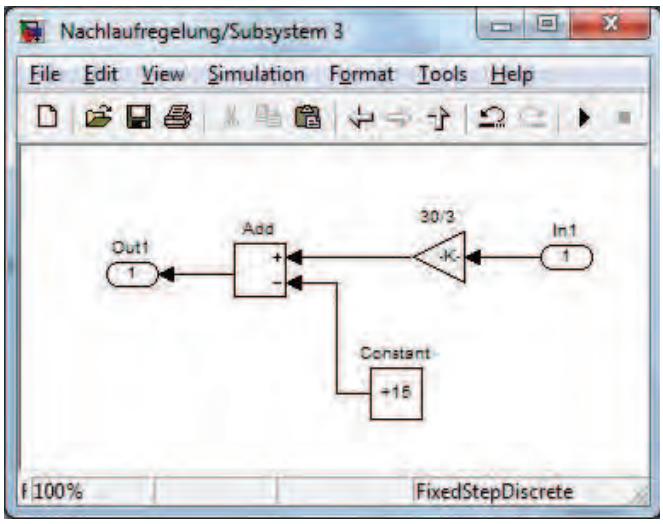


Abbildung 39: Subsystem 3

Das dritte Subsystem wandelt wiederum aus dem Eingangswert der OPV nach dem Schiebepotentiometer (0 bis 3V), einen für den Regler benötigten Bereich zwischen -15 und 15V durch.

Das Simulink-Modell für die Nachregelung ist erstellt und alle Parameter eingegeben. Dem folgend kommt die Code Erstellung. Dies wird ebenfalls in Simulink ausgeführt.

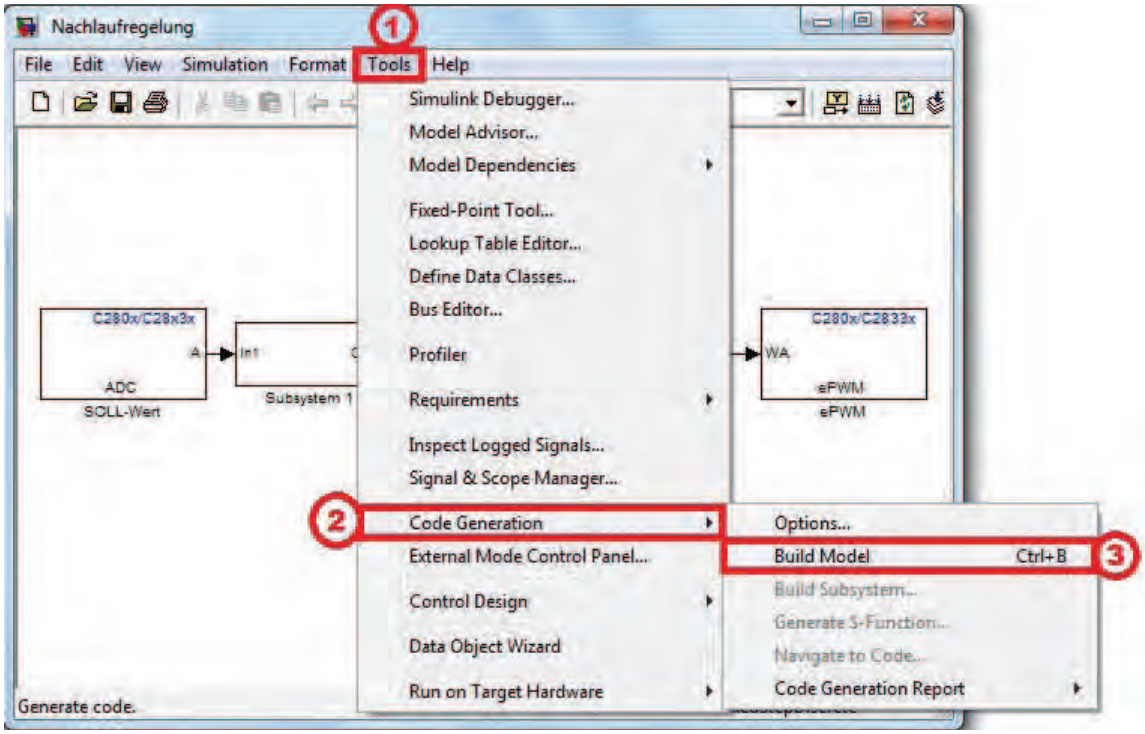


Abbildung 40: Build Model

Dazu wird in der Menüleiste unter *Tools* > *Code Generation* > *Build Model* (Abbildung 40) ausgewählt und der gesamte *Build* kann im Command Window von MATLAB verfolgt werden. Am Ende dieser Prozedur steht im Command Window ein *Build done* und *Download done*.
Im Workspace von MATLAB findet sich umgehend im Projektordner eine Output-Datei, die im nächsten Schritt Verwendung findet.

Nach dem Start von CCSv5, dem Verbinden und Einschalten des Peripheral Explorer Kit über das USB-Kabel mit dem PC wird die aktuelle Zielhardware-Konfiguration ausgewählt (Launch Selected Configuration) und somit eine Verbindung hergestellt (Connect Target), siehe Abbildung 41 und 42.
Der Import der *.out-Datei erfolgt über die Menüleiste (*Run* > *Load* > *Load Program*). Nach der erfolgreichen Ausführung auf dem Peripheral Explorer Kit, kann mit dem Drehpotentiometer 1 die Sollwertvorgabe durchgeführt werden und der Schlitten in die gewünschte Richtung nachgeführt.

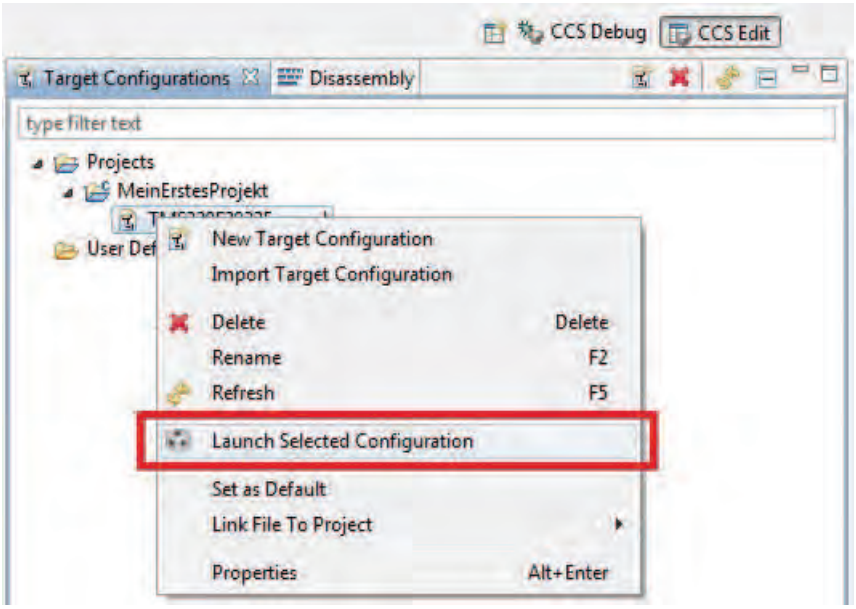


Abbildung 41: Launch Selected Configuration

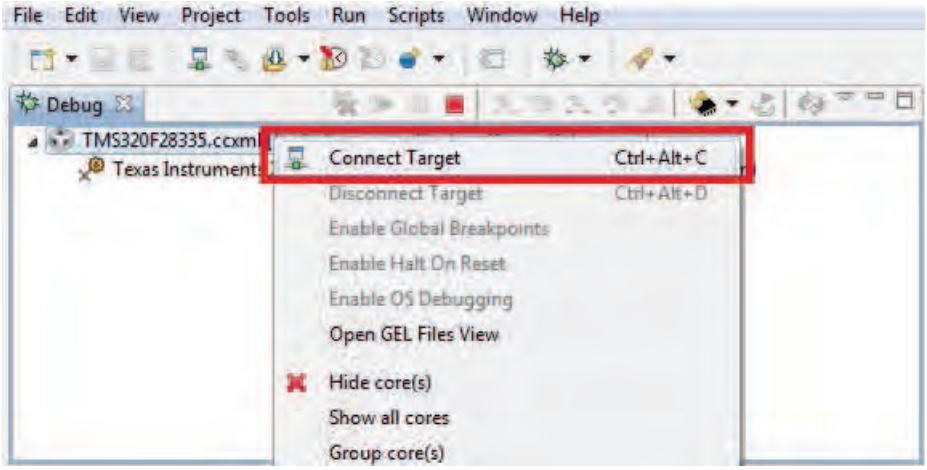


Abbildung 42: Connect Target

6. Schluss

Nach der erfolgreichen Durchführung dieser Anleitung für den modellbasierten Softwareentwurf besitzen die Studenten die Fähigkeit eigene Projekte zu planen und umzusetzen. Es wurden die verwendeten Programme MATLAB/Simulink sowie Code Composer Studio Version 5 erklärt, sowie die Erstellung eines Beispielprojekts und die anschließende Verwendung der Output-Datei damit ein Regler auf dem Delfino DSP läuft.

7. Quellenangabe

[1] TI C2000 Peripheral Explorer Kit, http://www.ti.com/tool/tmdsprex28335

[2] TMS320F28335 controlCARD, http://www.ti.com/tool/tmdscncd28335

[3] TMS320F28335, http://www.ti.com/product/tms320f28335

[4] Peripheral Explorer Kit Overview Quick Start (Rev.A), S.2 http://www.ti.com/litv/pdf/sprugm2a

[5] Andreas Bernhard, PraktikumNachlaufregREV20120201.pdf im Regelungstechnik Labor

[6] TMS320F28335 Data Manual, 4.7 ADC-Module, http://www.ti.com/litv/pdf/sprs439m

[7] http://www.mathworks.de/products/matlab/

[8] http://www.mathworks.de/products/simulink/

[9] http://www.mathworks.de/products/embedded-coder/

[10] http://processors.wiki.ti.com/index.php/Category:Code_Composer_Studio_v5

[11] C2000 Teaching Materials, Tutorials and Applications, Schulungs CD von TI, Module 3: F2833x Program Development Tools S. 13