Karlsruher Institut für Technologie
Institut für Technik der Informationsverarbeitung

Lastenheft zum Projektpraktikum Informationstechnik

Prof. Dr.-Ing. Eric Sax
M. Sc. Jijing Yan

Abgabefrist: 28. Kalenderwoche 09.-13.07.2018

Institutsleitung
Prof. Dr.-Ing. Dr. h. c. J. Becker
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

KIT - Die Forschungsuniversität in der Helmholtz-Gemeinschaft

Inhaltsverzeichnis

Kapitel 1

Vorwort

1.1 Willkommen im Entwicklungsteam der TivSeg AG!

Wir haben Sie eingestellt, da uns ein Kunde mit der Entwicklung einer neuen Software zur Steuerung eines Segway® ähnlichen Fahrgeräts beauftragt hat. Zur Verstärkung unseres Entwicklungs-Teams benötigen wir Sie dringend, um das gewünschte Produkt, den TivSeg™ zu entwickeln. Ihre Aufgabe für die nächsten Wochen wird die Planung und Umsetzung sowie das Testen und Erproben unseres neuen Produkts sein.

Bitte halten Sie sich bei der Planung an die Vorgaben unseres Kunden. Diese können Sie dem Abschnitt 4 entnehmen. Außerdem hat sich unser Unternehmen auf gemeinsame Richtlinien zur Formatierung des Quelltextes geeinigt, die Ihnen in einem extra Dokument ausgehändigt werden, bitte beachten Sie diese ebenfalls streng. Zusätzlich finden Sie noch unter dem Punkt Projektplanung die auf der letzten Teamsitzung beschlossene Zeitplanung.

Der benötigte Regler, welcher die Steuerung des TivSegs™ übernimmt, ist bereits vorhanden und muss von ihnen nicht mehr programmiert werden. Allerdings benötigen wir für diesen Regler noch eine Ansteuerung der verwendeten Hardwaremodule.

LaTeX ist ein Textsatzsystem zur Erstellung von PDF-Dokumenten, mit welcher die meisten wissenschaftlichen Arbeiten, wie z. B. Paper oder Abschlussarbeiten, angefertigt werden. Daher würden wir es begrüßen, wenn Sie Ihre Dokumentation mit LaTeX erstellen. Sollten Sie jedoch die Word-Vorlage verwenden, bitten wir Sie die Dokumentation vor jeder Abgabe in ein PDF zu konvertieren. Dies können Sie beispielsweise mit FreePDF[1] oder PDFCreator[2] erzeugen. Seit Word 2007 können PDF-Dateien auch direkt über ein Plug-In erstellt werden.

[1] FreePDF kann kostenlos unter `http://freepdfxp.de/` herunter geladen werden
[2] PDFCreator kann kostenlos unter `http://de.pdfforge.org/pdfcreator` herunter geladen werden

1.2 Übersicht über Aufgabenstellung

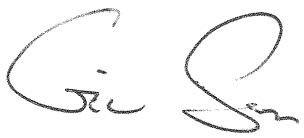
- Wir erwarten von Ihnen
  - zu Beginn, dass Sie ein Pflichtenheft anfertigen
  - am Ende des Projekts, zum letzten Termin, den Quellcode, den Sie Ihrem Gruppenleiter vorführen und erklären, sowie eine abgeschlossene Dokumentation der Software.
  - Eine abgeschlossene Dokumentation umfasst alle, im Kapitel 5 genannten Dokumente, sowie den Quellcode.
- Alle Teammitglieder sollen über alle Teile des Programms gut informiert sein.
- Zusätzlich muss der Quellcode auf dem TivSeg™ des Kunden ausführbar sein und diesen unfallfrei steuern können. Hierfür dient als Test der TivSeg™-Simulator unter `http://simplify.itiv.kit.edu/tivseg-sim`
- Sie müssen in C++ programmieren.
- Sie müssen die Programmierrichtlinien einhalten.
- Sie müssen den Zeitplan einhalten und zu allen Gruppenterminen erscheinen, um Ihre bisherigen Ergebnisse dem Gruppenleiter vorzulegen und diese zu besprechen.
- An den Abgabezeitpunkten der Meilensteine ist jeweils eine Dokumentation abzugeben, die während des gesamten Projektverlaufs durchgängig ergänzt wird. Die neue Dokumentation beinhaltet alle Punkte für den entsprechenden Termin sowie die letzte abgegebene Dokumentation.
- Sie sollten die Entwicklungsumgebung "Code Composer Studio (CCS)"[3] verwenden, wir sind jedoch auch offen für andere Entwicklungsumgebungen, wenn dies innerhalb Ihres Teams einheitlich gehandhabt wird.
- Während der Programmierphase dürfen und müssen Sie auch zu Hause arbeiten. Hierzu steht Ihnen zum Test Ihres Codes das TI Launchpad sowie der TivSeg™-Simulator zur Verfügung.
- Während der Gruppentermine stehen Ihnen fachkundige Gruppenleiter zur Seite, um Sie bei den verschiedenen Projektphasen zu beraten.
- Bei den offiziellen Review-Terminen müssen Sie anwesend sein und die geforderten Dokumente vorbereitet haben. Weiter müssen diese von jedem Teammitglied erklärt werden können.

[3] CCS kann unter `http://processors.wiki.ti.com/index.php/Download_CCS` kostenlos heruntergeladen werden.

1.3 Ziele des Projektpraktikums

Am Ende des Praktikums sollen Sie komplexe Probleme, dargeboten in natürlicher Sprache (Spezifikation), in einfache und übersichtliche Module und dementsprechend passende Algorithmen und Datenstrukturen anwenden können. Bei der Umsetzung in einen strukturierten und lauffähigen Quellcode unter Einhaltung von vorgegebenen Qualitätskriterien (u. a. Programmierrichtlinien) soll das Schreiben komplexer C++ Codeabschnitte und der Umgang mit einer integrierten Entwicklungsumgebung geübt werden. Der hierdurch entstandene Code soll auf einem Mikrocontroller lauffähig sein, wodurch Kenntnisse der hardwarenahen Programmierung erlernt werden. Während des Praktikums muss das Gesamtprojekt in Teilprojekte gegliedert werden, wodurch das teamorientierte selbstständige Arbeiten in Ihrer Gruppe gestärkt wird und das Projekt frei gestaltet werden kann. Darüber hinaus gehört zum erfolgreichen Projektpraktikum das Bewerten des geschriebenen Programms durch Erstellung von Testprogrammen sowie die Abgabe einer Projektdokumentation.

Viel Spaß und Erfolg bei Ihrer Arbeit!
TivSeg AG



Prof. Dr.-Ing. Eric Sax
Vorstandsvorsitzender

Kapitel 2

Einleitung

2.1 Motivation

Ein TivSeg™ ist ein elektrisch angetriebenes Transportmittel mit nur einer Achse. Die zu befördernde Person steht hierbei zwischen den beiden Rädern der Achse und kann sich an einer Lenkstange festhalten. Durch eine elektronische Antriebsregelung hält sich der TivSeg™ selbst in Balance. Jedes Rad wird hierbei per Einzelradantrieb von einem separaten Elektromotor angetrieben. Die unterschiedliche Drehzahlen der Räder ermöglichen dabei eine Kurvenfahrt wie bei Kettenfahrzeugen.

Ein Schwenken der Lenkstange nach rechts oder links bewirkt die dementsprechende Kurvenfahrt. Sobald sich der Fahrer mit der Lenkstange zur Seite neigt, wird das von den Neigungssensoren wahrgenommen und das jeweilige Rad dreht sich langsamer und verursacht dadurch die Kurvenfahrt.

Ein elektronischer Regelkreis lässt den TivSeg™ automatisch in die Richtung fahren, in die sich der Fahrer lehnt. Sobald die Neigungssensoren registrieren, dass sich der Fahrer nach vorne oder hinten neigt, drehen sich die Räder in diese Richtung. Die Fortbewegung wird ausschließlich durch solche Gewichtsverlagerungen gesteuert, es gibt keine Bedienelemente zum Bremsen oder Beschleunigen. Diese Funktionsweise entspricht dem aufrechten Gang, bei dem sich der Schwerpunkt des Körpers stets über der Auflagefläche der Füße befindet. Der TivSeg™ ist deshalb intuitiv zu bedienen.[1]

Durch die hohe Nachfrage an TivSegs™ für touristische Stadtführungen, als Patrouillenfahrzeug bei Polizeibehörden sowie der erhöhten Nachfrage des Lehrpersonals auf dem Weg zu Lehrveranstaltungen, hat sich die TivSeg AG entschlossen, ab Januar des nächsten Jahres in die Serienfertigung des TivSegs™ einzusteigen.

2.2 Problemstellung

Die Firma TivSeg AG hat bereits in der Vergangenheit eine TivSeg™-Plattform entwickelt, die in Abbildung 2.1 bei einer Probefahrt gezeigt ist. Für den Prototypen wurde jedoch ein Entwicklungsboard eingesetzt, das für die Serienproduktion ausgetauscht werden muss. Daher besteht der TivSeg™ zur Zeit lediglich aus einem mechanischen Modul, das durch Sensoren und Aktoren vervollständigt wurde sowie einem Regelalgorithmus. Für die neuen Prozessoren im TivSeg™ existiert jedoch noch keine Ansteuerlogik der Sensoren und Aktoren.

[1] vgl. auch `http://de.wikipedia.org/wiki/Segway_Personal_Transporter`

Abbildung 2.1: TivSeg™ Prototyp bei einer Erprobung

Zur Steuerung der Sensorik und Aktorik soll auf einem Mikrocontroller eine hardwarenahe Programmierlösung gefunden werden. Hierfür soll das von Ihnen bereits bekannte Launchpad Tiva™ C verwendet werden. In Kapitel 4 werden alle Schnittstellen zur bereits bestehenden Hardware und Mechanik aufgezeigt und erläutert. Im folgenden Dokument wird näher auf die bereits vorhandene Hard- und Software eingegangen sowie die Funktionen der zu implementierenden Module aufgezeigt.

Kapitel 3

Beschreibungen und Anforderungen

3.1 Funktionale Beschreibung

Um eine möglichst intuitive Bedienung des TivSegs™ zu erzeugen, soll dieser nur über eine Gewichtsverlagerung und das Kippen des Lenkhebels gesteuert werden. Dadurch soll es unseren Kunden am ITIV ermöglicht werden, in künftigen Lehrveranstaltungen ohne Schürfwunden oder Knochenbrüche anzukommen.

Für die Steuerung des TivSegs™ sollen daher die Sensorwerte der Neigung und des Lenkhebels abgefragt werden und die Fahreigenschaften entsprechend angepasst werden. Hierfür ist es notwendig diese Sensoren in regelmäßigen Abständen (10 ms) mit harter Echtzeitanforderung abzufragen. Innerhalb dieser Regelzeit müssen die Sensordaten abgefragt und verarbeitet werden.

Neigt sich der TivSeg™ beispielsweise nach vorne, soll beschleunigt werden. Hierfür wird die Pulsweite (siehe Kapitel 3.2.3) erhöht und die Motoren somit mit mehr Energie versorgt. Beim Verringern des Neigungswinkels wird die Pulsweite verringert und damit der TivSeg™ gebremst. Wird der TivSeg™ im Stillstand nach hinten geneigt, soll ein Rückwärtsfahren möglich sein.

Über ein Potentiometer kann der Winkel der Lenkstange abgefragt werden. Mit Hilfe dieses Winkels wird der Drehzahlunterschied zwischen rechtem und linkem Rad errechnet. Ein Lenken nach links erfolgt beispielsweise durch das Abbremsen des linken Rads. Das Lenken nach rechts entsprechend durch das Abbremsen des rechten Rads.

Zusätzlich haben wir im Rahmen unseres TivSegs™ einen Fußschalter verbaut. Dieser soll sicherstellen, dass ein Fahrer auf dem TivSeg™ steht, bevor losgefahren werden kann. Zusätzlich setzt dieser Fußschalter beim Aufsteigen auf den TivSeg™ den Offset der Sensorwerte auf Null. Dadurch wird sichergestellt, dass der TivSeg™ initialisiert wird und nicht direkt beim Aufsteigen unkontrolliert losfährt.

3.2 Überblick über vorhandene Module des TivSegs™

Als Plattform dieses Projekts dient der TivSeg™, der mechanische Rahmen ist in Abbildung 3.1 dargestellt und besteht hauptsächlich aus einem Aluminiumrahmen mit Lenkstange, zwei in Serie geschalteten 12 V Batterien und zwei Gleichstrommotoren, die über ein festes Getriebe mit den Einzelrädern verbunden sind.

Abbildung 3.1: Mechanischer Rahmen des TivSegs™

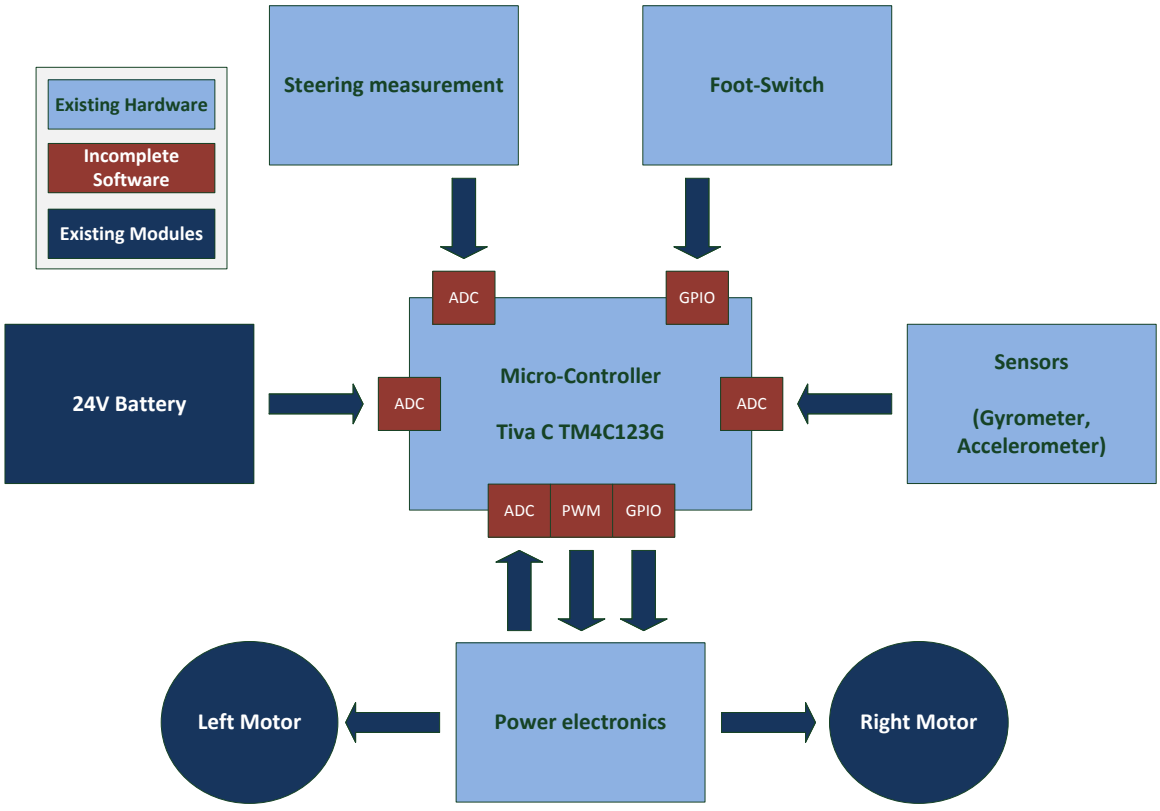


Abbildung 3.2: Übersicht über den TivSeg™-Aufbau

Als weitere Bauteile wurden im TivSeg™ eine Leistungsplatine, Lagesensoren, Fußschalter, Potentiometer zur Messung des Lenkausschlags sowie eine Steuerelektronik verbaut. Wobei Lagesensoren und Potentiometer zur Messung des Lenkausschlags als analoge Sensoren zu verstehen sind, deren Ergebnisse im Regelalgorithmus (siehe Kapitel 3.2.2 und Kapitel 4) verarbeitet und anschließend über ein Puls-Weiten-Moduliertes-Signal (PWM) an die Leistungselektronik geführt werden. Eine genaue Beschreibung der einzelnen elektrischen Module entnehmen Sie bitte Kapitel 3.2.3.

3.2.1 Beschreibung der vorhandenen Software

Für den TivSeg™ wurde eine Vielzahl an Softwaremodulen angefertigt, welche im TM4C123G implementiert werden können und im vorliegenden Projekt als Sourcecode eingefügt sind. Hierzu gehört beispielsweise auf Anwendungsebene (Application Layer) der Regelalgorithmus, die Lageregelung, die Motorsteuerung, u. v. m. In Abbildung 3.3 sind die Softwareblöcke aufgezeigt, die für dieses Projekt verwendet werden sollen. Alle Softwaremodule innerhalb der Anwendungsebene sind bereits vorhanden und getestet. Im Bereich der Hardwareabstraktionsschicht muss jedoch noch die Ansteuerung der Sensoren und Aktoren implementiert werden.

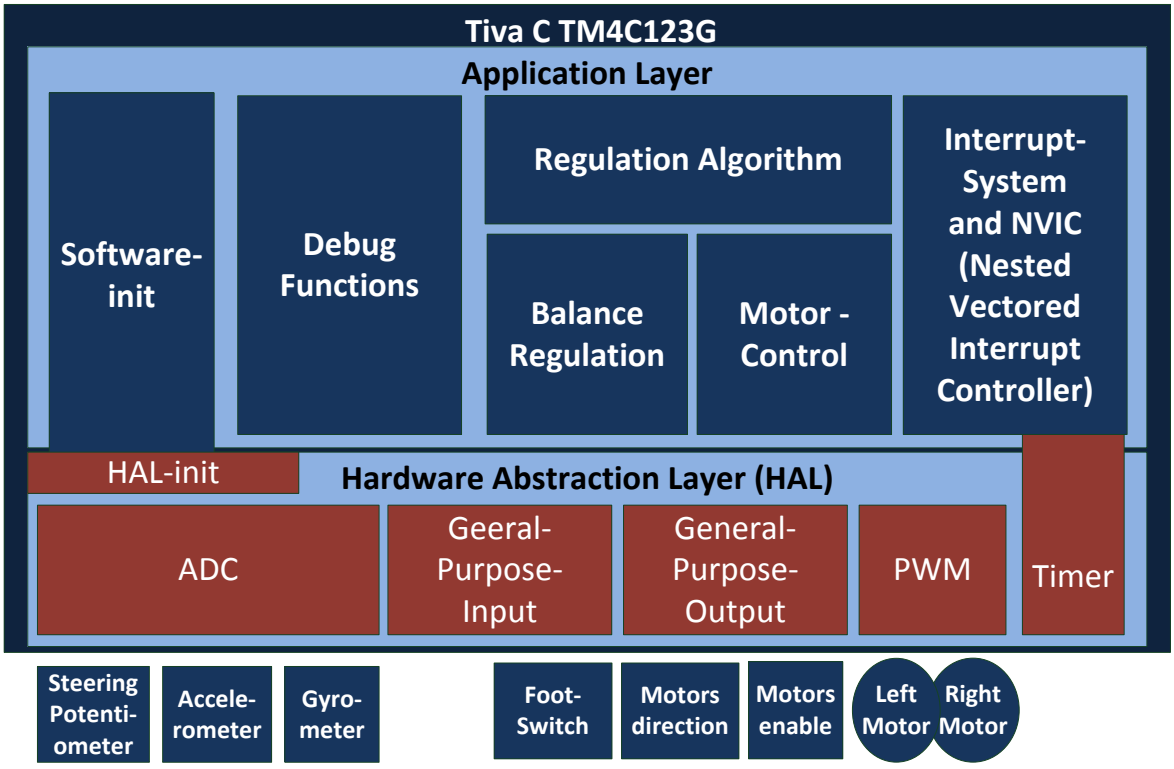


Abbildung 3.3: Blockdiagramm für die TivSeg™-Software

Das heißt, um einen funktionsfähigen TivSeg™ herstellen zu können, ist es wichtig, dass Sie die Hardwareansteuerung der Sensoren und Aktoren implementieren. Eine Übersicht über alle Sensoren und Aktoren wurde in Abbildung 3.2 und Abbildung 3.3 vorgestellt, wobei die rot markierten Module noch implementiert werden müssen. Wie in dieser Abbildung gezeigt wird, muss die Ansteuerung der noch fehlenden Module `ADC`, `PWM` und `GPIO` realisiert werden. Zusätzlich wird

eine `Timer`-Funktion benötigt, welche regelmäßig einen Interrupt ausführt, um die Steuerung des TivSegs™ zu aktivieren. Da der Regelalgorithmus des TivSegs™ durch die Interrupts des Timers ausgeführt wird, ist es nötig, den Timer auf genau 100 Hz einzustellen.

3.2.2 Beschreibung der vorhandenen Regelung

Für den Regelalgorithmus des TivSegs™ konnten wir in der Vergangenheit einen MatLab[1]-Algorithmus erwerben. Als Zusatzaufgabe können sie sich mit diesem Algorithmus vertraut machen. Für die Programmierung des HAL (Hardware Abstraction Layer) ist dies jedoch nicht zwingend nötig.

Der Kern des TivSeg™-Reglers ist ein PI-Regler, dessen Eingänge (Gyrometer und Beschleunigungssensor) zuvor entsprechend gewichtet werden. Über post-Scaler kann die Ansteuerung der Motoren noch verfeinert werden.

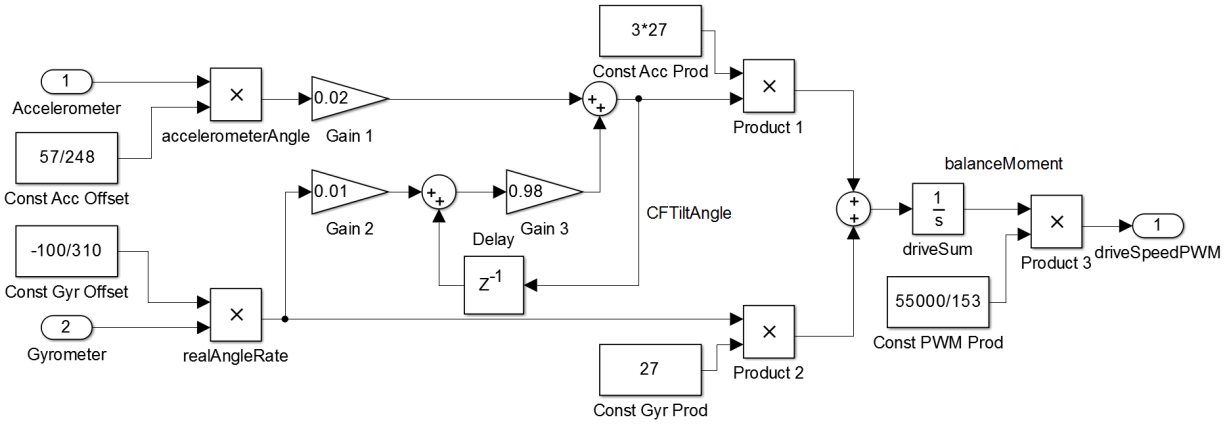


Abbildung 3.4: Matlab Regelalgorithmus des TivSegs™

Dieser Regler kann über die mitgelieferte GUI gesteuert werden. Beides (Regler und GUI) können Sie in der Datei `TivSeg_Regelalgorithmus_Matlab.mdl` einsehen.

Dieser Regelalgorithmus steuert jedoch nur die Fahrt gerade aus. Die Auswertung der Lenkstange ist hier nicht berücksichtigt, wurde jedoch im vorliegenden Code (`segway.cpp`) bereits implementiert. In den mitgelieferten Quellcode-Dateien können sie einen funktionsfähigen Regelalgorithmus finden. Dieser übernimmt sowohl das Berechnen der Geschwindigkeit mittels übergebenen Sensorwerten als auch die Regelung der Lenkung.

3.2.3 Beschreibung der vorhandenen Hardware

Für den Aufbau des TivSegs™ wird auf eine bereits vorhandene Hardwareplattform zurückgegriffen. Das Herzstück des TivSegs™ besteht wie das Tiva™ C Launchpad aus einem TM4C123G der Firma Texas Instruments.

[1]MatLab kann durch eine TAH-Lizenz für Studium und Lehre frei genutzt werden.
Nähere Infos unter `http://www.scc.kit.edu/produkte/3841.php`

Steuerplatine des TivSeg™

Um ihr Launchpad mit dem TivSeg™ zu verbinden, wurde eine spezielle Steuerplatine entworfen, an welcher die verschiedenen Sensoren und Aktoren über Steckverbindungen angeschlossen werden.

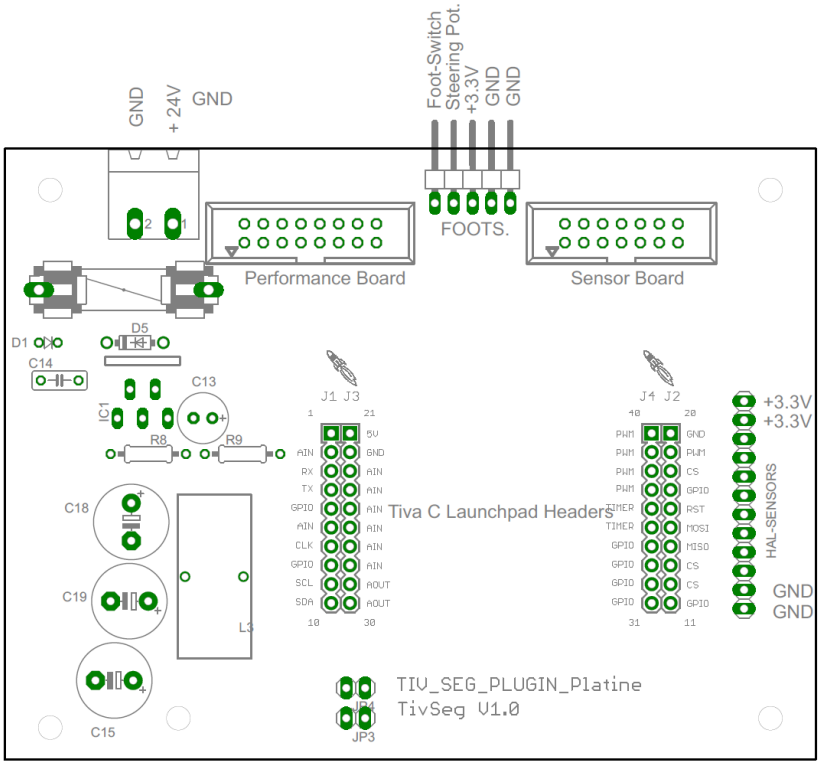


Abbildung 3.5: TivSeg™-Steuerplatine V1

Diese Steuerplatine ist bereits fest im TivSeg™ verbaut und muss nicht verändert werden. Lediglich die unter Kapitel 3.2.1 aufgeführten Software-Module müssen noch im TM4C123G ergänzt werden.

Im TM4C123G des Launchpads wird der Code des Regelalgorithmus ausgeführt. Selbiger kann in der mitgelieferten `Segway.cpp`, sowie im Matlab-Algorithmus (siehe auch Kapitel 3.2.2) eingesehen werden. Dieser Regelalgorithmus ist bereits funktionsfähig vorhanden und muss nicht verändert werden.

Messung des Lenkausschlags

Um den Lenkwunsch des Fahrers zu messen, wurde am Drehpunkt der Lenkstange ein Potentiometer angebracht. Über eine konstant angelegte Spannung von 3.3 V zwischen Pin 1 und Pin 2 entsteht ein Spannungsteiler am Ausgang des Potentiometers (Pin 3). Durch Drehen des Potentiometers kann dieser Spannungsteiler variiert werden. Dadurch ist es möglich über einen Analog-Digital-Converter (ADC) die Stellung der Lenkstange auszuwerten. Der prinzipielle Anschluss der Lenkstange an den TM4C123G ist in Abbildung 3.6 dargestellt.

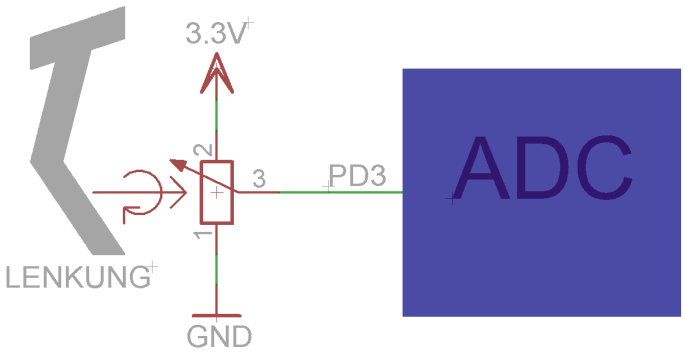


Abbildung 3.6: Messung des Lenkausschlags über ein Potentiometer

Lagesensoren

Zur Regelung des TivSegs™ wird ein Beschleunigungssensor (Accelerometer) und ein Drehratensensor (Gyrometer) benötigt. Diese geben als Messergebnis eine Spannung im Bereich zwischen 0 und 3.3 V zurück, welche mit dem ADC digitalisiert werden kann. Der ADC hat eine Genauigkeit von 12 bit und liefert damit einen Digital-Wert im Bereich von 0 bis 4095. Da der Regler für einen ADC mit 10 bit realisiert wurde, muss der vom ADC zurückgegebene Wert auf diesen Zahlenbereich umgerechnet werden, bevor er an die Regelung des TivSeg™ zurückgegeben werden kann! Durch diese beiden Sensoren wird die Neigung, Beschleunigung und damit die Soll-Geschwindigkeit des TivSegs™ ermittelt.

Ansteuerung der Gleichstrommotoren

Der TivSeg™ wird durch zwei 500 W Gleichstrommotoren angetrieben, wobei jeder Motor jeweils ein Rad antreibt. Um Gleichstrommotoren jedoch geschwindigkeitsabhängig regeln zu können, gibt es zwei prinzipielle Vorgehensweisen. Erste Möglichkeit ist, über eine Spannungsregelung die angelegte Spannung und damit die Leistung im Motor zu steuern. Als zweite Variante kann über die Modulation der Anschalt- und Pausezeit (Puls-Weiten-Modulation, PWM) die Leistung am Motor gesteuert werden. Vorteil dieses Verfahrens ist die einfache Ansteuerung durch einen Mikrocontroller über eine sogenannte H-Brückenschaltung ohne die Verwendung eines Digital-Analog-Wandlers. Diese H-Brückenschaltung ist im Anhang C dargestellt. Des Weiteren ist in diesem Anhang die gesamte Leistungsplatine dargestellt.

3.3 Nichtfunktionale Anforderungen

3.3.1 Zuverlässigkeit

Da unsere Firma in den weltweiten Markt der TivSeg™-Zulieferer einsteigen will, erwarten unsere Kunden ein hohes Maß an Zuverlässigkeit. Die Zuverlässigkeit unseres Produkts sollen Sie durch entsprechende Tests und eine stetige Dokumentation des Qualitätsmanagements nachweisen.

3.3.2 Erweiterbarkeit und Wartbarkeit

In späteren Projektpraktika soll der TivSeg™ um weitere Funktionen ergänzt werden können. Diese Erweiterungen gehen zum Beispiel vom selbstständigen Stehen des TivSeg™ bis hin zu einem autonomen Fahren. Um diese Erweiterbarkeit, aber auch eine entsprechende Wartbarkeit gewährleisten zu können, müssen alle Module mit Kommentaren versehen und dokumentiert werden. Zudem soll für eine bessere Verständlichkeit keine große Gesamtsoftware in einem Modul erstellt werden, sondern das Gesamtprojekt aus vielen kleineren Funktionen zusammen gesetzt werden.

Des Weiteren wird von uns vorausgesetzt, dass sie den Quellcode gemäß der Programmierrichtlinien erstellen. Dies dient der Übersichtlichkeit und damit der Erweiterbarkeit und Wartbarkeit.

3.3.3 Dokumentation des Quellcodes

Durch spezielle Kommentare im Quelltext können Softwareentwickler Erläuterungen zu Programmelementen definieren, aus denen Doxygen[2] eine übersichtliche Dokumentation erstellt. Außerdem ist es möglich, einen zusammenfassenden Überblick über den Aufbau und die Elemente eines bereits existierenden Programms (verwendete Dateien, Funktionen, Variablen sowie deren jeweiligen Initialisierung und die Rolle im Programmablauf) zu erzeugen. Das Ziel einer solchen Dokumentierungsmethode ist die Vermeidung von Redundanz. Eine Dokumentierung der Strukturen (Funktionen, Klassen, Methoden usw.) in einer separaten Entwicklerdokumentation is fehleranfällig, da sich der Quellcode im Laufe einer Entwicklung häufig ändern kann.

[2] DoxyGen ist unter `www.doxygen.org` kostenlos verfügbar

Kapitel 4

Schnittstellen-Spezifikation

4.1 Hardware-Spezifikation

4.1.1 Mikrocontroller

Da Sie über ein eigenes Tiva™ C Launchpad verfügen und damit hoffentlich bereits erste Erfahrungen gesammelt haben, wurde entschieden das TivSeg™ auf Basis dieses Mikrocontrollers aufzubauen. Dies hat außerdem den Vorteil, dass Sie Ihre Module zuerst auf Ihrem Launchpad testen können, bevor wir dieses in das TivSeg™ integrieren. Hierzu können und sollen Sie auf die Debug-Funktionalitäten ihres IDE zugreifen[1].

4.1.2 Motorsteuerung

Im TivSeg™ sind zwei Motoren für die jeweiligen Räder verbaut. Durch eine gezielte Ansteuerung ist es durch diesen Einzelradantrieb möglich das gesamte TivSeg™ zu drehen. Durch eine Puls-Weiten-Modulation (PWM) ist es möglich die Geschwindigkeit der Motoren zu regeln.

Zusätzlich zur Geschwindigkeit, die über PWM gesteuert werden kann, muss durch einen GPIO-Pin die Richtung geschaltet werden. Ein 'High'-Level am entsprechenden Ausgang führt zu einer Vorwärtsbewegung der Motoren, ein 'Low'-Level zu einer Rückwärtsbewegung. Des Weiteren müssen die Motoren über ein 'Enable'-Signal aktiviert werden ('active High'). Dieses Signal ist für beide Motoren gleich. In Tabelle 4.1 oder Anhang A ist die Belegung der Pins aufgelistet.

4.1.3 Puls-Weiten-Modulation (PWM)

Die PWM-Frequenz kann in einem großen Bereich beliebig eingestellt werden. Dieser ist durch die Größe der Zählregister in Verbindung mit dem Prescaler begrenzt. Das Pulsweiten-Verhältnis kann durch eine Zahl von 0 bis 255 festgelegt werden. Wobei 0 keine Spannung und 255 reine Gleichspannung bedeutet. Im TivSeg™ soll für beide Motoren eine PWM-Frequenz von 18 kHz verwendet werden.

WICHTIG: Zu beachten ist jedoch, dass die Motoren bei Gleichspannung nicht mehr funktionieren. Daher dürfen diese nur mit maximal 60 % des Maximalwertes betrieben werden. Dies ist im Programmcode unter der Variable `Configuration::PWM_maxPWMRatio` abgespeichert. Bitte

[1] CCS besitzt ausgiebige Debug- und Verifikations-Tools. Siehe z.B. `http://processors.wiki.ti.com/index.php/Debug_Handbook_for_CCS`

verwenden Sie diese Variable als Maximalwert in Ihrem Code.

4.1.4 Sensordaten

Durch den integrierten Analog-Digital-Wandler (ADC) wird das Messen von bis zu 12 Kanälen mit einer Auflösung von 12 Bit ermöglicht. Über ihn werden die Werte der verschiedenen Sensoren eingelesen. Bei der Messung werden die Analogen Eingänge je nach Konfiguration nacheinander durch einen Multiplexer auf ein ADC-Modul geschaltet. Die Messergebnisse werden, je nach verwendetem Sample-Sequenzer (Siehe Datenblatt des TM4C123GS.802 ff), in dessen FIFO-Register geschrieben, welches nach jeder Messung komplett ausgelesen werden muss, um Overflow zu verhindern.

Um Spannungen messen zu können, müssen die entsprechenden GPIO-Pins als Analoge Eingänge konfiguriert und dem ADC zugeschaltet werden (siehe Datenblatt S. 658).

Table 10-3. GPIO Pad Configuration Examples *(continued)*

Configuration	GPIO Register Bit Value[a]									
	AFSEL	DIR	ODR	DEN	PUR	PDR	DR2R	DR4R	DR8R	SLR
Digital Input (QEI)	1	X	0	1	?	?	X	X	X	X
Digital Output (PWM)	1	X	0	1	?	?	?	?	?	?
Digital Output (Timer PWM)	1	X	0	1	?	?	?	?	?	?
Digital Input/Output (SSI)	1	X	0	1	?	?	?	?	?	?
Digital Input/Output (UART)	1	X	0	1	?	?	?	?	?	?
Analog Input (Comparator)	0	0	0	0	0	0	X	X	X	X
Digital Output (Comparator)	1	X	0	1	?	?	?	?	?	?

a. X=Ignored (don't care bit)

?=Can be either 0 or 1, depending on the configuration

Abbildung 4.1: Konfiguration der GPIO-Register zur Verwendung als Analog Input

4.1.5 Fußschalter

Um den TivSeg™ gegen ungewolltes Anfahren zu schützen, wurde ein Fußschalter verbaut (siehe Kapitel 3). Dieser Schalter ist durch ein GPIO (General-Purpose-Input-Output) an Pin PA2 realisiert. Ein drücken des Fußschalters unterbricht die Verbindung von PA2 mit GND und soll einen 'High'-Pegel am entsprechenden Eingang erzeugen, welcher auch ins Hauptprogramm weitergegeben werden muss. Statt des sofortigen elektrischen Kontaktes ruft die Betätigung des Schalters kurzzeitig ein mehrfaches Schließen und Öffnen des Kontakts hervor. Dieses sogenannte Prellen muss jedoch unterbunden werden. Daher ist es wichtig, den Taster des Fußschalters zu entprellen.

4.1.6 Anschlussbelegung

Die Anschlussbelegung der Sensoren und Aktoren ist in Tabelle 4.1 zusammengefasst und ist in Anhang A als Auflistung über alle angeschlossenen Peripherien gegeben.

Name	Beschreibung	IO-Pin
Motor links	Erfordert ein PWM-Signal zur Ansteuerung des linken Motors	PC4
Motor rechts	Erfordert ein PWM-Signal zur Ansteuerung des rechten Motors	PC5
Richtung rechts	Richtungssteuerung des rechten Motors	PC6
Richtung links	Richtungssteuerung des linken Motors	PC7
Motor Enable	Aktivierung beider Motoren	PD6
Accelerometer	Beschleunigung der Y-Achse	PE1
Gyrometer	Drehrate der Y-Achse	PE3
V_{Ref}	Gyrometer-Referenzspannung	PE2
Lenkung	Potentiometer zum Messen des Lenkeinschlags	PD3
Batterie-Spannung	Spannungsteiler an Batteriespannung	PD2
Fußschalter	Auslesen des Fußschalters	PA2
Current Sense R	Strommessung des rechten Motors	PE5
Current Sense L	Strommessung des linken Motors	PE4

Tabelle 4.1: Belegungsplan der Ein-/Ausgangspins

4.2 Software-Spezifikation

4.2.1 Beschreibung der Klassen und Module

Für den TivSeg™wurde wie bereits in Kapitel 3.2.1 beschrieben der Regler implementiert. Zusätzlich wurde das in Abbildung 4.2 dargestellte Klassendiagramm für die Struktur der Software erarbeitet, wobei die Klassen `main`, `Segway` und `Configuration` bereits erstellt wurden. Die Klassen `ADCSensor`, `ADC`, `GPIOSensor`, `Motor`, `PWM`, sowie `Timer` sind lediglich angelegt jedoch inhaltlich nicht vorhanden.

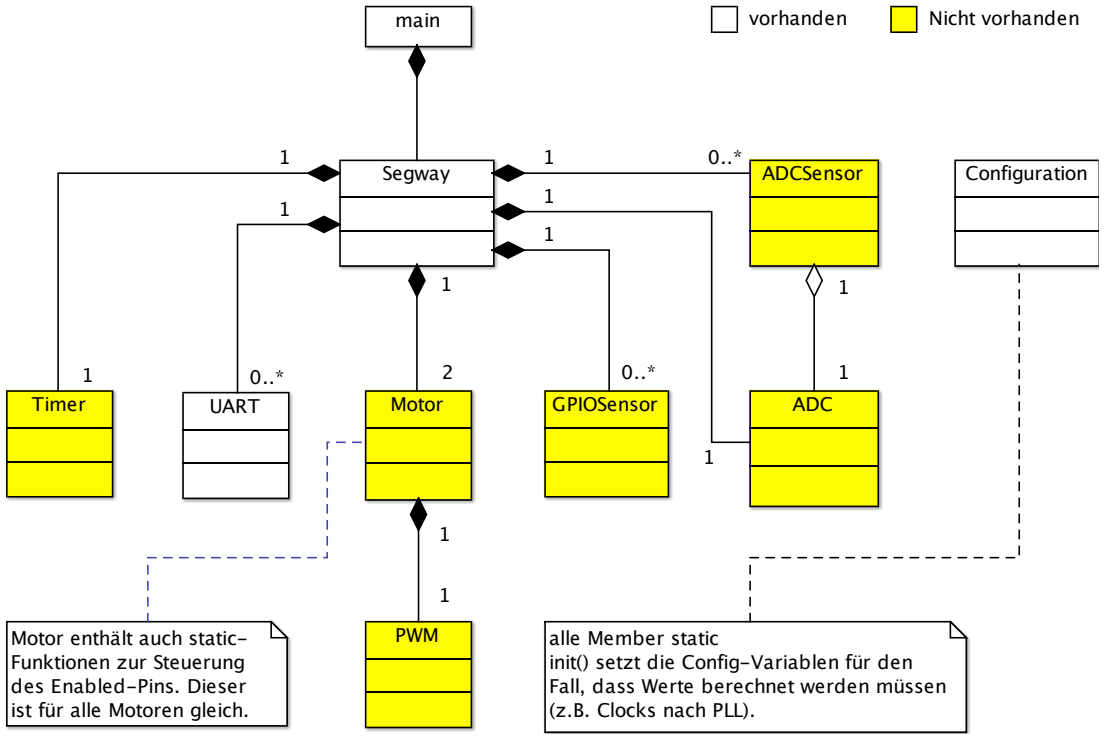


Abbildung 4.2: Klassendiagramm für die TivSeg™ Software

Ihre Aufgabe ist es nun, diese hardwarenahen Klassen zu implementieren. Im Folgenden sind die Schnittstellen zu diesen Klassen kurz erläutert. Eine Übersicht über die verwendeten Schnittstellen des Gesamtprojekts ist in `TivSeg-Klassenbeschreibung.pdf` beschrieben.

Schnittstellen Motor-Klasse

bool `init( Configuration::s_MotorConfig *thisMotorConfig_ )`

Diese Funktion erwartet als Parameter einen Zeiger auf das entsprechende Motorobjekt und gibt **true** bei erfolgreicher Initialisierung zurück. Sie konfiguriert den Richtungspin für den jeweiligen Motor und initialisiert/aktiviert dessen PWM Objekt.

struct `s_MotorConfig`

Diese Struktur beinhaltet die Konfiguration der Motoren und ist in `configuration.h` implementiert. **unsigned char** `directionPinPort` zum Übergeben des Ports für den Richtungspin (2 = Port PC), **unsigned long** `directionPinPin` zum Übergeben des Pins am entsprechenden Port, **bool** `directionPinForwardValue` für die Definition vorwärts Richtung des Motors (0 = low für Rückwärtsfahrt, 1 = high für Vorwärtsfahrt) und `s_PWMConfig* PWMConfig` beinhaltet die Konfiguration der PWM.

struct `s_PWMConfig`

Diese Struktur beinhaltet die Konfiguration der Puls-Weiten-Modulation und ist in `configuration.h` implementiert. **unsigned char** `channelID` beinhaltet den Kanal der PWM, **unsigned char** `maxPWMRatio` speichert das Puls-Weiten-Verhältnis der PMW, **unsigned long** `frequency` beinhaltet die entsprechende Frequenz der PWM, **unsigned char** `GPIO_port` beinhaltet den Ports für den PWM Ausgang (2 = Port PC), **unsigned char** `GPIO_pin` beinhaltet den Pin am entsprechenden Port auf dem die PWM ausgegeben wird und die Bits zum Setzten der PWM Funktion über das GPIOPCTL-Register wird in **unsigned char** `GPIO_Pin_Mux_Assignement` abgespeichert.

static void `initEnablePin( void )`

Initialisiert den Ausgangspin für das Enable-Signal der Motoren. (Setzten der GPIODIR und GPIODEN Register sowie setzen des Ausgangspins auf low.)

static void `setEnabled( bool enabled )`

Aktiviert oder deaktiviert die Motoren. Diese Funktion ist für beide Motoren gleich. Der Übergabeparameter `enabled` ist **true** wenn der Motor aktiviert sein soll.

static bool `getIsEnabled( void )`

Gibt den Enable-Zustand der Motoren zurück.

bool `setSpeed( unsigned char ratioOn )`

Setzt die übergebene Geschwindigkeit für die Motoren. Zulässige Werte sind 0 bis `PWM_maxPWMRatio`. Wobei `Configuration::PWM_maxPWMRatio` durch die Konfigurationsklasse vorgegeben wird. Gibt **false** zurück, wenn der Maximalwert überschritten wurde, sonst **true**. Bei **false** wird der Wert nicht gesetzt.

unsigned char `getSpeed( void )`

Gibt die beim letzten erfolgreichen Aufruf von `setSpeed` gesetzte Geschwindigkeit zurück.

void `setDirection( bool forward )`

Setzt die Richtung der Motoren über das entsprechende Steuerbit.

void `Motor_CleanUp( void )`

Wird vom Destruktor genutzt um Ausgangspins zu deaktivieren und Variablen zu löschen.

Weitere Funktionen, Variablen oder Konstanten in der Motorklasse können bei Bedarf ergänzt werden.

Schnittstellen PWM-Klasse

bool `init( Configuration::s_PWMConfig *thisPWMConfig_ )`

Initialisiert und aktiviert das PWM Modul 0 und die verwendeten PWM-Pins. Setzt die PWM Frequenz und konfiguriert die PWM für die Motoren.

bool `setChannelPWMRatio( unsigned char ratioOn )`

Setzt die aktuelle Geschwindigkeit in ein PWM Signal um, indem der errechnete Wert in das entsprechende Compare-Register geschrieben wird. Erwartet die Soll-Geschwindigkeit `ratioOn`.

bool `setChannelsEnabled( bool enabled )`

Aktiviert oder deaktiviert den jeweiligen PWM-Kanal.

void `cleanUp( void )`

Wird vom Destruktor genutzt um Ausgangspins zu deaktivieren und Variablen zu löschen.

Weitere Funktionen, Variablen oder Konstanten in der PWM-Klasse können bei Bedarf ergänzt werden.

Schnittstellen GPIOSensor-Klasse

void `init( Configuration::s_GPIOSensorConfig *thisGPIOSensorConfig_ )`

Initialisiert die entsprechenden Pins als Eingang, einschließlich Pull-Up Widerstand. Übergabeparameter ist ein Zeiger auf die entsprechende GPIOSensor-Konfiguration.

struct `s_GPIOSensorConfig`

Diese Struktur beinhaltet die Konfiguration der GPIO und ist in `configuration.h` implementiert. **unsigned char** `port` zum Übergeben des Ports für den GPIO-Pin (0 = Port PA; 1 = Port PB, etc.), **unsigned long** `pin` zum Übergeben des Pins am entsprechenden Port und **bool** `pullupEnabled` zum aktivieren oder deaktivieren des Pull-Up Widerstands.

bool `getValue( void )`

Gibt den aktuellen Wert des Sensors zurück. Rückgabewert **true** heißt, dass der Anschlusspin High-Level hat.

void `cleanUp( void )`

Wird vom Destruktor genutzt um Ausgangspins zu deaktivieren und Variablen zu

löschen.

Weitere Funktionen, Variablen oder Konstanten in der GPIOSensor-Klasse können bei Bedarf ergänzt werden.

Schnittstellen ADCSensor-Klasse

`bool init( Configuration::s_ADCSensorConfig *thisADCSensorConfig_, ADC *ADCController_ )`

Speichert für jedes Sensor-Objekt die durch den Zeiger im Übergabewert übergebenen Parameter von `thisADCSensorConfig_` in eigenen Variablen/Konstanten.

`struct s_ADCSensorConfig`

Diese Struktur beinhaltet die Konfiguration für die ADC Klasse und ist in `configuration.h` definiert. `unsigned long ADCChannelID` beinhaltet den Kanal des ADC, `bool useHWAverage` gibt an, ob die Hardware-Average Funktion des ADC-Modules verwendet werden soll, `unsigned char HWAverage` beinhaltet den Wert des HW-Averaging, `signed long zeroOffset` übergibt den aktuellen Offset falls bereits vorhanden, `bool useZeroOffset` aktiviert oder deaktiviert die Verwendung des Offset.

`long getIntegerValue( void )`

Gibt den Wert des jeweiligen ADC-Kanals/Sensor-Objekts zurück. Wenn ein Offset verwendet wird, so wird dieser vor der Rückgabe des Messwerts abgezogen.

`void setZeroOffset( bool active, signed long offset )`

Aktiviert oder deaktiviert die Verwendung eines Offset mit `active` und setzt den Offset auf den Übergabeparameter `offset`.

`signed long getZeroOffset( void )`

Gibt den Offset-Wert zurück.

`bool getZeroOffsetIsActive( void )`

Gibt zurück, ob ein Offset verwendet wird.

`void cleanUp( void )`

Wird vom Destruktor genutzt, um Ausgangspins zu deaktivieren und Variablen zu löschen.

Weitere Funktionen, Variablen oder Konstanten in der ADCSensor-Klasse können bei Bedarf ergänzt werden.

Schnittstellen ADC-Klasse

`bool init( void )`

Diese Funktion initialisiert ein oder mehrere ADC-Module und deren Eingangspins. Sie konfiguriert Sample-Sequenzer und setzt die Reihenfolge des Multiplexings der Eingänge, wenn eine ADC-Conversion stattfinden soll. Desweiteren aktiviert sie optionale ADC-Funktionen wie Hardware-Averaging, falls in der `configuration.cpp` vorgegeben.

`void clearIntFlag( void )`

Diese Funktion löscht den Interrupt flag, welcher verwendet werden kann um anzuzeigen, dass eine Conversion abgeschlossen wurde. Hierbei handelt es sich jedoch um

einen Interrupt der nicht an die NVIC (Nested Vectored Interrupt Table) weitergegeben wird!

`void prozessorTriggerConversion( void )`

Startet einen oder mehrere Sample-Sequenzer und damit ADC-Conversions gleichzeitig.

`bool IsConversionFinished( void )`

Gibt **true** zurück, wenn eine Conversion abgeschlossen wurde und **false**, wenn noch Conversions ablaufen.

`unsigned long getChannelValue( unsigned long channelID )`

Diese Funktion gibt den im Buffer gespeicherten Wert der ADC-Conversion, abhängig von der übergebenen channelID (0 = AIN0; 1 = AIN1, etc.), zurück. Vor der Rückgabe wird dieser Wert (12 bit ADC-Ergebnis) jedoch auf das Ergebnis eines 10 bit ADC zurückgerechnet, damit der Regler damit arbeiten kann.

`void ADCUpdateDataValues( void )`

Hier wird das gesamte FIFO (First in first out) Register der verwendeten Sample-Sequenzer ausgelesen und zwischengespeichert.

`void cleanUpADC( void )`

Wird vom Destruktor genutzt um Ausgangspins zu deaktivieren und Variablen zu löschen.

Weitere Funktionen, Variablen oder Konstanten in der ADC-Klasse können bei Bedarf ergänzt werden.

Schnittstellen Timer-Klasse

`bool initTimer( unsigned long frequency )`

Ruft `enableTimerBlockClock()` auf und initialisiert ein Timer-Modul. Wenn die Initialisierung erfolgreich war, werden die Interrupts aktiviert. Wenn Initialisierung nicht erfolgreich war, werden Interrupts deaktiviert. Übergabewert ist die Frequenz des Timers.

`void setIsTimerEnabled( bool enabled )`

Aktiviert oder deaktiviert den Timer.

`void setIsTimerInterruptEnabled( bool enabled )`

Aktiviert oder deaktiviert die Interrupts, die durch den Timer geworfen werden sollen.

`static void resetInterruptFlag( void )`

Setzt im TIMER-ICR Register das entsprechende Bit um den Interrupt Flag zu löschen.

`void enableTimerBlockClock( void )`

Aktiviert die Clock beim verwendeten Timer-Modul.

`void cleanUp( void )`

Wird vom Destruktor genutzt, um Ausgangspins zu deaktivieren und Variablen zu löschen.

Weitere Funktionen, Variablen oder Konstanten in der Timer-Klasse können bei Bedarf ergänzt werden.

Regelalgorithmus des TivSeg

Die vorgegebene `Segway-Klasse` wird anfangs verwendet, um das TivSeg™ sowie die Maximalwerte der Motoransteuerung zu initialisieren. Der Kern dieses Moduls steuert die Regelung des TivSegs™. Dazu wird die Funktion `Segway::timerFunktion` regelmäßig vom Objekt der `Timer-Klasse` aufgerufen. Anschließend werden die Sensoren aus den Objekten der `Sensor-Klassen` ausgelesen und durch den Regler verarbeitet. Durch die Werte der Lagesensoren und der Lenkregelung wird die Geschwindigkeit der einzelnen Räder berechnet und an die Objekte der `Motor-Klasse` übergeben. Für jede Peripherie erstellt sich der TivSeg™ Hilfsobjekte, welche sich aus den von Ihnen implementierten Klassen generieren sollen. Eine ausführliche Beschreibung der `Segway-Klasse` ist im Source-Code der Datei `Segway.cpp` enthalten. Des Weiteren wurde für diesen kommentierten Code die `TivSeg-Klassenbeschreibung.pdf` mit Hilfe von Doxygen erzeugt.

Kapitel 5

Zeitplanung und Meilensteine

Ihnen stehen insgesamt 7 Praktikumstermine zur Verfügung, während denen Sie folgende Teilschritte in 3er Gruppen mit der Betreuung eines Gruppenleiters bearbeiten müssen. Im Zeitplan in Abbildung 5.1 sehen Sie alle geforderten Meilensteine, die im Labortermin der jeweiligen Woche vorzulegen sind. Die einzelnen Meilensteine sind im Folgenden stichpunktartig aufgelistet:

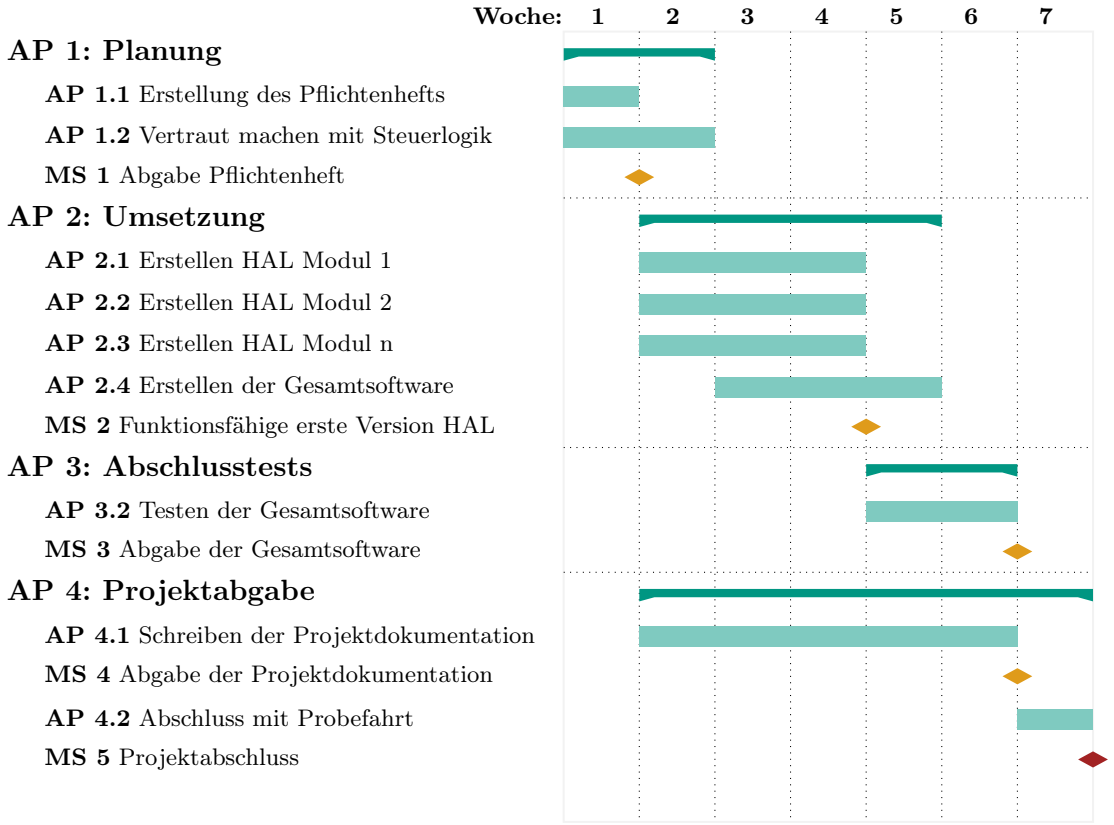


Abbildung 5.1: Gantt Diagramm des Projektpraktikums Informationstechnik

Wichtig: Die Abnahme der einzelnen Meilensteine erfolgt mit Ihrem Gruppenleiter (Tutor) anhand der unten aufgeführten Punkte. Sprechen Sie für eine Abnahme Ihren Gruppenleiter bitte rechtzeitig an.

5.1 Abgabe des Pflichtenhefts: (Ende Woche 1)

Das Pflichtenheft beschreibt in konkreter Form, wie Sie die Anforderungen des Auftraggebers zu lösen gedenken – das sogenannte wie und womit. Der Auftraggeber beschreibt vorher im Lastenheft möglichst präzise die Gesamtheit der Forderungen – was er entwickelt oder produziert haben möchte. Erst wenn der Auftraggeber das Pflichtenheft akzeptiert, sollte die eigentliche Umsetzungsarbeit beim Auftragsnehmer beginnen.

- **Ziel: Verständnis der Problem- und Aufgabenstellung sowie Identifizierung der Arbeitspakete und Erstellung eines Projektplans**
- Durcharbeiten der vorgegebenen Spezifikationen & Programmteile
- Knackpunkte identifizieren (nicht lösen)
- Aufgabenverteilung
  - ☐ Aufteilen des Gesamtprojekts in Teilaufgaben (insgesamt 6 hardwarenahe Module werden benötigt)
  - ☐ Faire Aufgaben- und Rollenverteilung (nicht nach Vorwissen) sowie Zuständigkeiten
  - ☐ Hinweis: Jeder sollte alle Module kennen
- Aufbau des Programms
  - ☐ Ergänzung des vorgegebenen UML-Klassendiagramms
  - ☐ Schnittstellen u. a. über Übergabe- und Rückgabeparameter
- Das Pflichtenheft als Liefergegenstand muss mindestens aus folgenden Punkten bestehen
  - ☐ Titelblatt
  - ☐ Abstract
  - ☐ Inhalts-, ggf. Abbildungs- und Tabellenverzeichnisse
  - ☐ Einführung
  - ☐ Analyse des Problems
  - ☐ Beschreibung der Schnittstellen mit Klassendiagramm
  - ☐ Beurteilung der Machbarkeit des Auftrags
  - ☐ Beschreibung der TivSeg™-Funktionsweise (Ablaufdiagramm)
  - ☐ Projektstrukturplan
  - ☐ Zeitplan (Gantt-Chart)
    - ∗ Aufteilen des Gesamtprojekts in Teilaufgaben
    - ∗ Planung der einzelnen Arbeitspakete
    - ∗ Festlegen von internen Deadlines und Fristen
    - ∗ Finden Sie den kritischen Pfad in ihrem Zeitplan
    - ∗ Projektorganisation / Zuständigkeiten (s. oben bei der Aufgabenverteilung)
  - ☐ Ablaufkontrolle (Wie man sicherstellt, dass der Zeitplan eingehalten wird.)

5.2 Abgabe eines funktionierenden HALs: (Ende Woche 4)

- **Ziel: Implementierung und Test des Hardware Abstraction Layers (HAL) anhand der Spezifikation und Planung. Dies beinhaltet noch keinen fehlerfreien Code, aber alle implementierten Schnittstellen.**
- Einhaltung der Programmierrichtlinien
- Verwendung der Codevorlage
- Gegenseitige Hilfestellung & Hilfestellung durch den Gruppenleiter
- Implementierung & Testen des HAL (siehe Abbildung 5.2)
  - ☐ Implementierung von einzelnen Modulen
  - ☐ Test der einzelnen Module mit dem TivSeg™-Simulator
  - ☐ Zusammenfügen der Module
  - ☐ Test des zusammengefügten Programms
  - ☐ Beginn wieder bei Punkt 1 bis zur Implementierung des vollständigen Programms
- Kommentierung der Funktionen und Testdokumentation (Doxygen).
- Die HAL muss mindestens die Implementierung der definierten Schnittstellen und Prototypen enthalten für:
  - ☐ Analog-Digital-Wandler (ADC)
  - ☐ Analog-Digital-Wandler Sensor (ADCSensor)
  - ☐ Puls-Weiten-Modulation (PWM)
  - ☐ Motor
  - ☐ Timer
  - ☐ GPIOSensor

5.3 Abgabe der Gesamtsoftware: (Ende Woche 6)

Für die Abgabe der TivSeg™-Steuerung soll die gesamte Software auf einem Simulator der TivSeg AG vorgeführt werden können.

- **Ziel: Implementierung und Test des Gesamtprogramms anhand der Spezifikation und Planung. Die Funktionsfähigkeit kann im Simulator nachgewiesen werden und der Code ist fehlerfrei.**
- Einhaltung der Programmierrichtlinien
- Gegenseitige Hilfestellung & Hilfestellung durch den Gruppenleiter

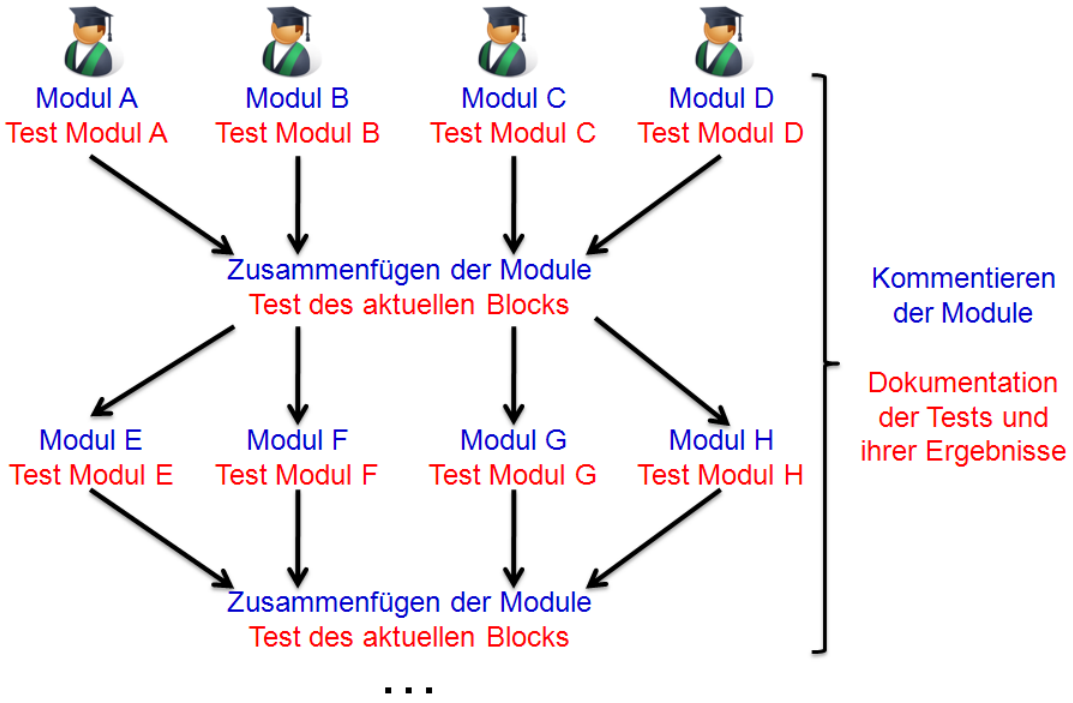


Abbildung 5.2: Beispiel: Implementierung und Test der Software-Module und der Gesamtlösung

- Implementierung & Testen der Gesamtsoftware (siehe Abbildung 5.2)
  - ☐ Zusammenfügen der HAL-Module
  - ☐ Test des zusammengefügten Programms
  - ☐ Beginn wieder bei Punkt 1 bis zur Implementierung des vollständigen Programms
- Kommentierung der Funktionen und Testdokumentation
- Testen der korrekten Lauffähigkeit des Programms anhand des TivSeg™-Simulators

5.4 Abgabe der Dokumentation: (Ende Woche 6)

Parallel zur Abgabe der Gesamtsoftware muss eine Abschlussdokumentation mit Betriebsanleitung abgegeben werden. Während der Abgabe kann der Gruppenleiter Fragen bezüglich bestimmter Abschnitte des Programms an jedes beliebige Mitglied aus dem Team stellen.

- **Ziel: Abgabe des vollständigen Projektdokumentation**
- Abgabe der Dokumentation umfasst mindestens:
  - ☐ Titelblatt
  - ☐ Abstract
  - ☐ Inhalts-, ggf. Abbildungs- und Tabellenverzeichnisse
  - ☐ Begriffserklärungen (Abkürzungen erklären)
  - ☐ Projektauftragsdokument (IST-Analyse)

- ☐ Aufgabenstellung (SOLL-Konzept)
- ☐ Anforderungen
- ☐ Projektziele (SOLL-IST-VERGLEICH)
- ☐ Projektstrukturplan mit Zeitplanung / Meilenstein (Projektmanagement)
- ☐ Beschreibung verwendeter nicht einfacher Funktionen und Algorithmen
- ☐ Beschreibung der Tests und ihrer Ergebnisse
- ☐ Ausblick / Folgeaktivitäten
- ☐ Fazit
- ☐ Anlagen / Anhang: (eigentlich Bestandteil der Projektakte)
  - Pflichtenheft
  - Kundeneinweisung
  - Abnahme- und Testprotokoll

Bei Abnahme aller Meilensteine durch Ihren Gruppenleiter (Tutor) haben Sie das Praktikum bestanden. In ILIAS können Sie den Stand Ihrer Abgaben einsehen.

Weiter müssen sie an mindestens fünf der sechs Praktikumstermine anwesend sein, es herscht Anwesenheitspflicht im Praktikum. Ausnahme ist der letzte Termin, die Probefahrt am siebten Termin ist freiwillig.

Wichtig: Sollten Sie an mehr als einem Termin durch Krankheit verhindert sein, reichen Sie bitte ein entsprechendes Attest bei den Betreuern des Praktikums ein. Die Termine der Meilensteine sind Ausschlussfristen. D. h., ILIAS wird die Abgabe zum angegebenen Zeitpunkt automatisch sperren. Laden Sie daher Ihre Dokumente vorzeitig hoch. Sollten Sie Probleme bei der Abgabe haben, melden Sie sich bitte frühzeitig beim Praktikumsbetreuer.

Eine nicht erfolgreiche Abgabe der obigen Meilensteine oder ein zu häufiges Fehlen führt zum Nichtbestehen des Praktikums. Ein Wiederholen ist erst im Folgejahr wieder möglich.

Kapitel 6

Einstieg in die hardwarenahe Programmierung

Im Folgenden soll Ihnen ein kleiner Überblick vermittelt werden, wie Sie das Tiva™ C Launchpad programmieren und verwenden können. Das Herz dieses Launschpads ist der TM4C123G Chip von Texas Instruments (siehe Abbildung 6.1). Er setzt sich im Wesentlichen aus einem ARM Cortex M4 Prozessor und der ihn umgebenden Peripherie bestehend aus Analog Digital Convertern (ADCs), General Purpose Input Outputs (GPIOs), Timern u.v.m. zusammen. Diese Peripherie kann durch Programmierung der entsprechenden Register konfiguriert und verwendet werden.

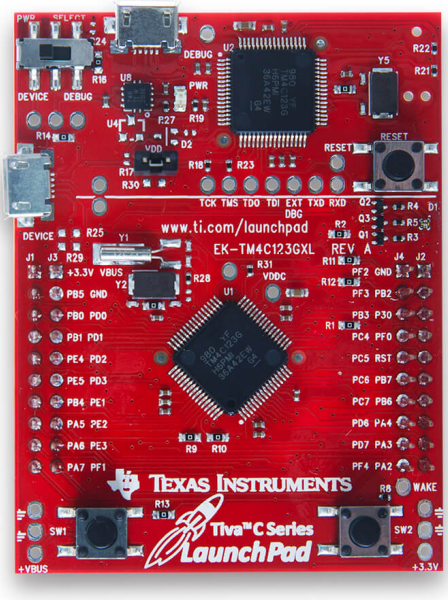


Abbildung 6.1: Launchpad Tiva™ C mit TM4C123G

Um einem Mikrocontroller eine spezielle Aufgabe zuzuweisen, muss er programmiert werden. Dies geschieht, indem das geschriebene Programm durch einen Compiler in Maschinensprache übersetzt und in den (meistens Flash-) Speicher des Mikrocontrollers geschrieben wird. Diesen Vorgang nennt man „Flashen“. Schaltet man den Mikrocontroller an, wird das Programm aus dem Programmspeicher direkt ausgeführt. Um eine ständige Funktionsweise zu ermöglichen und undefinierte Zustände zu vermeiden, wird das Programm immer wieder per Endlosschleife aufgerufen. Wie große PCs besitzt ein Mikrocontroller die Funktion des Neustartens. Die dazugehörige Taste ist auf dem Launchpad mit „Reset“ beschriftet. Im Folgenden werden Sie einige Register in ihrem TM4C123G direkt manipulieren um die auf dem Tiva™ C Launchpad angeschlossenen LEDs zu betätigen. Hierzu verwenden Sie die bereits genannte IDE Code Composer Studio.

Code Composer Studio 7 ist eine von Texas Instruments bereitgestellte Entwicklungsumgebung (IDE) für deren Prozessoren. Sie bietet eine große Auswahl an verschiedenen Treibern, Debug-Funktionen, Entwickler-Tools und vieles mehr.

Zur Bearbeitung des Praktikums mit Code Composer Studio (CCS) können Sie eine der folgenden zwei Alternativen wählen:

1. Auf der Webseite des Herstellers Texas Instruments können Sie Code Composer Studio unter `http://processors.wiki.ti.com/index.php/Download_CCS` kostenlos herunterladen, eine Registrierung wird hier nicht benötigt. Laden Sie CCS herunter und folgen Sie den Anweisungen der Installation.

2. Um auch mit eventuell nicht unterstützten Rechnern arbeiten zu können, wird zusätzlich eine virtuelle Maschine angeboten, die auch für Nutzer anderer Betriebssysteme die Möglichkeit bereitstellt das Code Composer Studio in einer virtuellen Windowsumgebung zu nutzen. Für nähere Informationen hierzu und für den Link zum Download nehmen Sie bitte Kontakt mit dem Betreuer auf.

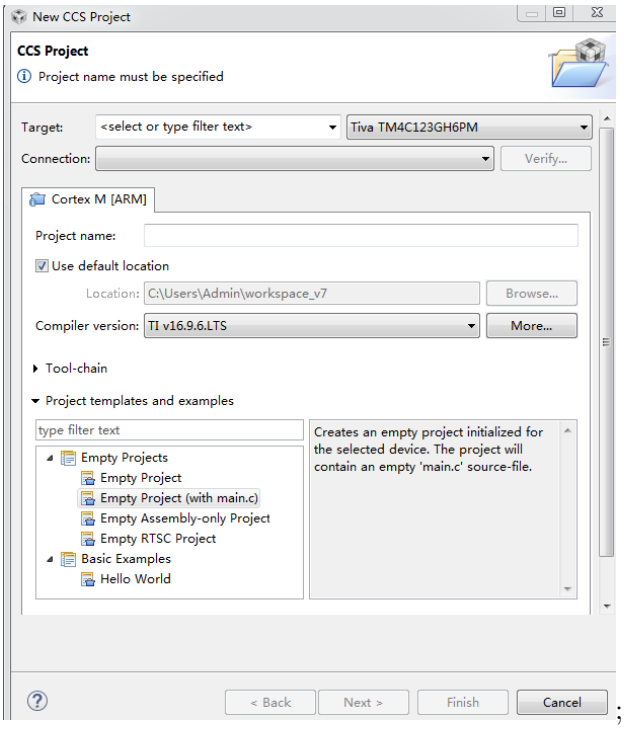


Abbildung 6.2: CCS-Projekt erstellen

Eine gute Einführung in die IDE CCS, sowie Ihr Launchpad stellt zum Beispiel der Tiva™ C Workshop TivaLab (`http://software-dl.ti.com/trainingTTO/trainingTTO_public_sw/GSW-TM4C123G-LaunchP`
`TM4C123G_LaunchPad_Workshop_Workbook.pdf`) dar.

6.1 Erstellen eines CCS-Projekts

Öffnen Sie zunächst Code Composer Studio auf ihrem Rechner und installieren Sie, falls noch nicht geschehen, die USB-Treiber für ihr Launchpad, indem Sie dieses mit dem Computer verbinden und die automatische Treiberinstallation abschließen. Erstellen Sie anschließend über *File > new > CCSProject* ein neues Projekt. Wählen sie als „Target“ ihren Mikrocontroller „TivaTM4C123GH6PM“, als „Connection“ das „Stellaris In-Circuit Debug Interface“ und wählen sie die Compiler Version „TI v5.2.7“. Geben Sie Ihrem Projekt einen Namen und wählen Sie als Template „Empty Project (with main.c)“. Dann haben Sie das Projekt erstellt. (siehe Abbildung 6.2)

6.2 Beispielprogramm: LED-Licht

Ersetzen Sie den Inhalt der *main.c* durch den unten in Quelltext 6.1 angegebenen C-Code.

Quelltext 6.1: Lauffähiges LED Programm.

```
/*
 * main.c
 */
#define SYSCTL_RUN_CLOCK_GATING_CONTROL_REGISTER    0x400FE608
#define GPIO_PORTF_DIR_REGISTER             0x40025400
#define GPIO_PORTF_DIGITAL_ENABLE_REGISTER        0x4002551C
#define GPIO_PORTF_DATA_REGISTER            0x400253FC

#define LED_RED     0x02
#define LED_BLUE    0x04
#define LED_GREEN   0x08

void main(void) {
  // enable System-clock to GPIO Port F
    *(volatile unsigned int *)SYSCTL_RUN_CLOCK_GATING_CONTROL_REGISTER |=
      (1u << 5);
  // Set Port F LED Pins (1, 2, 3) as outputs
  *(volatile unsigned int *)GPIO_PORTF_DIR_REGISTER |=
      (LED_RED | LED_BLUE | LED_GREEN);
  // Set Pins 1,2,3 digital Enable on Port F
  *(volatile unsigned int *)GPIO_PORTF_DIGITAL_ENABLE_REGISTER |=
      (LED_RED | LED_BLUE | LED_GREEN);
  // Clear Port F LED Pins
  *(volatile unsigned int *)GPIO_PORTF_DATA_REGISTER &=
      ~(LED_RED | LED_BLUE | LED_GREEN);
  // Set Pins High (LEDs on)
  *(volatile unsigned int *)GPIO_PORTF_DATA_REGISTER |=
      (LED_RED | LED_BLUE | LED_GREEN);

   while(42){
   }
}
```

Dieses Programm stellt eine maschinennahe Programmierung zur Ansteuerung der Onboard-RGB-LED dar.

Es werden über Zeiger und Speicherzugriffe die Register des Mikrocontrollers manipuliert. Die Angaben der Register geschehen in Hexadezimalzahlen. Hexadezimalzahlen werden mit einem *0x* markiert (alternativ kann man auch Binärzahlen, markiert durch *0b*, benutzen). In Dokumentationen sind praktisch immer Register mit Hex-Zahlen angegeben. So finden Sie die GPIO_PORTF_DIR_REGISTER-Adresse als Summe der PORT_F Basisadresse und dem DIR-Offset (in der Dokumentation des Mikrocontrollers auf der Seite 663). Die ausführliche Dokumentation des Mikrocontrollers erhalten Sie direkt auf der Internetseite von Texas Instruments oder in der Plattform ILIAS.

Um ein Register zu manipulieren, müssen Sie einen Zeiger/Pointer erstellen. Dies geschieht in vorliegendem Beispiel durch `(volatile unsigned int *)` gefolgt von der Registeradresse (Hier, zur Übersichtlichkeit, durch sogenannte „defines“ ersetzt). Der * am Beginn der Zeile 18 dereferenziert diesen Zeiger, um direkt auf den Registerinhalt zugreifen zu können. Durch den Operator |= wird die nachfolgende Adresse bitweise mit dem Registerinhalt ver-„ODER“t. Dies verwendet man, um einzelne Bits zu setzen, und nicht bei jedem Speicherzugriff das komplette Register neu zu beschreiben (Achtung Fehlerquelle!). Sollen einzelne Bits gelöscht werden, so kann dies ebenfalls durch eine Bitoperation geschehen (Beispiel im Code Zeile 24 und 25).

Die Onboard-RGB-LED des Launchpads ist an die Pins PF1, PF2 und PF3 angeschlossen. Um den Port F nutzen zu können, muss zuerst das Clock-Signal für diese Peripherie aktiviert werden (Siehe Zeilen 15 und 16) (Siehe hierzu auch Datenblatt Seiten 656 und 340). Die Pins 1, 2 und 3 werden durch die entsprechenden Bits in den Registern dargestellt. So muss eine 8 (Binär 0b1000, die 1 ist an der 3. Stelle) in die aufgeführten Register geschrieben werden, um die grüne LED, welche an PF3 angeschlossen ist, zu konfigurieren und einzuschalten. In obigem Code sind alle 3 LEDs gleichzeitig konfiguriert und eingeschaltet.

Um herauszufinden, welche Register für weitere Aufgaben verwendet werden müssen, ist es notwendig, mit dem Datenblatt des TM4C123G zu arbeiten.

Spielen Sie ein wenig mit dem Programm herum, um sich mit den Befehlen zur Registermanipulation vertraut zu machen und lesen sie insbesondere die Anleitung zur Konfiguration von GPIOs auf Seite 656 des Datenblatts.

6.3 Flashen und Debuggen

6.3.1 Flashen

Nun sollen Sie lernen, wie man den obigen Programmcode auf den Mikrocontroller lädt. Schließen Sie zunächst den Mikrocontroller über das mitgelieferte USB-Kabel an den PC an. Am Mikrocontroller muss der Mini-USB-Port mit der Aufschrift Debug verwendet werden, der Schalter ist ebenfalls auf Debug zu stellen. Durch Klicken des -Symbols wird das Programm kompiliert, auf den Mikrocontroller geladen und eine Debug-Session gestartet. Sie können über das -Symbol auf der Menüleiste von CCS das Programm starten und mithilfe des -Symbols das Programm anhalten. Wenn Sie auf das -Symbol klicken, gelangen Sie zurück zum Editiermodus. Das Programm ist dann auf den Mikrocontroller geflasht und läuft autonom. Zum Flashen kann auch UniFlash benutzt werden. Dieses Programm bietet unter anderem die Möglichkeit, ELF-Dateien auf den Mikrocontroller zu laden, welche zum Beispiel von dem TivSeg™-Simulator erzeugt werden. Sie können das Programm unter `http://processors.wiki.ti.com/index.php/Category:CCS_UniFlash` kostenlos herunterladen.

6.3.2 Debuggen

Der Debug-Modus hilft Ihnen dabei, zu verstehen, was Ihr Code auf dem Mikrocontroller bewirkt. So können Sie Fehler im Code identifizieren und bereinigen.

Der Debug-Modus erscheint nach Klicken des -Symbols automatisch, sofern der Code kompiliert und auf den Mikrocontroller geladen werden kann. Es sollte sich automatisch ein Fenster mit den Tabs Variables, Expressions und Registers öffnen. Falls dies nicht erfolgt, können Sie im Menü View einen entsprechenden Punkt auswählen.

Der Tab Variables kann benutzt werden, um die Werte von lokalen Variablen anzuzeigen.

Unter Expressions können Sie durch Klicken auf Add new expression und anschließender Eingabe eines definierten Variablennamens den aktuellen Wert und die Adresse einer Variable einsehen. Alternativ können Sie einen Ausdruck (z.B. 2+4, kann aber auch Variablen enthalten) oder eine Adresse eingeben und das Ergebnis einsehen. Durch Klicken des -Symbols wird die automatische Aktualisierung der Werte aktiviert (siehe Abbildung 6.3).

Der Tab Registers erlaubt es Ihnen, die Inhalte der wichtigsten Register des Mikrocontrollers einzusehen, sofern die entsprechenden Komponenten aktiviert wurden.

Außerdem können Sie Breakpoints setzen, indem Sie das -Symbol klicken oder im Quellcode-Fenster doppelt auf die Zeilennummerieung klicken. Beim Erreichen eines Breakpoints wird die Ausführung des Programms unterbrochen. Es kann so z.B. überprüft werden, ob Variablen nach Erreichen einer bestimmten Stelle im Code die erwarteten Werte haben.

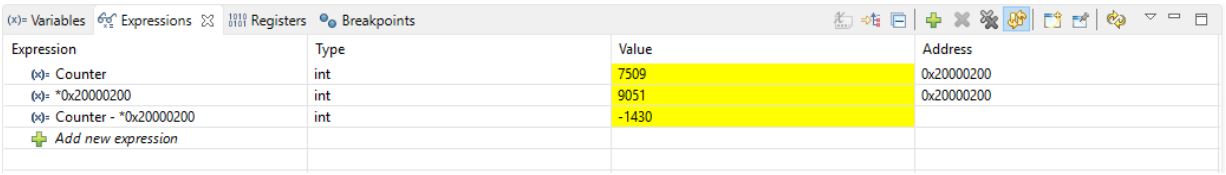


Abbildung 6.3: Expressions-Tab im Debug-Modus

6.3.3 Wichtige Hinweise

- Durch unberechtigte Speicherzugriffe oder undefiniertes Verhalten ist es möglich, dass Ihr Launchpad intern den Zugriff sperrt. Sollte dies der Fall sein, so können sie erst nach einem erfolgreichen Entsperrvorgang weitere Programme flashen. Eine erfolgreiche Entsperrung ist mit dem Tool LM Flash Programmer von Texas Instruments möglich, erhältlich unter `http://www.ti.com/tool/lmflashprogrammer`. Öffnen Sie hierzu den LM Flash Programmer und wählen Sie unter „Other Utilities“ den „Debug Port Unlock“ mit „Fury, DustDevil, TM4C...“ und führen sie „Unlock“ aus.
- Für jede Peripherie, die Sie verwenden, müssen Sie die Clock aktivieren. Wenn Sie versuchen, die Register einer Peripherie zu beschreiben, welche nicht aktiviert ist, erhalten Sie Fehlermeldungen.
- Ihnen ist der Gebrauch sogennannter APIs (Advanced Programming Interface) wie der TIVAWare nicht gestattet, da Sie selbst die Register manipulieren sollen.

6.4 Aufgabe 1: Umgang mit Datenblättern

Ein Datenblatt liefert Entwicklern die nötigen Informationen, um mit einem Produkt zu arbeiten. So enthält das Datenblatt des TM4C123G funktionale Beschreibungen der verschiedenen Komponenten, Registeradressen, Anweisungen zum Umgang mit den Registern und vieles mehr. Benutzen Sie das Inhaltsverzeichnis, um Informationen zu finden, die für Ihre Aufgabe relevant sind.

Öffnen Sie das Datenblatt des TM4C123G und recherchieren Sie die Basisadressen für ADC, PWM, GPIO Port F und GPIO Port E und geben Sie diese in Tabelle 6.1 an. Geben Sie zu jedem Peripheriegerät die Offset-Adresse für das Run-Mode-Clock-Gating-Control-Register (RCGC) an. Berechnen Sie anschließend die Gesamtadresse für die jeweiligen RCGC-Register. Diese ergibt sich als Summe von Basisadresse und Offset.

Peripherie	Basisadresse des Geräts	Offset für RCGC	Gesamtadresse
ADC			
PWM			
GPIO Port F			
GPIO Port E			

Tabelle 6.1: Register-Adressberechnung

6.5 Aufgabe 2: Blinklicht und Schalter

In dieser Aufgabe sollen Sie den unten in Quelltext 6.2 angegebenen C-Code so vervollständigen, dass sich die RGB-LED des LaunchPads wie folgt verhält: Wenn kein Schalter gedrückt wird, leuchtet die LED blau. Wenn der Schalter SW1 gehalten wird, blinkt die LED abwechselnd blau und grün. Beim Halten von Schalter SW2 blinkt die LED blau und rot.

Implementieren Sie dazu die fehlenden Funktionen im angegebenen Quelltext. Füllen Sie die Funktion `void PortF_Init(void)` aus, sodass die Pins an Port F korrekt arbeiten. Hierzu können Sie sich an Kapitel 10.3 im Datenblatt orientieren (Hinweis: Es werden nicht alle beschriebenen Punkte benötigt, alle benötigten Register sind bereits im Code definiert). Füllen Sie auch die `while(1)`-Schleife in der `main()`-Funktion aus. Hier werden die entsprechenden Pins gelesen und gesetzt, sodass sich die Farbe der LED wie beschrieben ändert. Zum Erreichen des Blinkens können Sie die bereits vorhandene `Delay()`-Funktion verwenden.

Quelltext 6.2: Unvollständiges Programm: Blinkende RGB-LED

```
1  //define easy to read names for registers
2  //#define GPIO_PORTF_DATA_R      (*((volatile unsigned long *)0x40025000))
3  #define GPIO_PORTF_DATA_R       (*((volatile unsigned long *)0x400253FC))
4  #define GPIO_PORTF_DIR_R        (*((volatile unsigned long *)0x40025400))
5  #define GPIO_PORTF_AFSEL_R      (*((volatile unsigned long *)0x40025420))
6  #define GPIO_PORTF_PUR_R        (*((volatile unsigned long *)0x40025510))
7  #define GPIO_PORTF_DEN_R        (*((volatile unsigned long *)0x4002551C))
8  #define GPIO_PORTF_LOCK_R       (*((volatile unsigned long *)0x40025520))
9  #define GPIO_PORTF_CR_R         (*((volatile unsigned long *)0x40025524))
10 #define GPIO_PORTF_AMSEL_R      (*((volatile unsigned long *)0x40025528))
11 #define GPIO_PORTF_PCTL_R       (*((volatile unsigned long *)0x4002552C))
12 #define SYSCTL_RCGC2_R          (*((volatile unsigned long *)0x400FE108))
13
14 // Global Variables
15 unsigned long SW1, SW2;     // input from PF0 and PF4
16
17 // Function Prototypes
```

```
18 void PortF_Init(void);
19 void Delay(void);
20
21 // Subroutines Section
22 int main(void){
23   PortF_Init();                  // Call initialization of port PF0-PF4
24   while(1){
25
26
27   }
28 }
29
30 // Subroutine to initialize port F pins for input and output
31 // PF4 and PF0 are input SW1 and SW2 respectively
32 // PF3,PF2,PF1 are outputs to the LED
33
34
35 void PortF_Init(void){// GPIO Initialization and configuration
36   // See datasheet chapter 10.3 on page 656
37
38 }
39
40 void Delay(void){
41   unsigned long volatile time;
42   time = 500000;
43   while(time){
44     time--;
45   }
46 }
```

6.6 Aufgabe 3: Lichtstärke der LED ändern

In dieser Aufgabe sollen Sie den Code so ergänzen, dass die Lichtstärke der Onboard-LED durch ein Potentiometer gesteuert werden kann. Dazu müssen die Module GPIO, PWM und ADC verwendet werden.

Ihre Aufgabe ist es, die Funktionen `PWM_init()` und `ADC_init()` zu ergänzen. Die Funktion `PWM_init()` initialisiert das PWM-Modul 0 und den GPIO Port B. Eine Anleitung zur Initialisierung des PWM-Moduls finden Sie auf Seite 1239 des Datenblatts. Die Funktion `ADC_init()` initialisiert das ADC-Modul 0, es soll der Sample-Sequencer 3 verwendet werden. Eine Anleitung dazu finden Sie auf Seite 817 des Datenblatts. Verwenden Sie den im Quelltext 6.3 angegebenen C-Code. Zum setzen, löschen und lesen von Bits können Sie die zu Beginn definierten Bitmakros verwenden.

Legen Sie zusätzlich die Dateien ADC.h, PWM.h und GPIOSensor.h an und definieren Sie dort alle benötigten Registeradressen.

Quelltext 6.3: Unvollständiges Programm: Lichtstärke der LED ändern

```
1  #include <stdio.h>
2  #include "PWM.h"
3  #include "ADC.h"
4  #include "GPIOSensor.h"
5
6  // bitmacros
7
8  /* set bit */
9  #define SET_BIT(var, bit) ((var) |= (1u << (bit)))
10
11 /* set bits */
12 #define SET_BITS(var, bits) ((var) |= (bits))
13
14 /* clear bit */
15 #define CLEAR_BIT(var, bit) ((var) &= (unsigned)~(1u << (bit)))
16
17 /* clear bits */
18 #define CLEAR_BITS(var, bits) ((var) &= (unsigned)~(bits))
19
20 /* bit is set/clear */
21 #define BIT_IS_SET(var, bit) ((var) & (1u << (bit)))
22 #define BIT_IS_CLEAR(var, bit) !BIT_IS_SET(var, bit)
23
24 /* Value behind an address */
25 #define HWACC(x) ((*((volatile unsigned int *)(x))))
26
27 // Declaration of Functions
28 void PWM_init(void);
29 void ADC_init(void);
30 int ADC_IsConversionFinished(void);
31
32
```

```
33 // main.c
34 int main(void)
35  {
36     PWM_init();
37     ADC_init();
38     //volatile int* data ;
39 
40      while(1){
41         SET_BIT(HWACC(ADC0_BASE_ISC), 3); //clear the interrupt of ss3
42         while(ADC_IsConversionFinished()==0)
43             continue;
44         if(BIT_IS_CLEAR(HWACC(ADC0_BASE_SSFSTAT3),8))
45         {// An interrupt occurs when the conversion finishes
46             PWM0_COMPARE_A = HWACC(ADC0_BASE_SSFIFO3); //read the value
47         }
48 
49     }
50   return 0;
51 }
52 
53 
54 void PWM_init(void) {
55   // GPIO Initialization
56 
57   // PWM Initialization
58 
59         return ;
60 }
61 
62 void ADC_init() {
63   // Module Initialization
64 
65 
66   // Sample Sequencer Configuration
67 
68 
69     return ;
70 }
71 int ADC_IsConversionFinished()
72 {
73     if (BIT_IS_SET(HWACC(ADC0_BASE_RIS),3))
74             return 1;
75         else
76             return 0;
77 }
```

Kapitel 7

Weitere Labore

Im weiteren Verlauf Ihres Studiums bietet ihnen die TivSeg AG in Zusammenarbeit mit ITIV weitere Möglichkeiten, den TivSeg™ zu optimieren.

Die weiteren Labore bestehen aus drei Themenbereichen des Systems-Engineering: Platinenentwurf, integrierte Schaltungen und eingebettete Software. Diese werden am Beispiel eines TivSegs™ für Sie erlernbar aufbereitet. Dadurch wird Ihnen eine eigenverantwortliche, fachliche Arbeitsweise näher gebracht. Schlüsselqualifikationen, wie zum Beispiel Teamfähigkeit, Projekt- und Zeitmanagement, sollen selbstverständlicher Teil der Arbeitsweise werden.

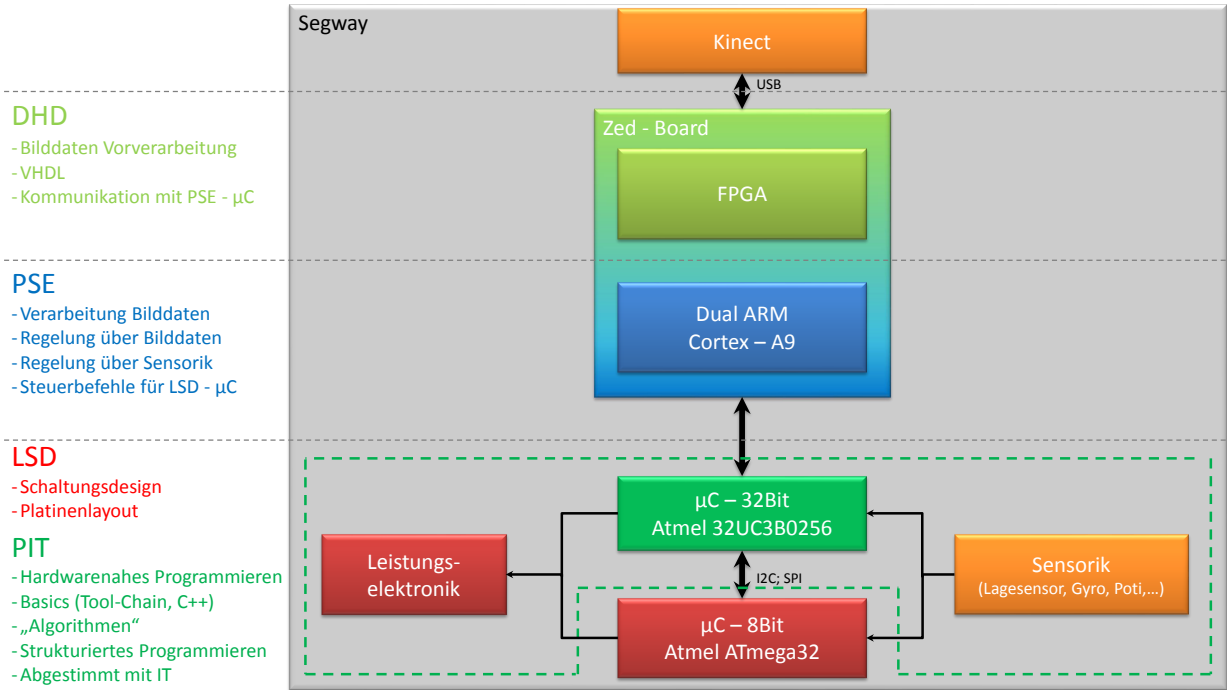


Abbildung 7.1: Übersicht der TivSeg™ Labore

LSD Labor Schaltungsdesign: Bei der Lehrveranstaltung handelt es sich um ein dreiwöchiges Blockpraktikum. Ziel des Praktikums ist die Entwicklung und der Aufbau der gesamten Elektronik zum Betrieb eines selbstbalancierenden einachsigen Beförderungsmittels (TivSeg™). Zunächst werden in einem vorlesungsartigen Teil häufig benötigte Grundschaltungen besprochen. Anschließend erstellen mehrere Zweierteams einzelne Schaltungskomponenten, welche am Ende zum Gesamtsystem zusammengesetzt und getestet werden.

PSE Praktikum Software Engineering: Im Labor entwerfen und implementieren die Studenten Software zur Steuerung eines autonom fahrenden TivSeg™. Dies umfasst die Verarbei-

tung von Videodaten und Tiefeninformationen zur Objekt- und Hinderniserkennung und die darauf aufbauende Ansteuerung des Fahrzeugs zur Objektverfolgung und Hindernisvermeidung. Die Aufgabe wird projektorientiert selbstständig in Teams von 3-4 Studenten bearbeitet. Kommerzielle Entwicklungswerkzeuge für computergestützte Softwaretechnik (CASE Tools) begleiten den Entwicklungsprozess.

DHD Digitales Hardware Design: Innerhalb eines projektbasierten Labors werden die Studenten in Teams an den Entwurf komplexer Hardware/Software Systeme herangeführt. Im Zentrum steht der Entwurf von Algorithmen für FPGAs, welche die Verarbeitung von Kameradaten zur Steuerung eines autonom fahrenden TivSegs™übernehmen.

Kapitel 8

Haftungsausschlusserklärung

für die freiwillige Teilnahme an einer TivSeg™-Testfahrt
gegenüber dem Karlsruher Institut für Technologie (KIT)

von

Vorname	Nachname	Geburtsdatum

Straße	PLZ	Ort

E-mail	Matrikelnummer

Bitte in Druckbuchstaben ausgefüllt und unterzeichnet zur Testfahrt mitbringen!
Vielen Dank!

Teilnahmebedingungen: Bitte prüfen Sie die Voraussetzungen und Bedingungen für Ihre Teilnahme an der TivSeg™-Testfahrt. Sie nehmen an der TivSeg™-Testfahrt unter Beachtung und Akzeptanz der nachfolgend aufgeführten Punkte teil:

- Die Teilnahme an der Testfahrt erfolgt auf freiwilliger Basis und auf Ihren ausdrücklichen Wunsch. Sie ist weder verpflichtender Bestandteil des Praktikums noch Voraussetzung für die schriftliche Prüfung.
- Sie lassen sich vor dem Start durch unsere Gruppenleiter in die Handhabung und das Fahren des TivSegs™ einweisen. Erst nach Ihrer Teilnahme an diesem Einweisungs- und Fahrtraining dürfen Sie an der TivSeg™-Testfahrt teilnehmen.
- Es gilt die Straßenverkehrs-Ordnung (StVO). Die Teilnahme am Straßenverkehr erfordert ständige Vorsicht und gegenseitige Rücksicht. Jeder Verkehrsteilnehmer hat sich so zu verhalten, dass kein anderer geschädigt, gefährdet oder mehr, als nach den Umständen unvermeidbar, behindert oder belästigt wird.
- Sie sind mindestens 18 Jahre alt und im Besitz einer gültigen Fahrerlaubnis (Mofa-Führerschein; ist im PKW Führerschein Klasse B enthalten).
- Sie wiegen mindestens 45 Kilo und sind nicht schwerer als 110 Kilo. Sie sind ohne fremde Hilfe in der Lage zu stehen und Stufen zu steigen. Sie leiden nicht an Epilepsie, Thrombosen,

Herz- oder Kreislaufkrankheiten, Folgen nach Schlaganfall, Gehbehinderungen, Gleichgewichtsstörungen oder vergleichbaren Handicaps.

- Schwangeren Frauen sollten an der Testfahrt nicht teilnehmen.
- Sie haben vor der Testfahrt weder Alkohol oder / und andere Betäubungsmittel konsumiert. Gleiches gilt während der Fahrt.
- Während der Fahrt mit dem TivSeg™ ist das Telefonieren, Fotografieren oder das Rauchen aus Sicherheitsgründen nicht gestattet.
- Sie folgen Ihrem Gruppenleiter indem seinerseits vorgegebenen Tempo sowie Richtung. Sie beachten und befolgen den Anweisungen unserer Gruppenleiter und halten stets einen ausreichenden Sicherheitsabstand zu anderen Praktikums- bzw. Verkehrsteilnehmern ein.
- Sie fahren ausschließlich in den von dem Gruppenleiter vorgegebenen abgesicherten Bereich. Insoweit dieser Bereich dennoch von anderen Verkehrsteilnehmern (PKW, Fahrräder, Fußgänger etc.) genutzt wird, unterbrechen Sie die Fahrt, bis der Bereich wieder frei ist. Eine Nutzung außerhalb des vorgegebenen Bereichs ist Ihnen untersagt.
- Sie fahren vorsichtig und gehen mit dem TivSeg™ sorgsam um, dies in Ihrem eigenen und unserem Interesse.
- Bei rot blinkenden Lichtern oder bei Vibrationen der Plattform muss der Fahrer den TivSeg™ sofort bremsen und absteigen, es kann zu einer Sicherheitsabschaltung kommen.

Das Karlsruher Institut für Technologie (KIT) übernimmt keine Haftung für Sach- und Vermögensschäden, die insbesondere durch die Nutzung der TivSegs™ oder durch einen anderen Teilnehmer oder einen Dritten entstehen, sofern die Schäden nicht auf Vorsatz oder grober Fahrlässigkeit seitens des KIT beruhen. Die Haftungsbeschränkung auf vorsätzlichem und grob fahrlässigem Verhalten seitens des KIT gilt nicht für Ansprüche aus der Verletzung des Lebens, des Körpers oder der Gesundheit.

Sie haften für sämtliche Personen-, Sach- und Vermögensschäden, die Sie dem KIT, dessen Mitgliedern und /oder Angehörige oder Dritten verursachen. Sie stellen das KIT von Ansprüchen Dritter frei, es sei denn die Haftung beruht auf vorsätzlichem oder grob fahrlässigem Verhalten des KIT.

Die Teilnahme an der TivSeg™-Testfahrt ist Ihnen nur dann gestattet, wenn Sie diese Haftungsausschlusserklärung zur Kenntnis genommen und unterschrieben haben.

☐ Der Inhalt des Haftungsausschluss wurde von mir zur Kenntnis genommen und inhaltlich vollständig akzeptiert.

☐ Ich trage einen eigenen oder gestellten Helm.

Ort, Datum Unterschrift

Anhang A

Pinbelegung des TM4C123G

Pin	Verwendung	Pin	Verwendung
PB05	Not Connected (NC)	PB2	NC
PB0	UART: RX	PE0	Rechter Onboard-Button
PB1	UART: TX	RST	Reset Button
PE4	Strommessung linker Motor (AIN9)	PB7	SPI-MOSI
PE5	Strommessung rechter Motor (AIN8)	PB6	SPI-MISO
PB4	NC	PA4	NC
PA5	NC	PA3	NC
PA6	I2C-Bus: SCL	PA2	Fußschalter
PA7	I2C-Bus: SDA	PF2	Blaue Onboard LED
PD0	Drehratensensor 2 (AIN7)	PF3	Rote Onboard LED
PD1	Drehratensensor 1 (AIN6)	PB3	NC
PD2	Spannungsteiler Batterie (AIN5)	PC4	Linker Motor PWM
PD3	Potenziometer Lenkausschlag (AIN4)	PC5	Rechter Motor PWM
PE1	Accelerometer X (AIN2)	PC6	Rechter Motor Richtungspin
PE2	Gyrometer Referenzspannung (AIN1)	PC7	Linker Motor Richtungspin
PE3	Gyrometer Y (AIN0)	PD6	Motoren Enable-Pin
PF1	Rote Onboard LED	PD7	NC
PF4	Linker Onboard-Button		

Tabelle A.1: Pinbelegung des TM4C123G

Anhang B

Schaltplan der Steuerplatine

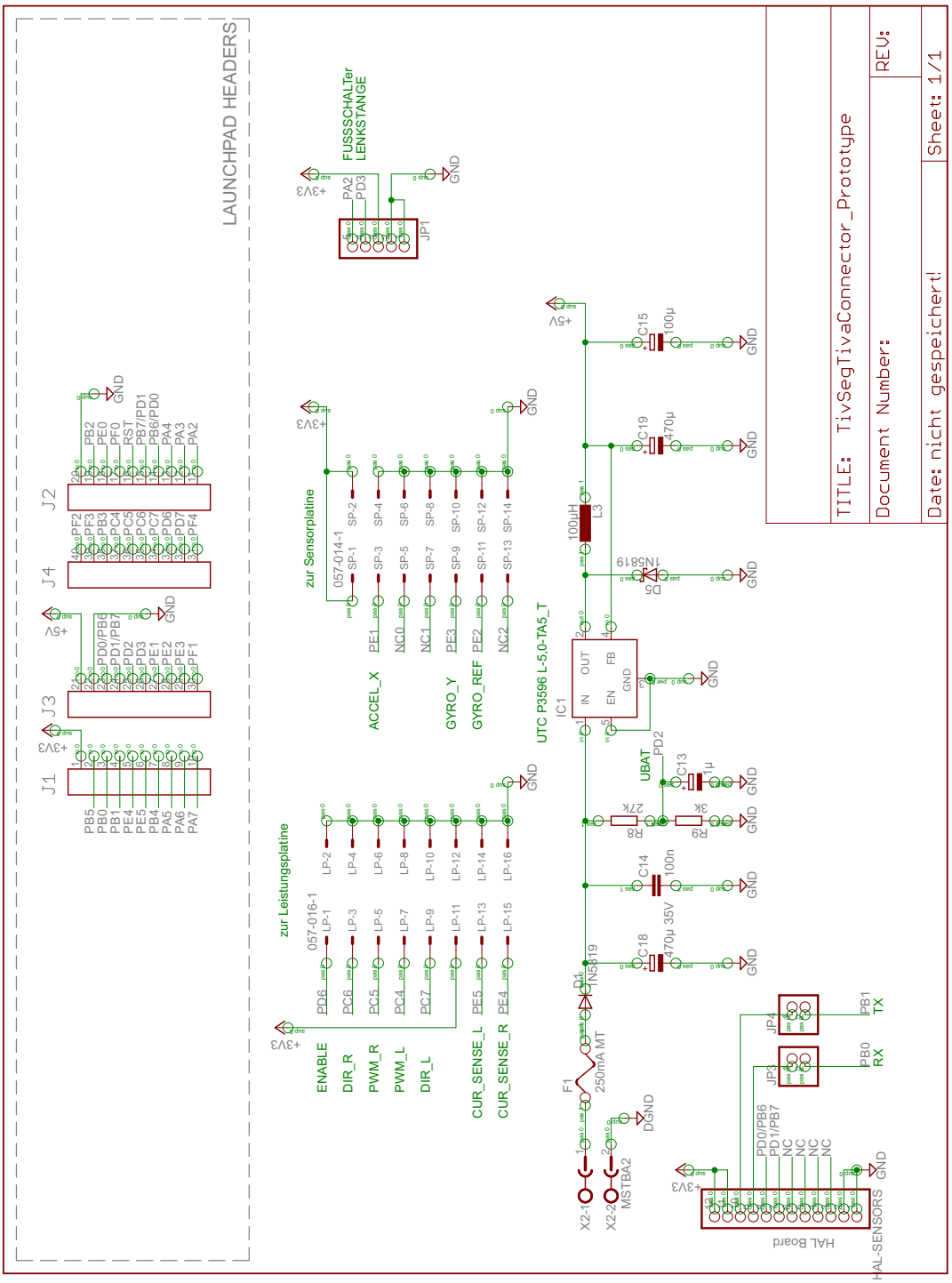


Abbildung B.1: Schaltplan der Steuerelektronik

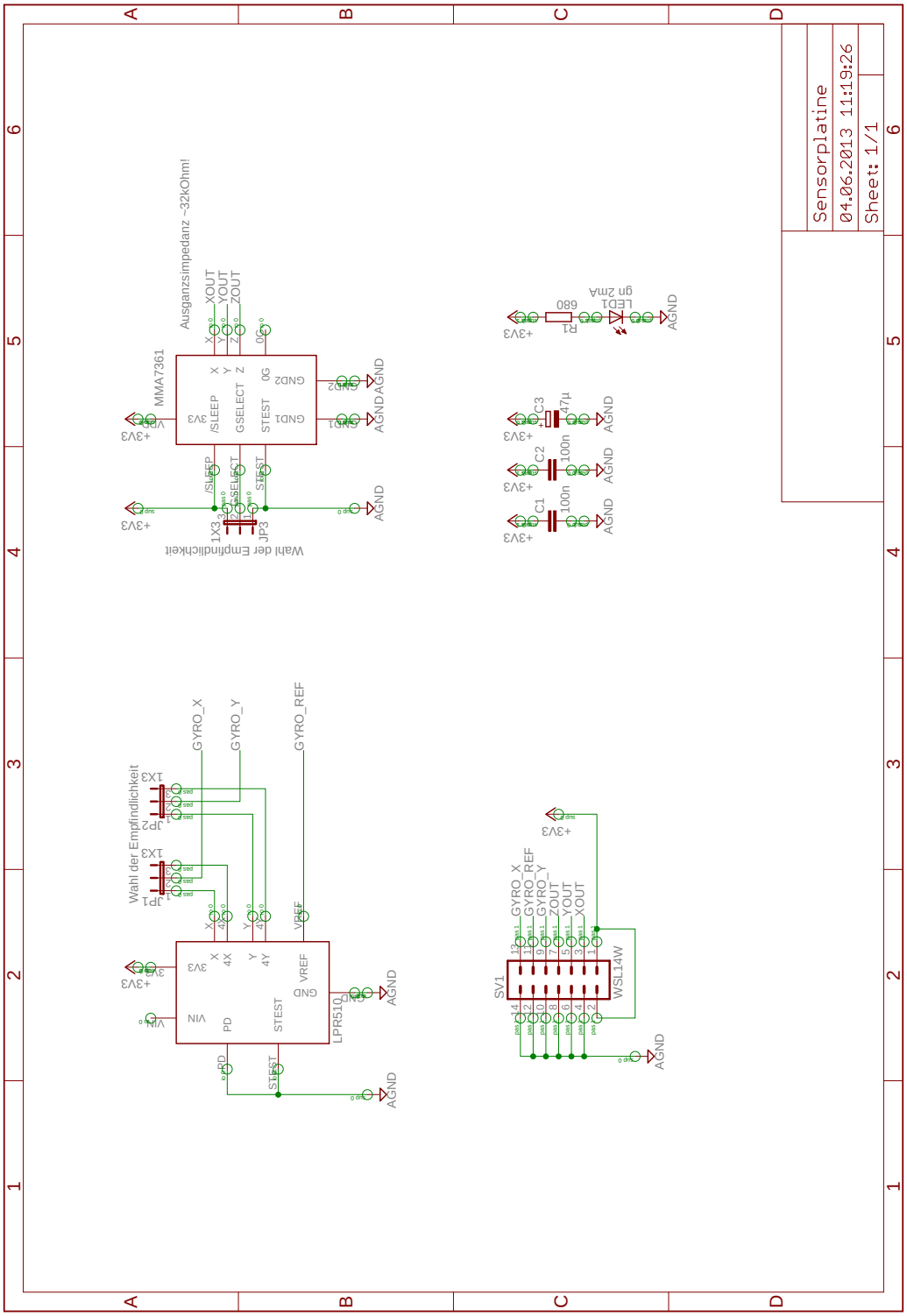


Abbildung B.2: Schaltplan der Sensorplatine

Anhang C

Schaltplan der Leistungsplatine

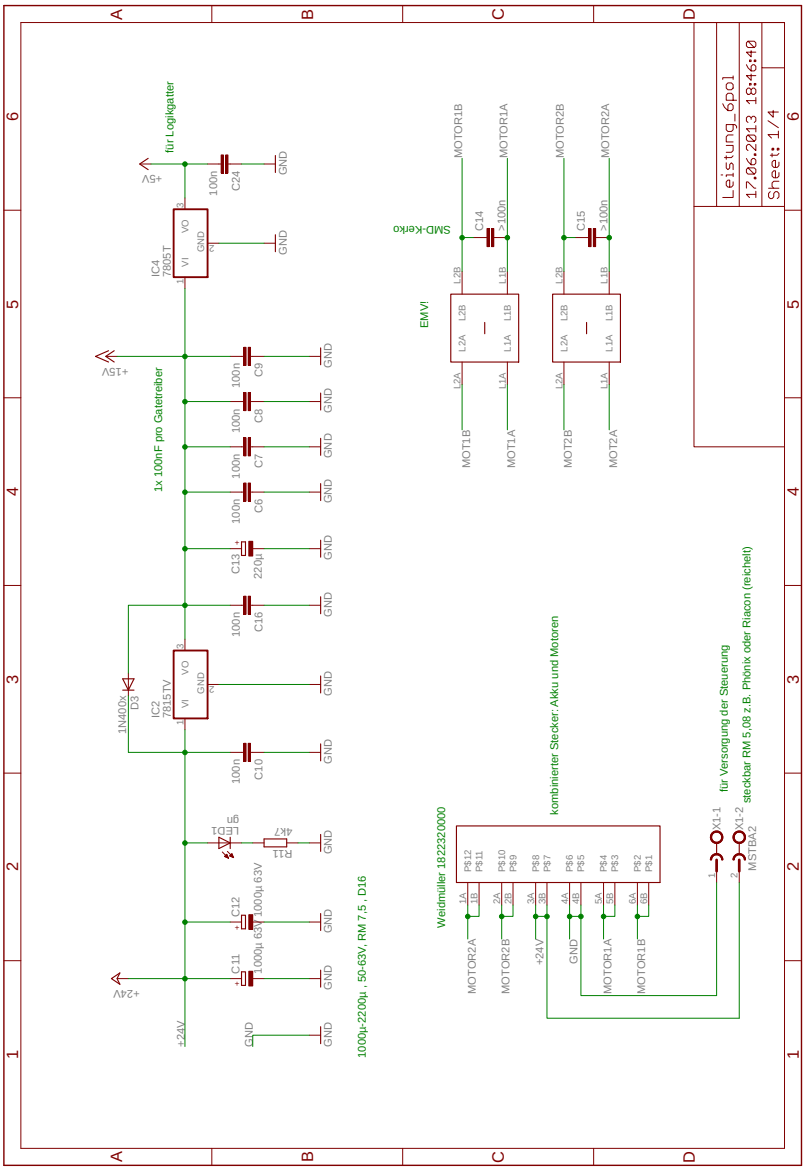


Abbildung C.1: Schaltplan der Leistungselektronik Teil 1

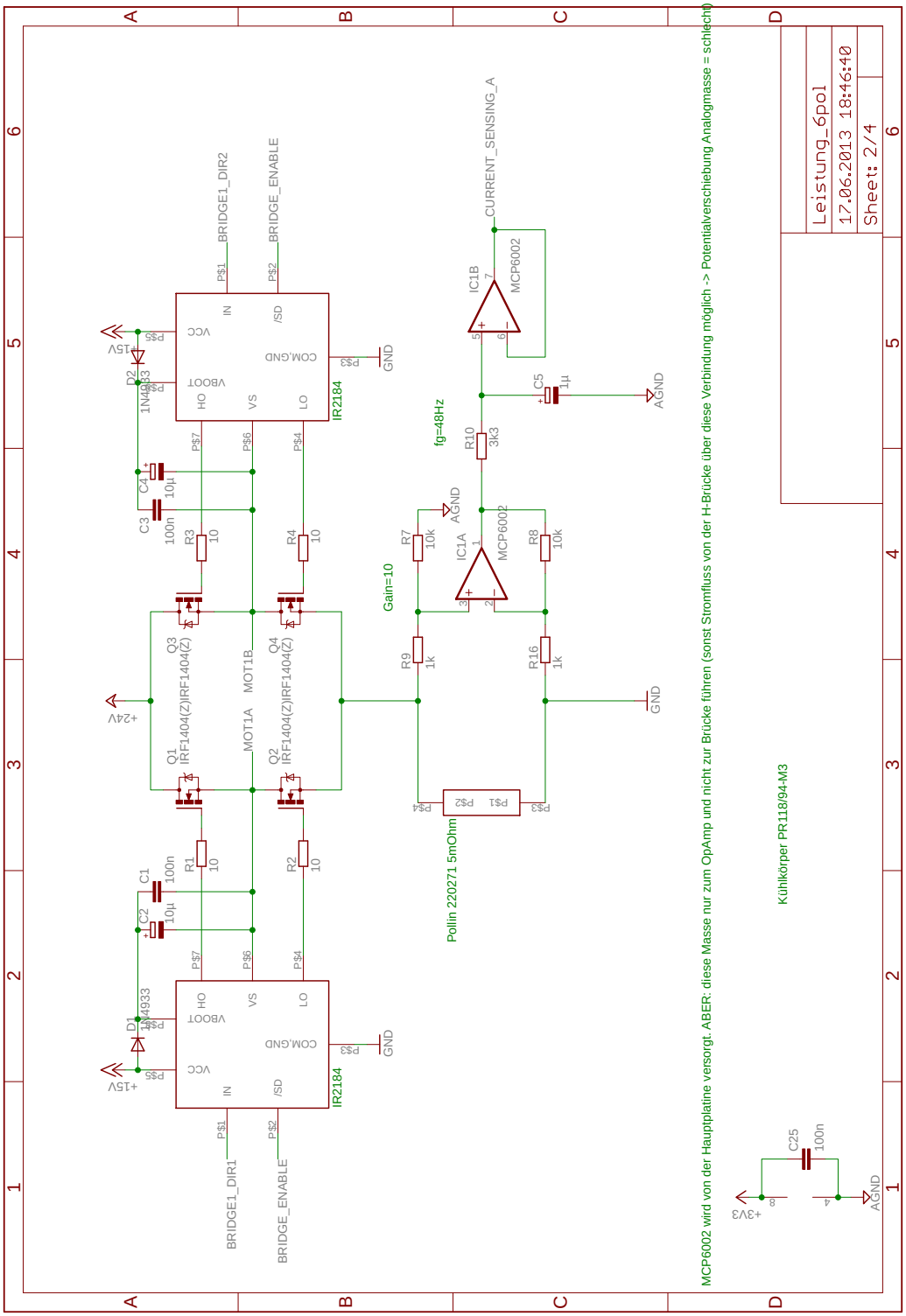


Abbildung C.2: Schaltplan der Leistungselektronik Teil 2

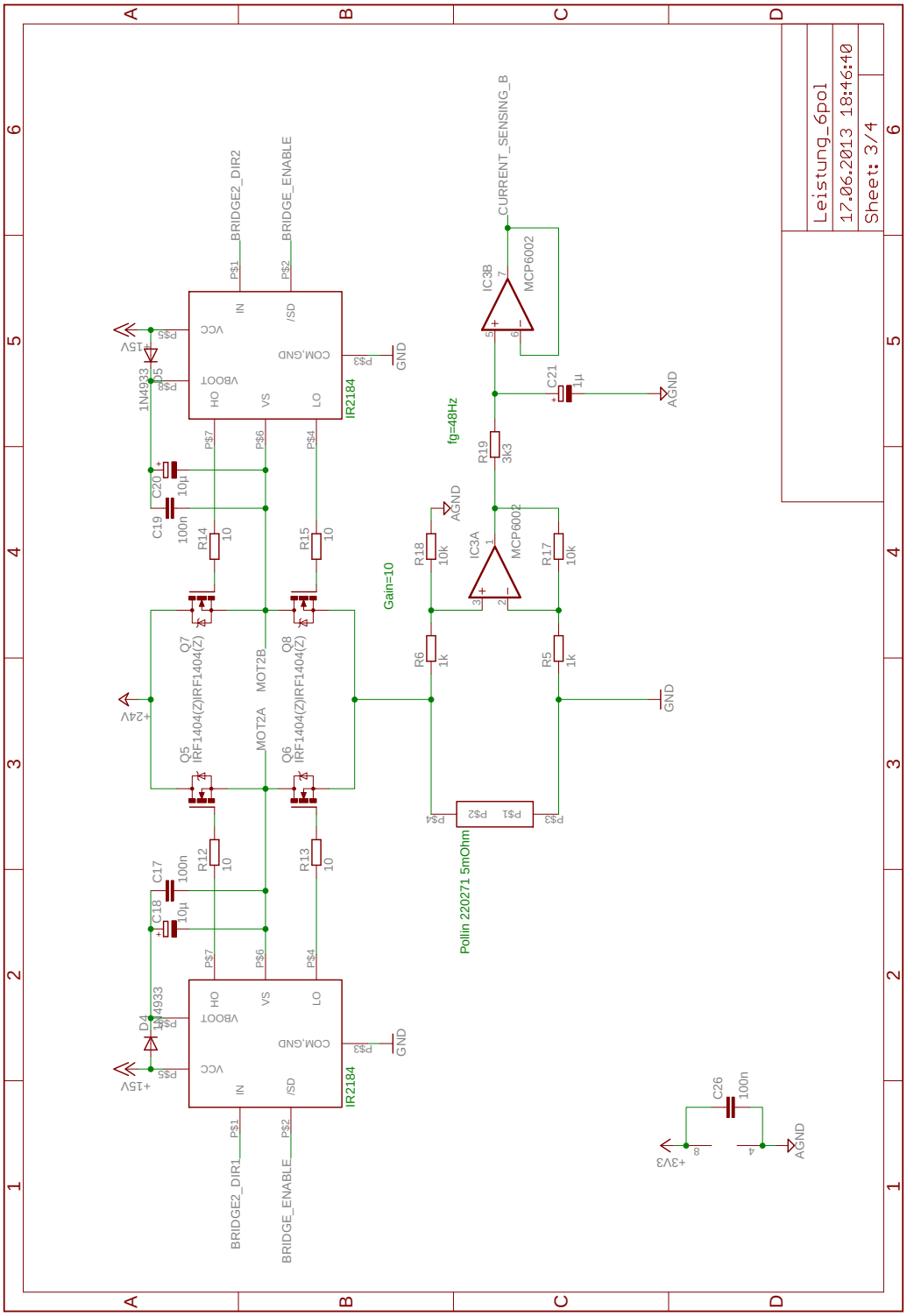


Abbildung C.3: Schaltplan der Leistungselektronik Teil 3

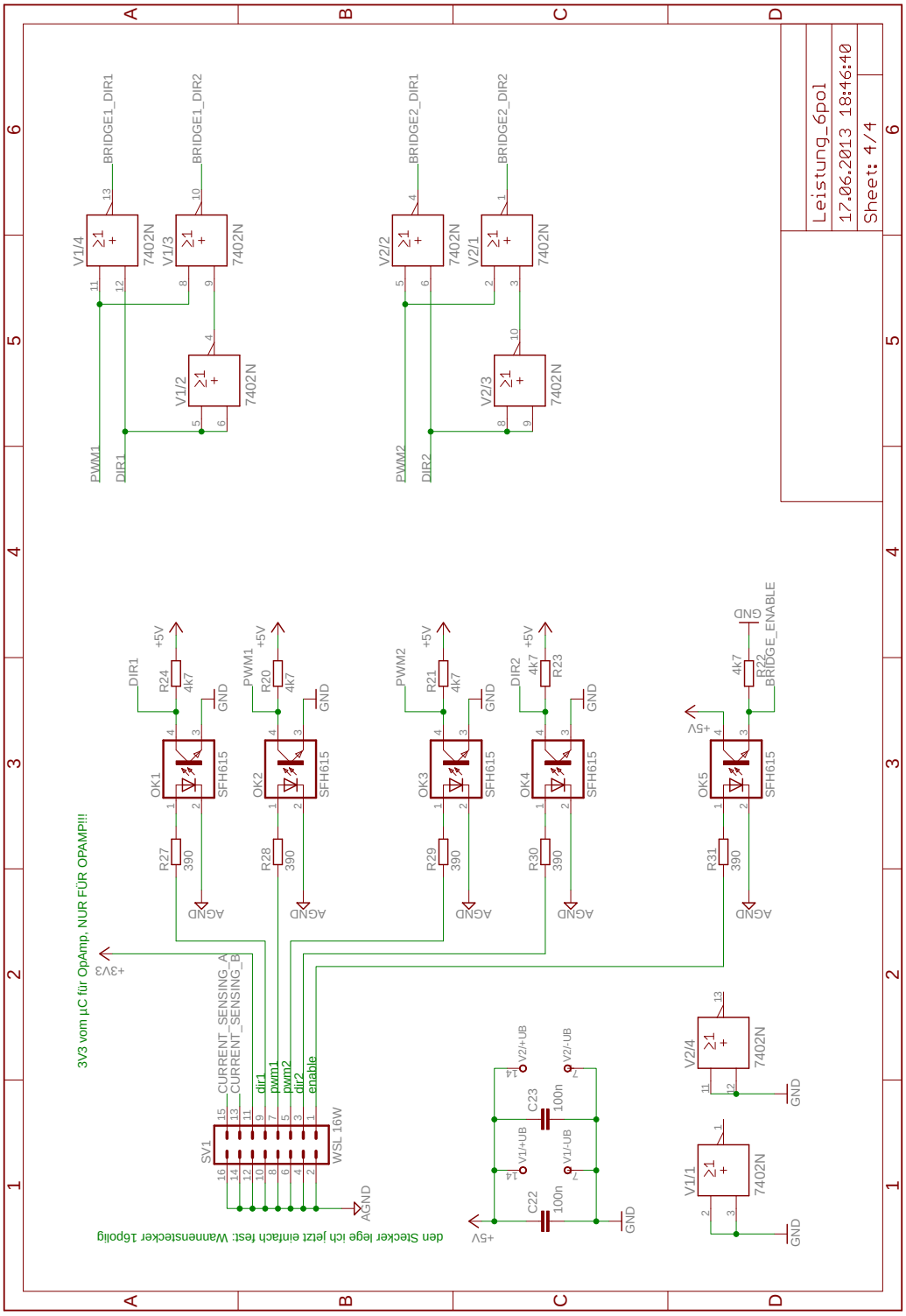


Abbildung C.4: Schaltplan der Leistungselektronik Teil 4