Julius-Maximilians-Universität Würzburg
Fakultät für Mathematik und Informatik

Informationstechnik für Luft- und Raumfahrt
Lehrstuhl für Informatik 8 Prof. Dr. Sergio Montenegro

Diplomarbeit

Design und Implementierung eines konfigurierbaren Controllers zur Ansteuerung bürstenloser Motoren und Regelung von Echtzeitsystemen

Vorgelegt von

Erik Dilger

Matr.-Nr.: 1603532

Erstgutachter: Prof. Dr. Sergio Montenegro

Zweitgutachter: Prof. Dr.-Ing. Hakan Kayal

Betreuender wissenschaftlicher Mitarbeiter: Dipl.-Ing. Nils Gageik

Würzburg, 28.02.2013

Erklärung

Ich versichere, dass ich die vorliegende Arbeit einschließlich aller beigefügter Materialien selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Werken entnommen sind, sind in jedem Einzelfall unter Angabe der Quelle deutlich als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form noch nicht als Prüfungsarbeit eingereicht worden.
Mir ist bekannt, dass Zuwiderhandlungen gegen diese Erklärung und bewusste Täuschungen die Benotung der Arbeit mit der Note 5.0 zur Folge haben kann.

Würzburg, 28.02.2013

_____

Erik Dilger

Aufgabenstellung:

Design und Implementierung eines konfigurierbaren Controllers zur Regelung von Echtzeitsystemen

Zur Regelung von Echtzeitsystemen sind heutzutage bürstenlose Motoren weit verbreitet. Controller, die diese bürstenlosen Motoren ansteuern, können heute käuflich erworben werden. Erhältliche Controller sind jedoch nicht frei konfigurierbar und teuer. Das Design dieser Systeme ist somit unflexibel und auf die Motorregelung begrenzt.

Um für Systeme zur Echtzeitregelung wie Quadrocoptern ein System bereitzuhalten, das ohne Aufwand unterschiedlichen Anforderungen nachkommt sowie zur Kostenersparnis und Optimierung soll im Rahmen dieser Arbeit ein ultimativer Controller basierend auf einem PSOC5 Mikrocontroller implementiert werden. Das System soll bürstenlose Gleichstrommotoren regeln und darüber hinaus einfach konfigurierbar sein, so dass auch Motoren mit PWM angesteuert werden können. Desweiteren soll eine Echtzeitregelung auf den Controller portiert werden. Ziel der Arbeit ist somit die Fusion der Motorregelung und Systemregelung (Attitude) in einem Mikrocontroller, der alles in einem macht.

Teil der Arbeit ist das Entwickeln und Realisieren einer Schaltung zur Regelung von bürstenlosen Motoren. Darüber hinaus ist die nötige Treibersoftware für das PSOC5 bereit zu halten, damit die Funktionen als Motorregelung und Echtzeitregelung erfüllt werden können. Dazu gehören Treiber für USART, I2C, SPI, GPIO, PWM sowie Timer Counter und Interrupts. Am Ende der Arbeit soll ein Echtzeitsystem portiert werden.

Zur Aufgabe gehört eine ausführliche Dokumentation aller Hard- und Software sowie eine detaillierte Erklärung der Funktionsweise des bürstenlosen Reglers. Das System ist am Ende umfangreich zu evaluieren.

Aufgabenstellung (Stichpunktartig):

- Ausführliche Erklärung BL-Ctrl
- Design & Realisierung Schaltung bürstenloser Motor
- Programmierung BL-Ctrl, PWM, I2C, SPI, USART, GPIO, TimerCounter, Interrupts
- Portierung Echtzeitsystem (z.B.: Quadcopter)
- Evaluierung des Systems (Tests)
- Dokumentation

Zusammenfassung

Das Thema dieser Arbeit ist die Entwicklung und Evaluierung eines Echtzeitregelsystems mit integrierten Controllern für bürstenlose Gleichstrommotoren. Diese Motoren zeichnen sich durch ihre hohe Leistungsdichte aus. Das Ziel ist die Entwicklung eines voll integrierten Fluglagenregelungssystems für Quadrokopter.

Hierzu wurde der Psoc5-Chip von Cypress benutzt. Dieser integriert einen ARM-Mikrocontroller und programmierbare Digital- und Analoghardware in einem Chip. Es wurde ein Brushlessmotorcontroller unter Verwendung der programmierbaren Hardwareblöcke dieses Chips entwickelt. Bei der Implementierung wurden dann die vier für einen Quadrokopter benötigten Brushlessmotorcontroller in der Hardware des Chips instanziiert. Durch die Ausführung in Hardware belasten die Motorcontroller nicht den Mikorcontroller und dieser ist für die Fluglagenregelung und andere Steueraufgaben frei. Für die Fluglagenregelung wurde die bestehende Software aus dem Quadrokopterprojekt AQopterI8 des Lehrstuhls auf das System portiert. Ebenfalls wurde eine Platine für die Leistungselektronik der vier Brushlesscontroller entworfen, hergestellt und getestet.

Am Ende wurde das System in einen Quadrokopter eingebaut und ein erfolgreicher Flugtest durchgeführt.

Inhalt

Verzeichnis der Abbildungen

Verzeichnis der Abkürzungen

BLMC	Brushless Motor Controller
BL-Motor	Brushless Motor
IMU	Inertial Measurement Unit (Inertiale Messeinheit)
UDB	Universal Digital Block
FET	Feld Effekt Transistor
API	Application Programming Interface

1 Einleitung

Autonom fliegende Fluggeräte sind wegen der Verfügbarkeit von preisgünstigen und leistungsfähigen Komponenten wie Sensoren, Mikrocontrollern und Motoren aktuell ein beliebtes Forschungsthema. Hierbei sind Quadrokopter mit vier Rotoren eine beliebte Bauform. Diese benötigen, um stabil zu fliegen, zwingend eine aktive Regelung der Fluglage. Quadrokopter werden vorwiegend elektrisch angetrieben. Hier werden wegen ihrer Effizienz und Leistungsdichte vorwiegend bürstenlose- Gleichstrommotoren eingesetzt. Diese benötigen einen speziellen elektronischen Controller zum Betrieb.

Üblicherweise ist bei einem solchen Quadrokopter für jeden Motor ein Motorcontroller an ein Fluglagenregelungssystem angeschlossen. In dieser Arbeit wird nun ein Echtzeitregelsystem mit vier integrierten Motorcontrollern entwickelt, um ein integriertes, kompaktes System zur Fluglagenregelung und Brushlessmotoransteuerung von Quadrokoptern zu erhalten. So kann man die Anzahl der Einzelkomponenten reduzieren, und reduziert somit die Komplexität des Systems und verringert mögliche Fehlerquellen.

Das Herausstellungsmerkmal hierbei ist, dass der Brushlessmotorcontroller (BLMC) mit einer Hardwareschaltung realisiert wird. Hierzu wird der Psoc5 Chip von Cypress benutzt. Dieser verbindet programmierbare Hardwareschaltungen mit einem ARM-Mikrocontroller. Somit wird der Mikrocontroller nicht mit der Motorsteuerung belastet, wenn diese von den konfigurierbaren Hardwareblöcken ausgeführt wird. Somit steht ihre gesamte Rechenzeit für die Fluglagenregelung und für andere Steueraufgaben des Quadrokopters zur Verfügung. Um das System später auch für andere Aufgaben benutzen zu können, soll die physikalisch vorhandene Endstufe auch für andere Motorarten genutzt werden können. Dies ist durch die Reprogrammierbarkeit der Schaltkreise des Psoc-Chips gegeben.

In Kapitel 2 dieser Arbeit werden die Unterscheide von Brushlessmotoren zu klassischen Gleichstrommotoren, sowie Funktionsweise von Brushlessmotoren und der Brushless-Motorcontroller erklärt. Ebenso wird auf verschiedene Arten der Kommutierungserkennung eingegangen. In Kapitel 3 wird ein Konzept eines Echtzeitregelsystems mit integrierten Motorcontrollern entwickelt. In Kapitel 4 wird ein BLMC für die

konfigurierbare Hardware des Psoc implementiert. Dazu wird eine Platine hergestellt, welche die erforderlichen externen Komponenten für vier BLMC bereitstellt und den Psoc aufnimmt. Zur Fluglagenregelung für den Quadrokopter, wird die am Lehrstuhl benutze Software auf den Psoc portiert. In Kapitel 5 werden die Funktionen und Eigenschaften des implementierten BLMC getestet. Ebenso wird das komplette System mit vier BLMC in einen Quadrokopter eingebaut und einem Flugtest unterzogen. Kapitel 6 zeigt die Verbesserungsmöglichkeiten und gibt einen Ausblick auf weitere Entwicklungsmöglichkeiten.

2 Stand der Technik

2.1 Klassische Gleichstrommotoren mit Bürsten

Klassische Bürstengleichstrommotoren bestehen aus einem Stator und einem Rotor (vgl. Abb 1). Der Rotor ist dabei der rotierende Teil, an dem die Antriebswelle befestigt ist, der Stator der stillstehende Teil, welcher mit dem Gehäuse verbunden ist. Im Stator enthaltene Permanentmagnete durchsetzen den Rotor mit einem konstanten Magnetfeld. Fließt durch die Spule im Rotor ein Strom, erzeugt dieser ein Magnetfeld, welches in magnetische Wechselwirkung mit dem Magnetfeld der Stators tritt. So wird durch magnetische Anziehungs- und Abstoßungskräfte eine Kraft auf den Rotor ausgeübt. Aufgrund des Drehmomentes, das sich aus der Kraft ergibt, dreht sich der Rotor solange, bis sein Magnetfeld mit dem des Stators parallel ausgerichtet ist. Damit sich jetzt eine kontinuierliche Drehung ergibt, muss das Magnetfeld des Rotors, und somit der Stromfluss durch die Spule, jede halbe Umdrehung umgepolt werden. Diesen Vorgang bezeichnet man als Kommutieren. Da man bei dieser Art Gleichstrommotor Schleifkontakte benutzt, um der sich drehenden Spule auf dem Rotor Strom zuzuführen, wird die Kommutierung gleich durch diese realisiert. Dazu wird ein Zylinder in zwei zueinander elektrisch isolierte Segmente unterteilt(vgl Abb. 1). Jedes dieser Segmente ist mit einem Pol der Spule verbunden. Auf diese Segmente wird nun mit Kohlebürsten, die auf der Oberfläche schleifen, Strom geleitet. Diese haben nach einer halben Umdrehung immer Kontakt zum jeweils anderen Segment. Wegen der Kohlebürsten bezeichnet man diese Motoren auch als Bürstenmotoren.

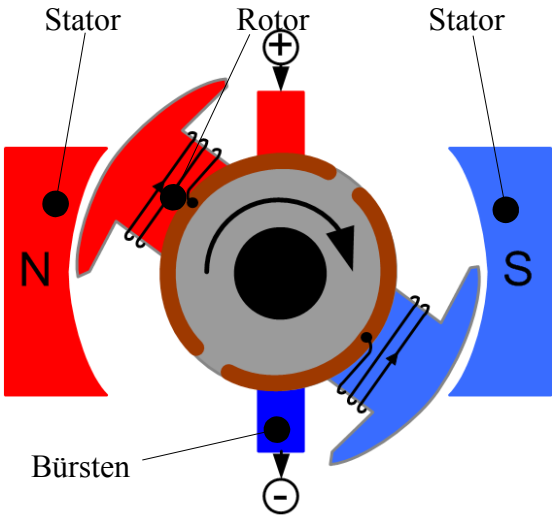

***Abbildung 1: Herkömlicher Gleichstrommotor* [Gleichstrommotor]**

Diese Bürsten haben jedoch den Nachteil, den Motor durch Reibung zu bremsen und Verlustwärme zu erzeugen, sowie zu verschleißen. Sie stellen ebenfalls einen erhöhten elektrischen Widerstand im Vergleich zum Kupferleiter dar und sorgen so zu-

sätzlich für Wärmeverluste. Ebenfalls nachteilig bei diesem Motor ist die Anordnung der stromdurchflossenen Spule auf dem rotierenden Teil. Dies erschwert die optimale Abfuhr der Verlustwärme, weil dies nur über die Luft und nicht über metallische Gehäuseteile erfolgen kann.
[Gleichstrommotor]

2.2 Funktionsweise von Brushlessmotoren

Bei einem Brushlessmotor lässt man diese Bürsten weg und verlegt die Spulen in den Stator. Diese können nun auch durch Wärmeabgabe an das Gehäuse gekühlt werden. Der Rotor ist nun der Permanentmagnet, der sich im Magnetfeld der um ihn herum angeordneten Spulen dreht (vgl. Abb 3). Die Kommutierung erfolgt nun durch eine elektronische Schaltung, dem Brushlessmotorcontroller (BLMC). Die Schwierigkeit hierbei ist den korrekten Zeitpunkt der Kommutierung zu erkennen.

In Abb. 3 ist das Schema eines im Modellbau üblichen Drei-Phasen-Brushless-Motor zu sehen. Die zwei jeweils gegenüberliegenden Spulen sind dabei in Serie geschaltet und gehören zu einer Phase. Die eine Seite dieser Spulenpaare ist dabei motorintern zusammengeschaltet. Da sich hier alle drei Phasen sternförmig treffen, nennt man diesen Punkt der Zusammenschaltung Sternpunkt (Abb.3 com). Die andere Seite wird jeweils als Anschluss herausgeführt, so dass es für jede der drei Phasen einen Anschluss gibt. Die Phasen werden jetzt nach dem in Abb.2 dargestellten Abblauf angesteuert. Dabei ist pro Zyklus immer genau eine Phase mit der Betriebsspannung verbunden (high), eine mit Masse (low), und eine hat keine Verbindung (floating). Es fließt also immer Strom durch genau zwei Phasen.

Im ersten Zyklus liegt Phase A auf Spannung und Phase C auf Masse. Nun fließt, wie in Abb. 3 Teilbild a) dargestellt, Strom über den Pfad A-a-com-c-C. Daraus ergibt sich ein Magnetfeld, nach dem sich der Rotor wie dargestellt ausrichtet. Jetzt erfolgt die Kommutierung und das System befindet sich im zweiten Zyklus (Abb. 3, Teilbild b). Phase C bleibt hier weiterhin auf Masse, so dass sich der Stromfluss durch die Spule an C nicht ändert. Aber es wird anstatt Phase A jetzt Phase B auf Spannung geschaltet, so dass sich jetzt der Strompfad B-b-com-c-C ergibt. Das durch den Stator erzeugte Magnetfeld hat sich jetzt um 60° gedreht und der Rotor folgt diesem und erreicht schließlich die Ausrichtung, die auf Abb. 3 Teilbild b) zu sehen ist. Jetzt wird wieder kommutiert. In Zyklus drei bleibt nun Phase B auf Spannung und die auf Masse befindliche Phase wird umgeschaltet. Es ergibt sich ein Strompfad von B-b-com-a-A und eine Rotorausrichtung wie in Abb. 3 Teilbild c) zu sehen. Die letzten drei Zyklen sind in Teilbild d) zusammengefasst und verlaufen analog der ersten drei, so dass das Statormagnetfeld und der Rotor schließlich eine komplette Umdrehung durchlaufen. Der ganze Ablauf wiederholt sich nach 6 Zyklen, so dass eine kontinuierliche Drehbewegung entsteht.

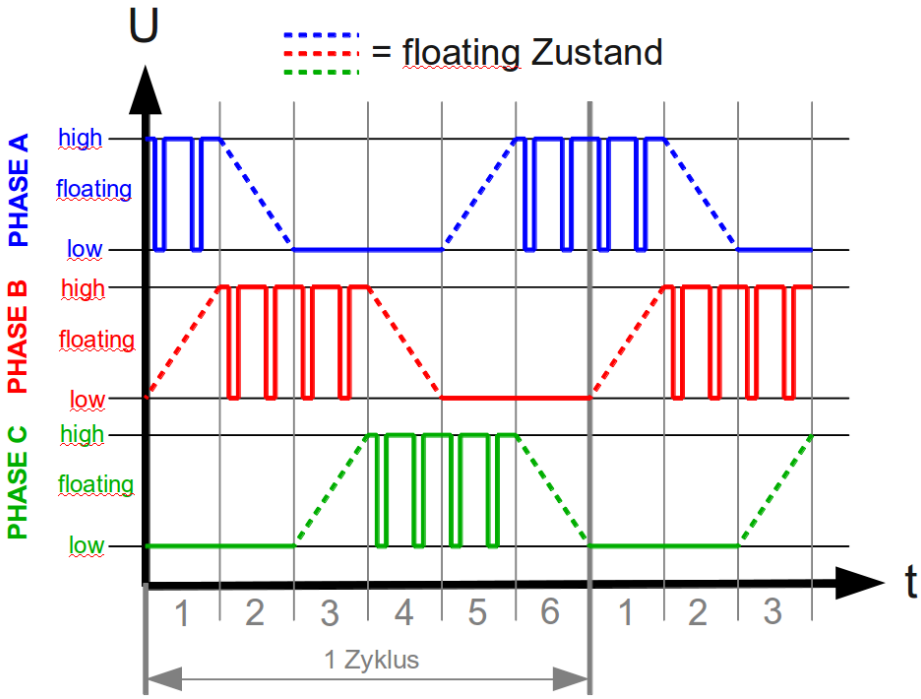

Abbildung 2: Die sechs Ansteuerzyklen eines BL-Motors **[Mikrocontroller]**

In Wirklichkeit richtet sich der Rotor aber nicht genau zum Statormagnetfeld aus, sondern dreht sich 90° versetzt zu diesem, da sich so das maximale Drehmoment ergibt, weil die Kraft maximal ist, wenn die beiden Magnetfelder senkrecht zueinander stehen. Die am Rotor angreifende Kraft ist direkt proportional zum Sinus des Winkels zwischen den Magnetfeldern, und somit bis 90° am stärksten.

In diesem vereinfachten Modell entsprechen 6 Zyklen einer Umdrehung des Rotors. Man spricht hier auch von einer elektrischen Umdrehung. Der Modellmotor hat also

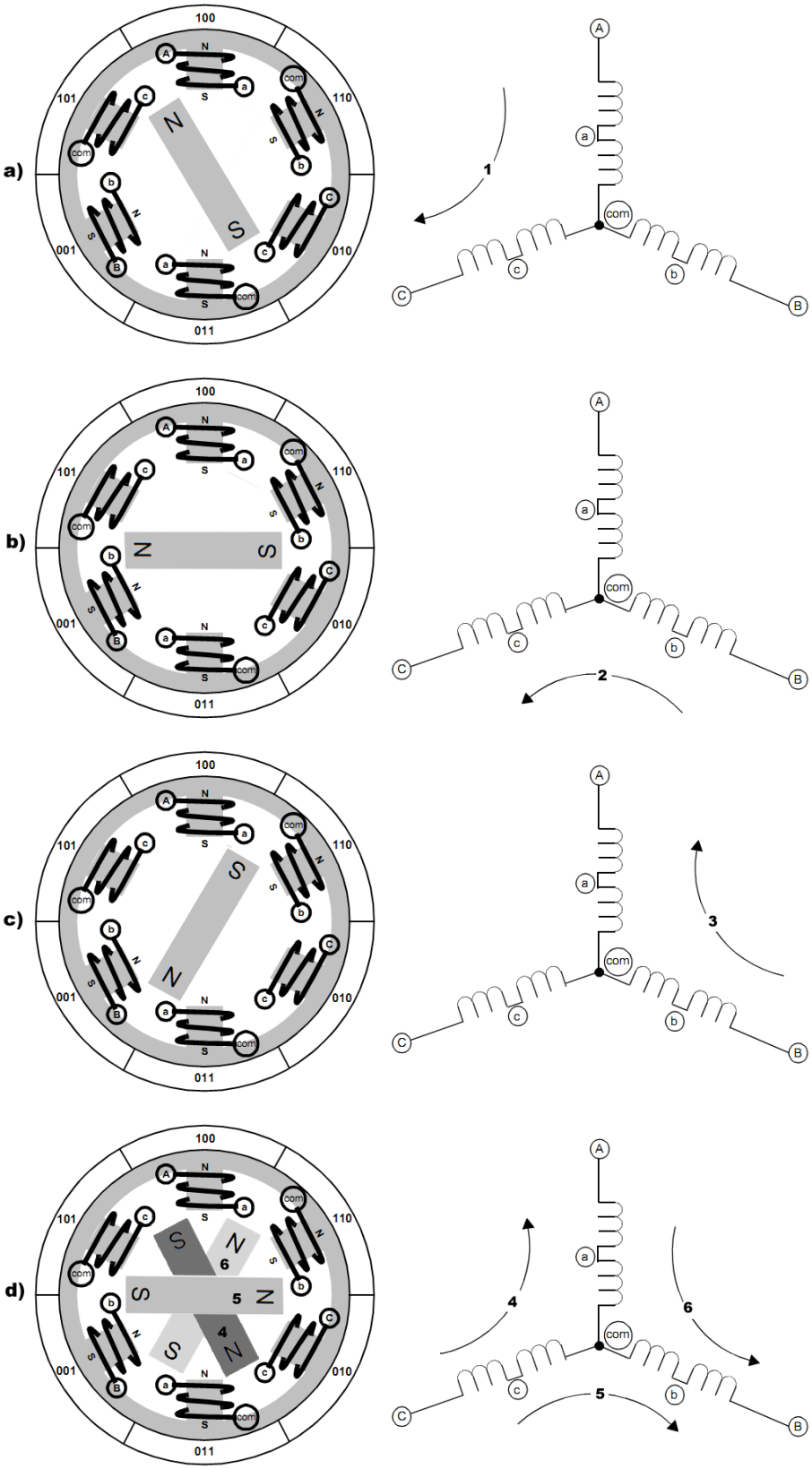

Abbildung 3: Schema und einzelne Zyklen eines BL-Motors **[Microchip]**

einen Pol. Reale Motoren besitzen allerdings eine n-fache Anzahl an parallel geschalteten Spulen (also bei drei Phasen 6*n Spulen) im Stator und Permanentmagneten im Rotor (vgl. Abb. 4). Der Motor besitzt also n Pole oder hat eine Polzahl von n. Eine höhere Polzahl, erhöht die am Rotor angreifende Kraft. Sie dient damit dazu, das zu der Kraft direkt proportionale Drehmoment des Motors zu erhöhen. So bringt der Motor bei gleichem Hebel eine größere Antriebskraft auf. Die Drehzahl wird dabei um den Faktor n reduziert. Eine reale Umdrehung enthält also immer n elektrische Umdrehungen.

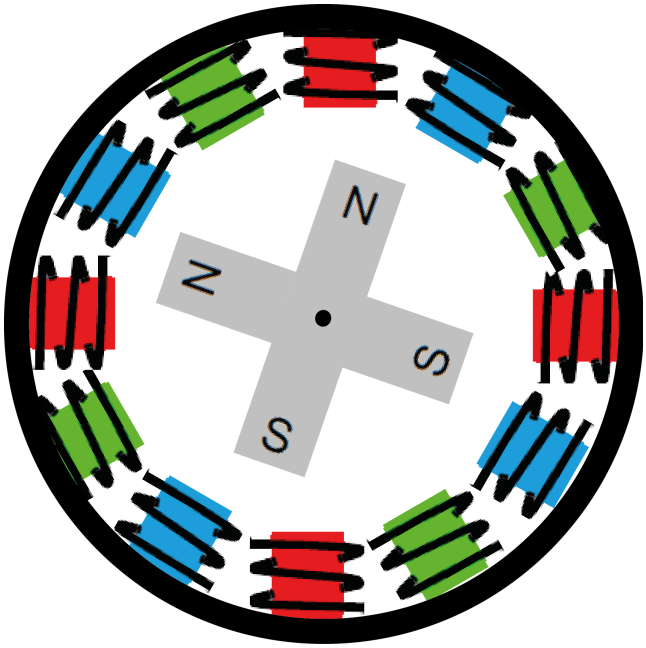

Abbildung 4: Zweipoliger BL-Motor: Spulen gleicher Farbe gehören zu einer Phase

Der Vorteil dieser elektronischen Kommutierung gegenüber der herkömmlichen mechanischen durch Kontaktbürsten ist, dass keine Bürsten vorhanden sind, welche verschleißen, Reibung erzeugen und einen zusätzlichen elektrischen Widerstand bieten. BL-Motoren sind somit wartungsfreier und verschleißfester als herkömmliche Gleichstrommotoren. Wegen der geringeren Reibung und des Wegfalls des Übergangswiderstandes der Kohlebürsten ist der Wirkungsgrad höher und die Wärmeverluste geringer. Dies ermöglicht besonders hohe Leistungsdichten und macht den Brushlessmotor ideal für den Modellflug, wo große Antriebsleistungen bei wenig Gewicht und Raum benötigt werden.

Eine weitere übliche Bauform von BL-Motoren ist der sogenannte Außenläufer. Hier ist der Rotor mit den Permanentmagneten außen um den Stator mit den Spulen angeordnet. Diese Bauform verhält sich elektrisch genauso wie ein Innenläufer, besitzt aber ein größeres Drehmoment als Innenläufer. Der Grund ist, dass die Kraft im kreisförmigen Luftspalt zwischen Stator und Rotor angreift. Dessen Radius ist beim Außenläufer größer. In Modellfluggeräten kann man sich durch dieses größere Drehmoment meistens ein zusätzliches Getriebe, welches weiter Leistungsverluste und Verschleiß bedeutet, sparen und Propeller direkt antreiben.[Torquemotor]
[Probst] [Microchip] [Mikrocontroller] [Atmel]

2.3 Funktionsweise eines Brushlessmotorcontroller

BL-Motoren benötigen zwingend einen Brushlessmotorcontroller(BLMC), um zu funktionieren. Dieser übernimmt vor allem die Aufgabe der elektronischen Kommutierung, kann aber noch weiter Features enthalten, wie z.B. eine Strombegrenzung oder eine Drehzahlregelung.

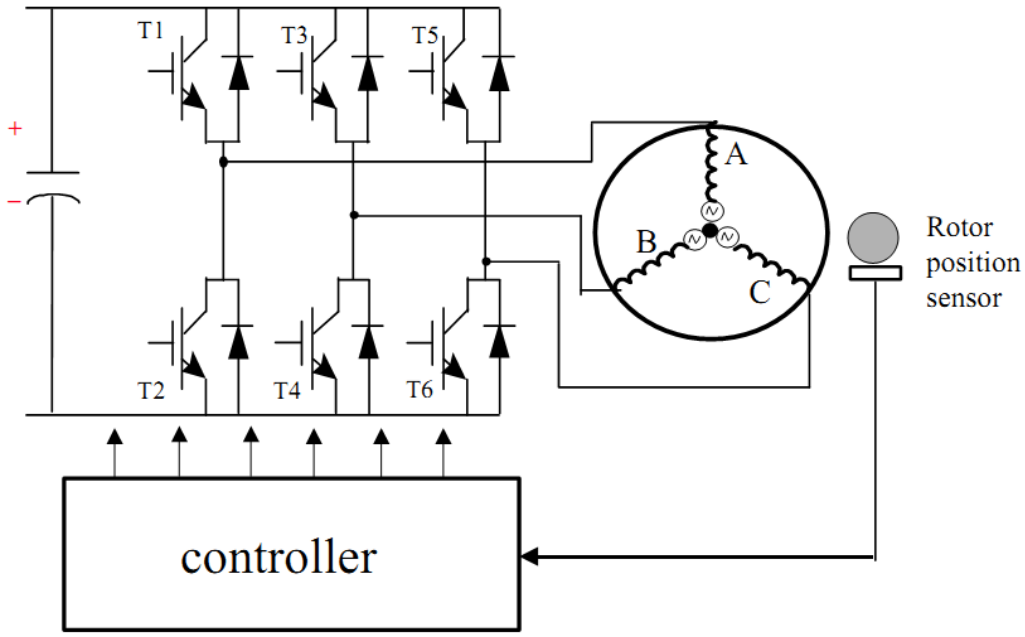

Abbildung 5: Schema eines BLMC mit optionalem Rotorpositionssensor **[Shao]**

Der BLMC besteht wie in Abb. 5 im Wesentlichen aus einem Controller und einer Endstufe. In der Endstufe gibt es für jede Phase ein Transistorpaar, das vom Controller angesteuert wird. Es existieren also insgesamt sechs Transistoren. Von jedem der drei

Paare schaltet ein Transistor die Phase auf die Betriebsspannung (T1, T3, T5), der andere auf Masse(T2, T4, T6). Ist keiner der beiden Transistoren eingeschaltet, ist die Spannung an der Phase „frei schwebend“(floatend). Es dürfen niemals beide Transistoren einer Phase eingeschaltet werden, denn sonst entstünde ein Kurzschluss.

Der Controller steuert nun die einzelnen Phasen entsprechend der in Kap. 2.2 beschriebenen Abfolge an. Dazu muss er den Zeitpunkt ermitteln, wann genau er kommutieren muss. Bei einem Sensor-Motor sind im Motor Sensoren integriert. Durch diese erfasst ein BLMC für BL-Motoren mit Sensoren die genaue Position des Rotors. Anhand dieser und bekannter Daten über den Motor, wie z.B. die Polzahl, kann er den nach Abb. 2 aktuell richtigen Ansteuerzyklus wählen und den Zeitpunkt der nächsten Kommutierung bestimmen. Dieses Verfahren funktioniert auch bei langsamen Drehzahlen und aus dem Stand heraus, benötigt allerdings spezielle Motoren mit Sensoren.

Eine andere Möglichkeit, den Zeitpunkt der Kommutierung zu erkennen, ist, die an der nicht beschalteten Phase induzierte Spannung zu messen. Dies funktioniert allerdings nicht aus dem Stand heraus und nicht bei einer zu geringen Drehzahl, weil die Höhe der induzierten Spannung von der Drehzahl abhängt und nicht mehr zuverlässig ausgewertet werden kann, wenn sie zu gering ist. Um den Motor zu starten, wird bei dieser Methode eine niedrige Kommutierungsfrequenz fest vorgegeben. Der Motor passt sich so der Zyklusabfolge des Controllers an und wird auf eine niedrige Anfangsdrehzahl gebracht. Anschließend wird in den Normalbetrieb mit automatischer Kommutierungszeitpunkterkennung umgeschaltet. Der Vorteil dieser Methode ist, dass sie keinen Sensor im Motor benötigt und den Verkabelungsaufwand reduziert. Auch sind BL-Motoren und BLMC ohne Sensor im Modebau weit verbreitet und somit sehr günstig zu erwerben.

Die Leistungsregelung und somit die Drehzahlregelung erfolgt durch die am Motor angelegte Spannung. Am Motor stellt sich dann je nach Last und Kenndaten des Motors eine bestimmte Drehzahl ein. Die Kommutierungsfrequenz ist dabei nicht von der angelegten Spannung abhängig, sondern passt sich nur dem Verhalten des Motors an. Der Motor gibt also die Kommutierungsfrequenz vor. Die Leistungssteuerung des Motors erfolgt ausschließlich durch Veränderung der Spannung und nicht durch Beeinflussung der Kommutierungsfrequenz. Um nun effizient am Motor eine Effektivspannung einstellen zu können, wird PWM benutzt. Dazu werden die spannungsseitigen

oder die masseseitigen Transistoren nicht mit einem kontinuierlichen Signal angesteuert, wenn diese im aktuellen Zyklus angesteuert werden sollen, sondern mit einem PWM-Signal. So werden die aktuell angesteuerten Phasen nicht kontinuierlich mit der Betriebsspannung angesteuert, sondern mit eine PWM-Modulation.
[Probst] [Microchip] [Mikrocontroller] [Atmel] [Shao]

2.4 Kommutierungszeitpunkterkennung eines sensorlosen BLMC mit Back-EMF

Bei Systemen ohne Sensor zur Erfassung der Rotorposition muss eine andere Möglichkeit gefunden werden, wie der BLMC den Zeitpunkt erkennt, wann er kommutieren muss. Eine gängige Methode ist die Auswertung der Spannung der aktuell floatenden Phase. Der Verlauf dieser Spannung ist, wie in Abb 2 zu sehen immer an- bzw. absteigend von Masse zur Betriebsspannung oder umgekehrt. Diese Spannung wird mit der Spannung des Sternpunktes des Motors mit einem Analogkomparator verglichen. Kreuzt sie das Potenzial des Sternpunktes, ist der Kommutierungszeitpunkt erreicht. Ob die Kreuzung aufsteigend oder absteigend erfolgt, hängt dabei vom aktuellen Zyklus ab. Je nachdem muss dann auf eine auf- oder absteigende Flanke des Digitalsignals am Komparator reagiert werden.

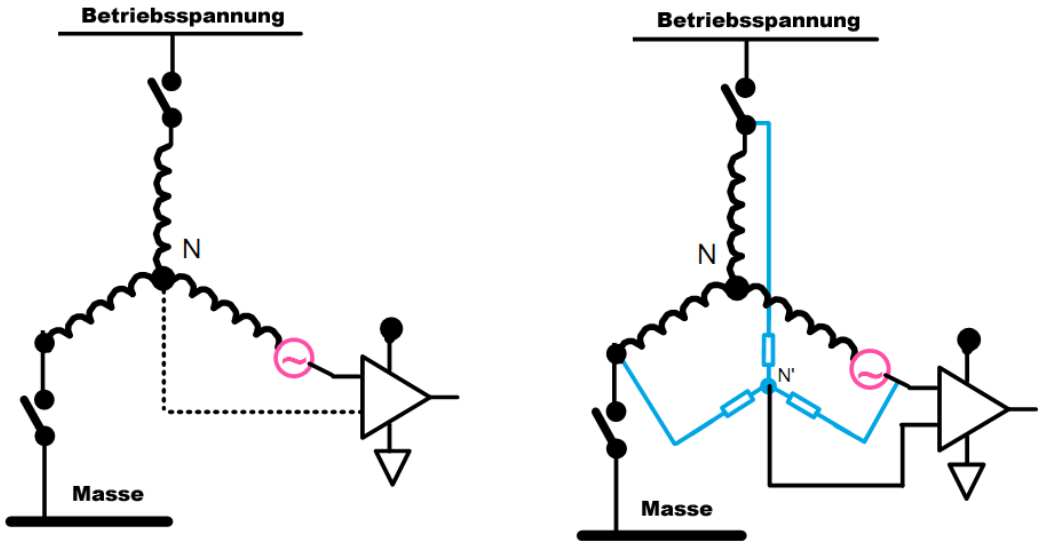

Abbildung 6: Auswertung der Back-EMF Spannung ohne und mit virtuellem Sternpunkt **[Shao]**

Da der Sternpunkt in der Regel nicht extra aus dem Motor herausgeführt wird, muss dieser im Controller nachgebildet werden. Dies geschieht, wie in Abb 6 rechts dargestellt, indem je ein Widerstand von jeder Phase zu einem virtuellen Sternpunkt verbunden wird. Dieser kann dann innerhalb des Controllers zur Auswertung genutzt werden und man kommt mit drei Kabeln zum Motor aus. Ebenso ist es nötig wegen der PWM Ansteuerung des Motors mit einem Tiefpassfilter die PWM-Frequenz aus dem Signal zu filtern.
[Microchip] [Mikrocontroller] [Atmel] [Shao]

2.5 Bestehende Systeme

Ein preisgünstiges bestehendes System am Markt ist der Burshlesscontroller von Mikrokopter. Dieser kann einen sensorlosen BL-Motor mit einem Dauerstrom von 12A bzw 35A in der neueren Version ansteuern. Die Drehrichtung kann dabei nicht verändert werden. Er kann Steuerwerte per PPM(Puls-Pausen-Modulation) und I2C-Bus entgegennehmen. Als Zusatzfeature wird eine Strombegrenzung geboten, eine Drehzahlregelung steht nicht zur Verfügung. Er besteht aus einer MOSFET-Endstufe und einem Atmega8 Mikrocontroller, welcher die Steuerung, PWM-Modulation und Kommutierung übernimmt. Die Software ist bei der neuen Version nicht mehr quelloffen und bei der alten Version sind ca. 90% des verfügbaren Programmspeichers belegt, so dass die Implementierung neuer Features schwer bis unmöglich ist. Auch ist dadurch, dass die Kommutierung durch Software ausgeführt wird, die verfügbare Rechenzeit eingeschränkt. Ebenso ist für jeden Motor eine eigener Controller notwendig. So benötigt man z.B. für einen Quadrokopter vier BLMC, die über I2C mit einer Zentraleinheit verbunden sind. Dies bedeutet zusätzlicher Verkabelungsaufwand und der I2C Bus stellt eine mögliche Fehlerquelle dar.

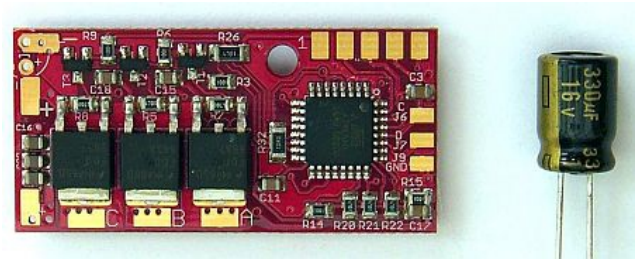

Abbildung 7: BLMC von mikrokopter.de **[Mikrokopter]**

Andere Systeme stellen zwar vier Brushlesscontroller auf einer Platine zur Verfügung, sind aber auch in Ihrer freien programmierbarkeit eingeschränkt und/oder sehr teuer.

System	Strom	Motoren	Programmier-bar	Preis	
Mikrokopter 1.2	12A	1	+	39.95 €	[Mikrokopter]
Mikrokopter 2.0	35A	1	-	54,95 €	
Mikrokopter 2.0 Quadro	35A	4	-	321.30 €	
Herkules III	30A	4	-	?	[Herkules]
NGBLC-4mini	30A	4	+	349,00 €	[UAVP]

Tabelle 1: Übersicht über verschiedene BLMC

2.6 Zusammenfassung

Zusammenfassend ist zu sagen, dass man im Modellfug wegen der benötigten großen Leistungen und Kräfte bei gleichzeitig geringem Gewicht und Größe, auf Brushlessmotoren angewiesen ist. Man ist aber auch auf günstige Systeme bedacht und setzt so bevorzugt sensorlose BL-Motoren ein. Deren Nachteile, nämlich die problematische Anlaufphase und die Schwierigkeiten bei niedrigen Drehzahlen, sind hier nicht von Bedeutung.

Benötigt man z.B. für einen Quadrokopter mehrere Motoren, möchte man die vier notwendigen BLMC am besten auf einer Platine integrieren, um Platz und Gewicht zu sparen. Optimaler Weise integriert man auf der selben Platine gleichzeitig die Regelung des Gesamtsystems und spart so weiter Platz und fehleranfällige Schnittstellen. Solche System sind aber, sofern überhaupt existent, sehr teuer. Alle am Markt befindlichen Systeme haben gemeinsam, dass die vor allem beim Einsatz in der Forschung gewünschte freie Reprogrammierbarkeit, gar nicht oder nur sehr eingeschränkt vorhanden ist.

Aufgrund dieser Nachteile soll nun ein neuer, flexibler Mehrfach-Brushlesscontroller mit integrierter Regelungseinheit entwickelt werden.

3 Konzept

3.1 Überblick

Es wird nun ein Konzept entwickelt werden, welches alle in Kapitel 2.6 aufgeführten Nachteile aufhebt. Es wird ein integriertes, flexibles und reprogrammierbares Regelungs- und Steuersystem (Echtzeitregelungssystem) für verschiedene Sensoren und Aktuatoren beinhalten. Insbesondere kann das System verschiedene Motoren, vor allem Brushless und PWM-gesteuerte Gleichstrommotoren ansteuern. Es soll auch bei der Ansteuerung von mehreren Motoren noch genug Ressourcen für Regelaufgaben verfügbar sein. Ein günstiger Preis sowie eine hohe Strombelastbarkeit werden ebenfalls als Anforderung gestellt.

Dabei steht vor allem die Anwendung zur Lageregelung von Modellfluggeräten wie z.B. bei einem Quadrocopter im Vordergrund. Die Lageregelung ist bei solchen Fluggeräten essentiell, weil sie ein natürlich instabiles System sind. Ohne kontinuierliches Erfassen der aktuellen Lage und Gegensteuern durch einen Lageregler über die Rotoren, würde der Quadrokopter aus der ebenen Fluglage in Schieflage geraten und abstürzen.

Das Gerät erscheint dem Benutzer dabei wie eine „Blackbox“. Diese erhält über eine Schnittstelle die Messung eines Sensors in Form eines Rohwertes. Dieser wird anschließend in einer Signalverarbeitung durch Filter aufbereitet. Der Ausgangswert der Signalverarbeitungseinheit dient als Eingang des Reglers. Des Weiteren erhält der Regler von der Steuereinheit den gewünschten Soll-Wert. Nun berechnet er mit den eingestellten Regelparametern daraus einen Stellwert für einen oder mehrere Motoren und steuert diese entsprechend an, so dass der Regelfehler minimiert wird. Der Regelfehler ist die Differenz aus Soll- und Ist-Wert.

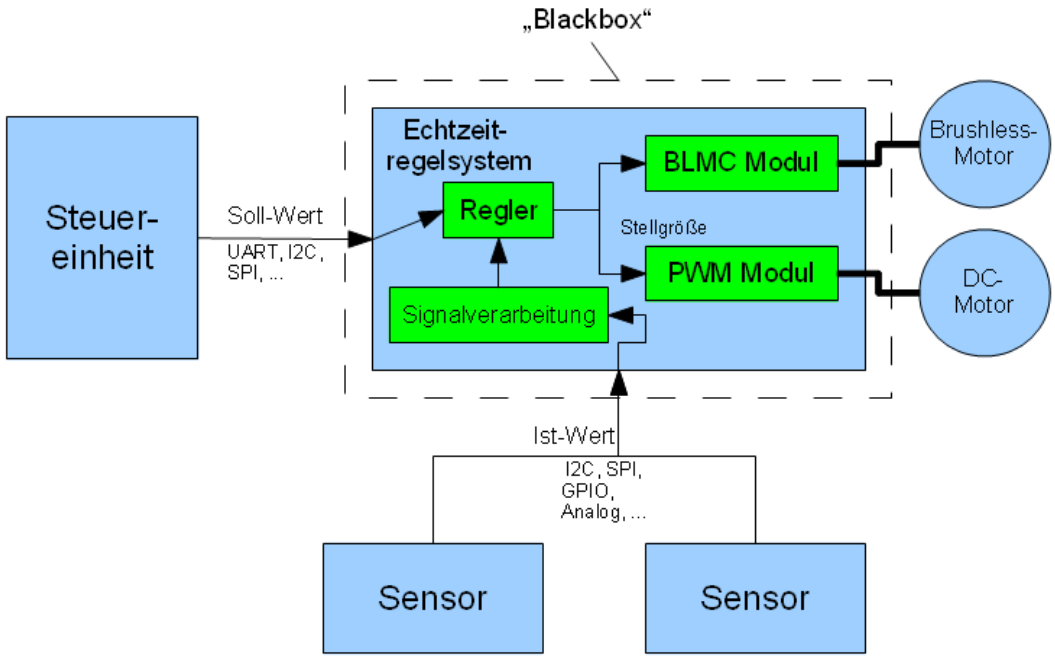

Abbildung 8: Übersicht Echtzeitregelsystem

3.2 Sensoren

Die Sensoren dienen der Erfassung des Ist-Wertes, wie z.B. die IMU für die Lage, der Drucksensor für die Höhe oder Abstandssensoren für die Position. Dafür werden an dem Echtzeitregelungssystem verschiedene Schnittstellen zur Verfügung gestellt, um eine Vielzahl verschiedener Sensoren und Sensortypen zu unterstützen. Dies sind zum einen Bussysteme wie I2C und SPI, wo mehrere Sensoren angeschlossen werden können, zum anderen digitale GPIO Pins und Analogeingänge zum direkten Anschluss entsprechender Sensoren. Damit werden beinahe alle für den Embedded Bereich gebräuchlichen Standartschnittstellen abgedeckt.

3.3 Signalverarbeitung

In der Signalverarbeitungseinheit werden die Rohdaten der Sensoren aufbereitet, um sie der Regeleinheit zur Verfügung zu stellen. Hier werden Störsignale wie z.B. ein Rauschen aus den Rohdaten durch geeignete Filteralgorithmen herausgefiltert. Dies kann z.B. ein gleitender Mittelwertfilter sein. Außerdem liefern Sensoren oft nicht direkt die gewünschte Messgröße. So muss z.B. bei einem Beschleunigungssensor die

Nulllage kalibriert werden. Ein Gyro liefert nur eine Winkelbeschleunigung. Um daraus die Lage zu erhalten, müssen diese Werte zeitlich integriert werden. Außerdem kann man die Daten von verschiedenen Sensoren z.B. mit einem Kalman-Filter fusionieren, um eine höhere Genauigkeit und Zuverlässigkeit zu erreichen. Die gefilterten, aufbereiteten und fusionierten Werte können anschließend an die Reglereinheit übergeben werden.

3.4 Steuereinheit

Die Stellwerte werden von der Steuereinheit vorgegeben. Diese ist entweder physikalisch im System integriert und spricht die Reglereinheit über eine API an oder ist über digitale Kommunikationsschnittstellen wie z.B. UART, I2C oder SPI angebunden. Die Steuereinheit kann den Soll-Wert dabei selbständig über einen Algorithmus bestimmen oder auf Benutzereingaben reagieren.

Beim Quadrocopter ist dies z.B. der über UART angebundene Funkfernsteuerungsempfänger. Es kann aber auch ein im Hauptprozessor befindliches Softwaremodul sein, welches z.B. mit Hilfe eines Positionserfassungssystems bestimmte Wegpunkte selbständig abfliegt.

3.5 Reglereinheit

Die Reglereinheit ist der Kern des Systems. Hier laufen Ist-Wert und Soll-Wert zusammen. Daraus wird anhand der Regelparameter und des Regelalgorithmus die Stellgröße berechnet und an die Motoransteuerung übergeben. Als Regelalgorithmus soll ein PID-Regler (Abb. 9 zum Einsatz kommen, es ist aber denkbar später auch andere Algorithmen zu implementieren.

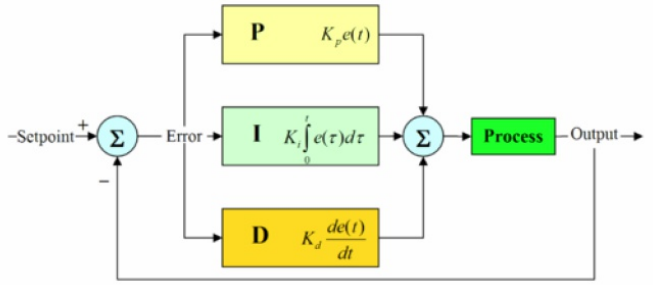

Abbildung 9: Schema eines PID-Reglers **[PID]**

Es soll auch möglich sein, mehrere Regler für verschiedene Regelgrößen parallel arbeiten zu lassen und anschließend die verschiedenen Stellgrößen in Abhängigkeit von-

einander durch einen auf den Anwendungsfall zugeschnittenen Algorithmus auf mehrere vorhandene Motoren zu verteilen.
Im einfachsten Fall besteht die Regeleinheit aus drei Lagereglern, die jeweils nur einen Freiheitsgrad regeln. Die wären dann zweimal ein Attituderegler für Roll und Pitch und ein Regler für die Yaw-Achse. Die Reglerausgänge werden dann geeignet überlagert und an die Motoransteuerung weitergegeben.
Dies kann dann beispielsweise die Lageregelung eines Quadrokopters mit vier oder mehr Propellern und der gleichen Anzahl an Motoren sein, wobei ein Motor mehr als einen Freiheitsgrad beeinflusst.

Die Reglereinheit bildet in Verbindung mit der Signalverarbeitung und der Motoransteuerung den Teil des Systems, den der Benutzer als „Blackbox“ sieht.

3.6 Motoransteuerung

Die Motoransteuerung wird aufgebaut aus BLMC-Modulen und/oder PWM-Modulen. Weiter Module sind auch möglich. Dadurch ist das System sehr modular gehalten, so dass für verschiedene Motortypen, verschiedene Motoransteuerungen zur Verfügung stehen. Das System enthält für jeden angesteuerten Motor ein Motoransteuerungsmodul. So können auch jeweils die gerade benötigten Module zur Ansteuerung der Motoren flexibel geladen werden. Dadurch sind verschiedene Systeme wie z.B. ein Quadrocopter mit vier Brushlessmotoren oder ein Helikopter mit einem BL-Motor und einem PWM-Motor möglich.
Die wesentlichen Teile der Motoransteuerung werden in programmierbarer Hardware implementiert werden, um den Hauptprozessor für Regelaufgaben freizuhalten. Die Anforderung an die Hardware wird dementsprechend formuliert.
Das Motoransteuerungsmodul nimmt die Stellgröße von der Reglereinheit entgegen und generiert die notwendigen Steuersignale, um den Motor entsprechend der Stellgröße entweder mit einer integrierten oder einer externen Endstufe anzusteuern.

Die Stellgröße ist in der Regel das PWM-Tastverhältnis oder die Spannung, mit welcher der Motor angesteuert wird. Bei Brushlessmotoren steht durch die notwendige Erfassung des Kommutierungszeitpunktes eine Möglichkeit zur Drehzahlmessung ohne weitere Komponenten zur Verfügung. Daher wird die Drehzahl durch das System erfasst. Die Drehzahlmessung wird in das Motoransteuerungsmodul für Brushlessmo-

toren in Form eines Drehzahlreglers integriert, der unabhängig vom Regler in der Reglereinheit arbeitet. So wird eine Regelung der Drehzahl ermöglicht.

3.7 Motoransteuerungsmodul BLMC

Im Folgenden soll nun ein Motoransteuerungsmodul für sensorlose Brushlessmotoren (BLMC) entworfen werden. Dieses soll als Besonderheit hauptsächlich mit konfigurierbarer Hardware funktionieren. Der Aufwand an externen, diskreten Bauteilen soll dabei minimiert werden. Der Prozessor, auf dem der Softwareteil läuft, wird damit möglichst entlastet und hat dadurch noch viel Rechenzeit für Signalverabreitung, Regelung und andere Aufgaben zur Verfügung.

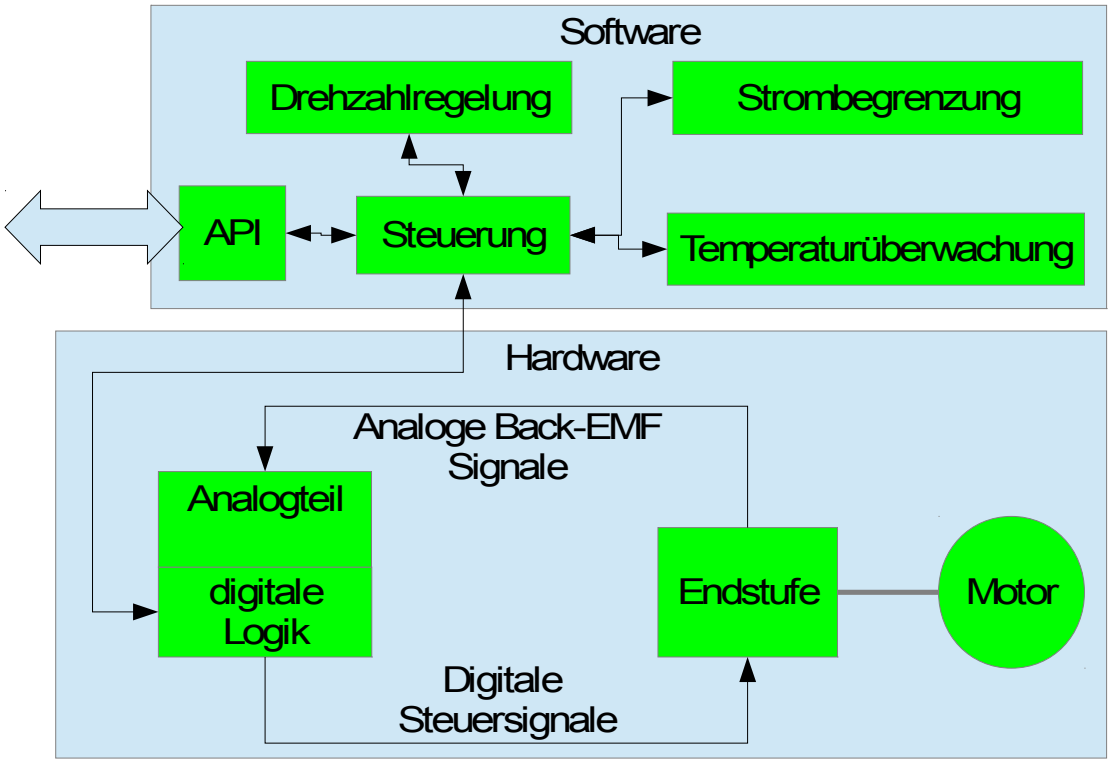

Abbildung 10: Übersicht BLMC Modul

Wie in Abb. 10 zu sehen besteht der BLMC im Wesentlichen aus den Teilen Hardwareendstufe, Analogteil, programmierbare digitale Logik und Softwaremodul. Die Hardwareendstufe mit dem Leistungsteil enthält die Leistungstransistoren, welche die Phasenströme schalten. Daran ist der Motor direkt angeschlossen. Der Analogteil filtert und wandelt den Spannungsverlauf der Motorphasen, damit dieser in digitale Signale umgesetzt werden kann. Dies dient dazu den Kommutierungszeitpunkt mittels Back-

EMF Auswertung zu erkennen. Die digitale Logik erzeugt mit dieser Information die für die Transistoren der Endstufe nötigen Steuersignale.

Das Softwaremodul stellt die Schnittstelle zwischen dem BLMC und dem Rest des Echtzeitregelsystesm dar. Es steuert die Hardware und überwacht das System auf fehlerhafte Zustände. Es übernimmt auch weiter Regelaufgaben wie Drehzahlregelung oder Strombegrenzung. Dem Nutzer wird ein komfortables API für den BLMC zur Verfügung gestellt. Darüber müssen lediglich Stellwerte übergeben werden. Um Dinge wie Starten des Motors bei Stillstand kümmert sich das Softwaremodul selbständig.

Ein wesentliches Ziel bei der Implementierung diese BLMC ist, die meisten Hardware-Funktionen in eine konfigurierbare Hardware zu integrieren, um die Anzahl an externen Bauteilen und somit Kosten, Größe und Fertigungsaufwand für die Platine möglichst gering zu halten. Da die Schaltung auch analoge Komponenten enthält, ist es von Vorteil, wenn diese sich zumindest zum Teil ebenfalls in die programmierbare Hardware integrieren lassen. Hierfür bietet sich beispielsweise der Psoc 5 Chip mit seinen konfigurierbaren Analogkomponenten an. Optimalerweise müssen dann nur noch wenige analoge Komponenten und keine digitalen Komponenten, bis auf die Leistungstransistoren, als diskrete Bauteile zur Verfügung gestellt werden.
Außerdem sollen viele Controllerfunkionen in Hardware anstatt Software ausgeführt werden, damit die CPU nur wenig belastet wird und für andere Aufgaben zu Verfügung steht. Allerdings steht dieser Bestrebung das Ziel entgegen, möglichst mehrere BLMC und evtl. weitere für den Betrieb notwendige Komponenten in den vorhandenen Funktiosnblöcken der eingesetzten Hardware unterzubringen. Also sollte rechenzeitintensiven und einfach in Hardware zu implementierenden Funktionen der Vorrang gegeben werden. Funktionen, die wenig Rechenzeit auf dem Prozessor beanspruchen, aber viele Hardwareresourcen verbrauchen, sollten dann besser im Softwaremodul des BLMC implementiert werden. Hier muss eine Abwägung getroffen werden.

Die vorhandene Endstufe kann bei veränderter Ansteuerung, was durch die programmierbare Hardware kein Problem darstellt, auch für PWM angesteuerte Gleichstrommotoren benutzt werden. So kann man mit der gleichen physikalischen Hardware sowohl Brushless- als auch Gleichstrommotoren ansteuern.

Als weiteres Feature besitzt der BLMC eine Strombegrenzung und Temperaturüberwachung. Überschreitet die Stromaufnahme einen vom System vorgegebenen Maximalwert, wird die Stellgröße schrittweise in der Höhe begrenzt bis die Stromaufnahme wieder im akzeptablen Bereich liegt. Wird die Maximaltemperatur der Platine überschritten, wird der maximale Stromwert, bei dem die Strombegrenzung anfängt zu greifen, heruntergesetzt, um die entstehende Verlustwärme zu verringern und somit die Komponenten nicht zu beschädigen. Eine komplette Abschaltung bei Kurzschluss ist so ebenfalls möglich.

4 Implementierung

4.1 Überblick

Das Ziel bei der Implementierung des vorliegenden Konzeptes ist es, die Hardwareplattform des aktuell vom Lehrstuhl entwickelten Quadrokopters zu ersetzen. Diese ist in Abb. 12 dargestellt. Sie besteht aus einem AVR32 Board für die Regelung. Daran sind via I2C eine IMU und vier Mikrokopter-BLMC angeschlossen.

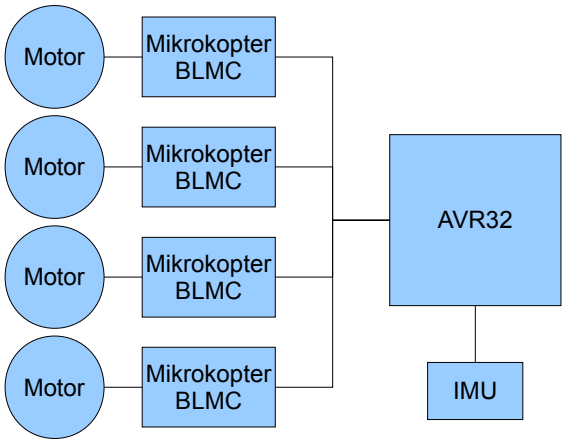

Abbildung 12: Bestehende Quadrokopterplattform mit AVR32 und Mikrokopter BLMC

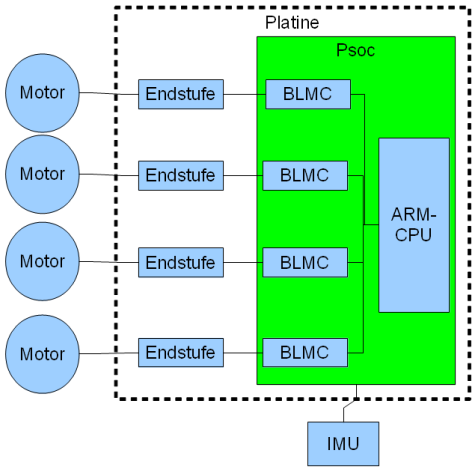

Abbildung 11: Quadrokopterplattform mit CPU und BLMC auf einer Platine

Die vier BLMC und die Regelung sollen nun auf einer Platine integriert werden (Abb. 12). Als Plattform dient dazu der Psoc5 Chip von Cypress. Dieser Chip verbindet einen ARM-Prozessor mit programmierbarer Digitalhardware. Ebenso enthält er durch programmierbare Verbindungen beliebig verschaltbare Analogkomponenten. Diese besondere Kombination aus programmierbarer Digital- und Analoghardware ermöglicht es, die BLMC bei minimalem Aufwand an externen Komponenten in Hardware zu implementieren. So bleibt die ARM-CPU frei für Regelungsaufgaben.

Im Zuge der Implementierung eines BLMC müssen nun die Komponenten entwickelt werden, die in der programmierbaren Hardware des Psoc-Chips implementiert werden sollen. Dazu muss ein Softwaremodul erstellt werden, welches diese Komponenten steuert und eine Schnittstelle zum Echtzeitregelsystem zur Verfügung stellt. Diese BLMC-Komponetnen werden dann viermal instantiiert. Ebenso wird eine eigene Platine für die externen Hardwarekomponenten, wie Leistungs-FETs und Signalfilte-

rung, entworfen und hergestellt. Diese Platine stellt Komponenten für die vier BLMC bereit und nimmt den Psoc-Chip auf.

Die vom Lehrstuhl im Augenblick auf der AVR32 Plattform benutzte Quadrocoptersoftware wird auf den ARM-Prozessor des Psoc portiert. So kann das System mit Hilfe einer über I2C angebundenen IMU die Lageregelung und Steuerung eines Quadrocopters inklusive Ansteuerung der Brushlessmotoren übernehmen. Das System ist dann auf einer einzigen Platine integriert und enthält nur selbst entwickelte Komponenten.

4.2 Programmierte BLMC-Komponenten im Psoc Chip

4.2.1 Der Psoc5 Chip und seine Entwicklungsumgebung

Der benutzte Psoc5 Chip der Firma Cypress besitzt die außergewöhnliche Eigenschaft sowohl eine 32 Bit ARM-CortexM3 CPU als auch programmierbare Digital- und Analogkomponenten in einem Chip zu verbinden. Sowohl die programmierbare Digitalhardware als auch die CPU takten mit bis zu 66 Mhz. Der CPU stehen 256 kb Flashspeicher und 64 kb SRAM zur Verfügung. Abb. 13 zeigt eine Übersicht über die im Psoc enthaltenen Komponenten.

Programmierte digitale Hardwarekomponenten werden in sogenannten „Universal Digital Blocks“ (UDB) implementiert. Der Psoc besitzt 24 davon. Ein UDB besteht aus zwei Programable Logic Devices (PLD), einem Datapath und Status- und Controlregistern. Zwischen diesen Komponenten in den UDBs, zwischen den UDBs selbst und den GPIO Pins gibt es flexible programmierbare Routingverbindungen. Die PLDs haben eine frei programmierbare AND- und ODER-Matrix sowie vier Makrozellen. Der Datapath enthält verschiedene 8 Bit Register und eine ALU, die auf diesen Registern arbeiten kann. Auf diese Register kann über ein Bussystem auch von der CPU aus zugegriffen werden. So kommuniziert das Programm auf der CPU mit der programmierten Hardware. Über diesen Bus werden auch die Status- und Controllregister gelesen bzw. geschrieben. So kann man von der CPU aus direkt routbare Signale auslesen, oder setzen. Die Idee hinter diesem Aufbau ist, dass die Datenverarbeitung von erstellten Hardwarekomponeten im Datapathteil mit ALU und Registern erfolgt. Die Steuerung dieser Verarbeitung kann dann in die PLDs implementiert werden.

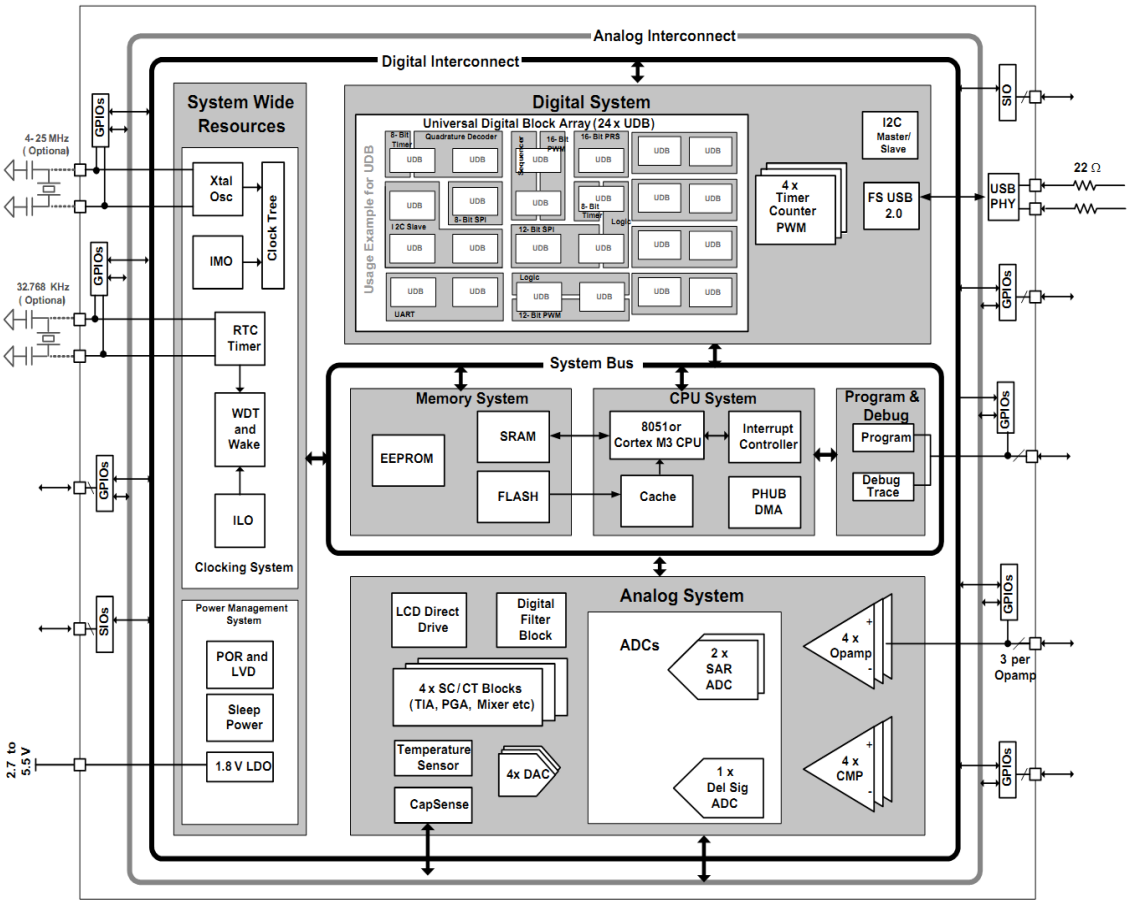

Abbildung 13: Schematischer Aufbau des Psoc5 Chips **[Psoc5]**

In der Entwicklungsumgebung Psoc Creator gibt es eine Vielzahl an vorgefertigten Komponenten, wie z.B. UART, SPI, Timer oder einfache Logik-Gatter, die sich mit einem grafischen Editor auf einem Schaltplan platzieren, und beliebig verbinden lassen. (siehe Abb. 15). Es ist auch möglich, eigene Komponenten in der Hardwarebeschreibungssprache Verilog zu definieren, allerdings ist der Zugriff auf den Datapath-Teil darüber kompliziert. Neben den frei programmierbaren UDBs gibt es auch noch mehrere Fixed-Function Blöcke wie z.B. Timercounter oder I2C, so dass bei der Verwendung dieser Komponenten keine UDBs belegt werden.

Der programmierbare Analogteil des Psoc enthält verschiedene Funktionseinheiten, wie beispielsweise ADC, DAC, Komparatoren oder Operationsverstärker. Diese können auch durch ein flexibel programmierbares analoges Routing untereinander und mit den GPIO-Pins verbunden werden. Diese analogen Einheiten werden genauso wie die digitalen im Psoc-Creator auf dem Schaltplan platziert und mit Analogsignalen verbunden. Eine Definition neuer, eigener Komponenten ist hier selbstverständlich nicht

möglich, man ist auf die in der Hardware physikalisch vorhandenen Einheiten beschränkt.

Die vorgefertigten Komponenten besitzen eine Software-API, über die sie von der CPU aus angesprochen werden können. Der Psoc-Creator generiert dabei für jede Instanz einer Komponente eine eigene C-Quellcodedatei, wobei der Instanzname der Komponente den Funktionsnamen als Prefix vorangestellt wird. Für eigene Komponenten kann man auch selbst eine API definieren.

Der Psoc-Creator übernimmt schließlich den kompletten Buildvorgang. Er synthetisiert die digitale Hardware, erstellt ein digitales und analoges Routing, generiert die APIs aller Komponenten und compiliert die Software. Am Schluss generiert er aus allem ein Hexfile, das in den Controller geflasht werden kann.
[Psoc5]

4.2.2 Überblick über die im Psoc implementierten Komponenten

Abb. 14 zeigt eine schematische Darstellung der im Hardwareteil des Psoc5 abgebildeten Komponenten des BLMC. Diese übernehmen im Normalbetrieb die Steuerung des BL-Motors autark, ohne Einwirkung von Software. Diese Steueraufgaben sind die elektronische Kommutierung des Motors und die PWM-Modulation des Ansteuersignals.

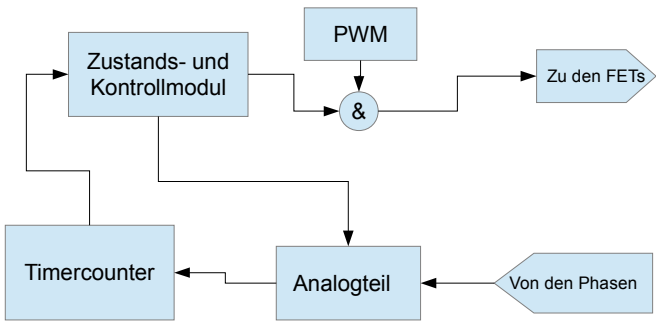

Abbildung 14: Schematische Übersicht der programmierten Hardware

Die Nulldurchgangserkennung im Analogteil erkennt den Nulldurchgang der aktuell floatenden Phase (bezogen auf den Sternpunkt des Motors). Dies ist der Zeitpunkt die elektronische Kommutierung auszulösen. Das Signal geht anschließend durch den Timercounter. Dieser misst zum einen die Zeit zwischen zwei aufeinanderfolgenden Kommutierungen, um die Drehzahl des Motors festzustellen. Zum anderen sorgt er da-

für, dass nach einer Kommutierung eine fest eingestellte Zeitspanne keine weitere Kommutierung erkannt wird. Dies verhindert Falscherkennung von Kommutierungen. Eine korrekt erkannte Kommutierung wird an das Zustandsspeicher- und Kontrollmodul weitergeleitet. Diese speichert den aktuellen Motoransteuerzyklus und schaltet bei einer Kommutierung in den nächsten Zyklus. In Abhängigkeit vom aktuellen Zyklus werden dann die Ansteuersignale für die LOW und HIGH FETs erzeugt. Anschließend werden die Ansteuersignal der HIGH FETs noch mit einem von der PWM-Einheit generierten PWM-Signal moduliert. So ist die Steuerung der am Motor anliegenden Spannung möglich.

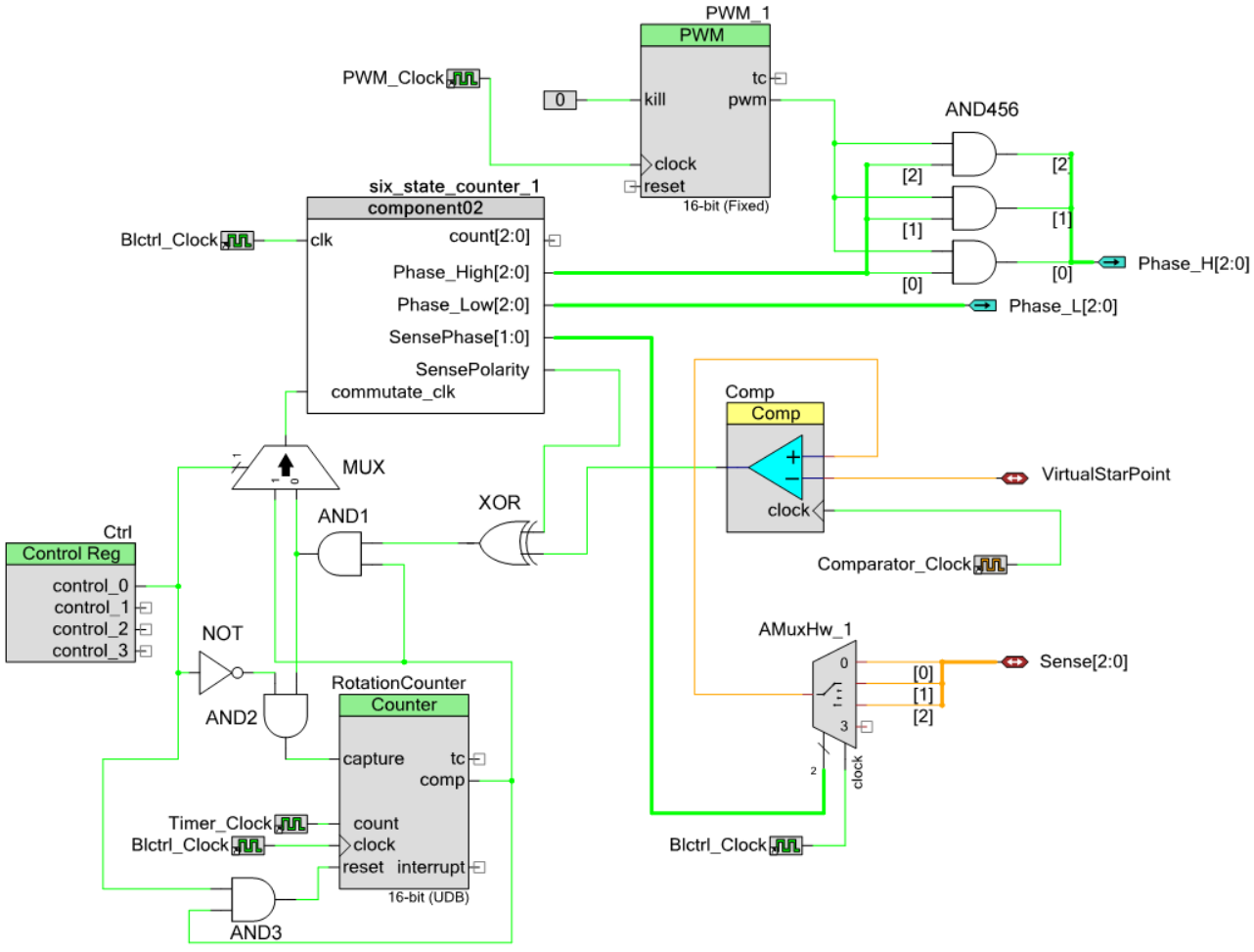

Abbildung 15: Detaillierter Schaltplan der im Psoc programmierten Digital- und Analogschaltungen

Alle Hardware-Funktionen werden durch das Programm, das in der CPU des Psoc läuft, gesteuert. Dieses Programm greift beispielsweise zum Starten des Motors in die

Kommutierungserkennung ein, oder stellt das gewünschte Tastverhältniss im PWM-Modul ein. Für den Normalbetrieb des Motors bei konstantem Tastverhältnis ist allerdings bis auf Überwachungsfunktionen kein Eingreifen der Software notwendig.

Da für den Quadrokopter später vier BLMC benötig werden, muss darauf geachtet werden, dass die digitalen und analogen Ressourcen des Psoc ausreichen, um vier BLMC und alle zusätzlich benötigten Peripheriekomponenten wie UARTs, I2C, und System-Timer zu instantiieren.

4.2.3 Analogteil

Der Analogteil der Schaltung ist in Abb. 15 an den orangen Verbindungen zu erkennen. Er ist dafür zuständig, die Spannung an den Motorphasen auszuwerten und den Kommutierungszeitpunkt zu erkennen. Dazu wird mit einem Analogkomparator die Spannung der Motorphase, welche im Augenblick nicht geschaltet ist, mit dem virtuellen Sternpunkt des Motors verglichen. Der virtuelle Sternpunkt ist dabei fest mit dem negativen Eingang verbunden. Die aktuell zu messende Phase wird über einen Analogmultiplexer auf den positiven Eingang des Komparators geschaltet. Dieser wird über einen 2-Bit breiten Bus vom Kontrollmodul gesteuert. An diesem Bus legt das Kontrollmodul die Nummer der aktuell zu messenden Phase binär codiert an. Der Multiplexer schaltet dann in einer Periode seines Arbeitstaktes entsprechend um. Am Ausgang des Komparators liegt nun ein Digitalsignal an. Diese besitzt, je nachdem ob die Spannung der gerade gemessenen Phase den virtuellen Sternpunkt mit positiver oder negativer Steigung kreuzt (Nulldurchgang), eine positive oder negative Flanke.

Der Analogkomparator des Psoc bietet die Möglichkeit, das Ausgangssignal mit einem Taktsignal zu synchronisieren. Das bedeutet, dass sich der Ausgang nur bei einer steigenden Flanke des Taktes ändert. Dies ist erforderlich, damit die nachfolgenden Schaltkreise keine Signalimpulse erhalten, die schneller als ihr Arbeitstakt sind.

Der Psoc5 bietet sich für die Realisierung des Analogteils besonders an, vor allem wenn mehrere BLMC in einem Chip realisiert werden sollen. Der Psoc enthält vier Analogkomparatoren, was maximal vier BLMC ermöglicht. Die Multiplexer werden durch die flexibel schaltbaren Analogverbindungen gebildet. Das ermöglicht für jeden Komparator einen eigenen Multiplexer. Die Möglichkeit vier Komparatoren mit jeweils vorgeschaltetem Analogmultiplexer anzubieten ist eine besondere Eigenschaft

des Psoc5. Würde man einen BLMC auf einer anderen, rein digitalen, programmierbaren Logikplattform, wie z.B. einem FPGA, realisieren wollen, müsste man je BLMC einen Analogkomparator und einen mindestens Drei-Wege Analogmultiplexer als externe Komponenten bereitstellen. Auch bei einer Software Implementierung findet man solch eine Anordnung von Analogkomponenten selten auf den gängigen Plattformen.

4.2.4 Timercounter

Der Timercounter (Block „Counter“ in Abb. 15) ist ein in den UDB-Blöcken des Psoc realisierter 16-bit Timercounter. Er zählt im Takt des an *count* liegenden Signals von 0 aufwärts. Er läuft nicht über, sondern bleibt stehen, wenn der Maximalwert erreicht ist. Hat das Signal an *capture* eine steigende Flanke, wird der aktuelle Zählerstand in das Captureregister gespeichert und der Zähler auf 0 zurückgesetzt. Der *reset* Eingang setzt den Zähler ebenso zurück. Der compare Ausgang gibt ein HIGH Signal aus, wenn der Zähler einen durch Software einstellbaren Wert übersteigt. Dieser Timercounter mit den umliegenden Logikgattern hat drei Aufgaben.

Die wichtigste Aufgabe ist hierbei sicherzustellen, dass nach einer erkannten Kommutierung für eine feste Zeitspanne keine weitere Kommutierung erkannt wird. Bei Tests der fertigen Platine mit vier BLMC, welche diese Blockierzeitspanne noch nicht implementiert hatten, zeigte sich, dass zwei der vier BLMC nur schlecht funktionieren. Es war häufig der Fall, dass nach einer erkannten Kommutierung direkt die nächste Kommutierung erkannt wurde, was zu einem unsauberen Lauf des Motors und erhöhtem Stromverbrauch führte. Der Grund dafür, dass das Problem nur zwei BLMC betraf, wird, nachdem Layoutfehler und fehlerhafte Lötstellen ausgeschlossen waren, in einer ungünstigen Platzierung von Leiterbahnen oder Bauteilen vermutet. Dadurch können sich Störsignale einkoppeln. Die genau Ursache konnte nicht ermittelt werden. Daher wurde die Blockierzeitspanne implementiert, um einen stabilen Betrieb aller BLMC zu ermöglichen.

Wenn eine Kommutierung erkannt wird, geht der Ausgang von XOR(Abb. 15) auf HIGH. Liegt die Kommutierung nicht in der Blockierzeitspanne, ist der *comp* Ausgang ebenfalls HIGH und somit der Ausgang von AND1. Da AND2 im Normalbetrieb das Signal durchlässt, wird ein capture im Timercounter ausgelöst und dieser zurückgesetzt. Jetzt ist auch die compare-Bedingung nicht mehr erfüllt und AND1 verhindert,

dass vorerst auf weitere (fehlerhaft) erkannte Kommutierungen reagiert wird. Überschreitet der Zähler nach einer bestimmten Zeit den eingestellten compare-Wert, geht *comp* wieder auf HIGH und es kann die nächste Kommutierung erkannt werden. Der eingestellte Wert muss dabei so groß sein, dass die meisten Fehlkommutierungen verhindert werden. Er darf allerdings auch den Motor in seiner Drehzahl nicht einschränken.

Die zweite Aufgabe ist die Drehzahlmessung. Der Zählerstand wird bei jeder Kommutierung im capture-Register gespeichert. Diese Register kann man nun per Software auslesen und erhält somit unter Berücksichtigung des Timertaktes von 4 MHz die Zeit-

$$n = \frac{1}{t * 6 * p} \qquad (1)$$

spanne t zwischen den letzten beiden Kommutierungen. Über die Formel(1) erhält man mit der Polzahl p des Motors die Drehzahl in Umdrehungen pro Sekunde. Die Drehzahl ist wichtig, um festzustellen, ob der Motor stehengeblieben ist, und um eine Drehzahlregelung zu realisieren.

Die dritte Aufgabe des Timercounters ist das Erzeugen eines fix vorgegebenen Kommutierungstaktes in der Anlaufphase. Dazu wird der capture-Eingang durch AND2 deaktiviert und der reset Eingang durch AND3 aktiviert. Erreicht der Zähler jetzt den eingestellten compare-Wert, geht *compare* und damit auch der reset-Eingang auf HIGH. Da der reset-Eingang synchron mit dem Arbeitstakt arbeitet, wird der Zähler erst im nächsten Arbeitstaktzyklus zurückgesetzt, womit *capture* auch wieder auf LOW geht. So entstehen periodische Impulse von einem Arbeitstakt Länge. Die Impulsfrequenz lässt sich über den compare-wert einstellen. Dieses manuell erzeugt Kommutierungssignal wird durch Umschalten des Multiplexers MUX direkt zum *commutate_clk* Eingang des Kontrollmoduls geleitet.

Der Zähltakt des Timercounters ist so zu wählen, dass bei der gegebenen Zählerauflösung von 16-Bit der Zähler bei der im normalen Betrieb maximal zu erwartenden Zeitspanne zwischen zwei Kommutierungen, also der minimalen Drehzahl, nicht überläuft. Für den Fall, dass die Minimaldrehzahl doch unterschritten wird, ist der Zähler so konfiguriert, dass er bei Erreichen des Maximums stehen bleibt und dieser Zustand erkannt werden kann. Ist er zu niedrig, verringert sich die Messauflösung bei der Drehzahlmessung.

4.2.5 Zustandsspeicher und Kontrollmodul

Das Kontrollmodul (Block „six_state_counter“ in Abb. 15) ist das zentrale Stück des BLMC. Es speichert den aktuellen Zustand, also den einen Zyklus aus den sechs Zyklen, in dem sich der Motor gerade befindet(siehe Abb. 2). In Abhängigkeit von diesem Zustand generiert das Modul die Ansteuersignale für die Leistungs-FETs, das Steuersignal zur Auswahl der für die Back-EMF Kommutierungserkennung zu messenden Phase und ein Signal, das festlegt, ob bei der Kommutierungserkennung auf ansteigende oder abfallende Spannung reagiert wird. Es wurde als eigene Komponente in Verilog implementiert.

Der Zustandsspeicher ist ein 3-Bit Binärzähler, der immer von 0 bis 5 zählt. Bei 5 bricht der Zähler automatisch um und beginn wieder bei 0. Soll eine Kommutierung erfolgen, muss auf dem Eingangssignal *commutate_clk* eine steigende Flanke erfolgen. Dann erhöht sich der Zähler. Der Psoc5 kann nicht direkt auf Flanken von Nicht-Taktsignalen reagieren. Daher wird mit jeder Periode des Arbeitstaktes *commutate_clk* in einem Latch gespeichert. So kann man den Zustand des Signals von der vorhergehenden Periode mit dem aktuellen vergleichen. Ist der vorhergehende Zustand LOW und der aktuelle HIGH, findet eine Kommutierung statt und der Zähler wird erhöht.

Zyklus	Zähler (3 Bit,dezimal)	Phase_High (3 Bit,binär)	Phase_Low (3 Bit,binär)	SensePhase (2 Bit,dezimal)	SensePolarity (1 Bit)
1	0	001	010	2	0
2	1	100	010	0	1
3	2	100	001	1	0
4	3	010	001	2	1
5	4	010	100	0	0
6	5	001	100	1	1

Tabelle 2: Ausgangssignale des Kontrollmoduls in den einzelnen Zyklen

Aus dem Zählerstand werden nun, wie in Tabelle 2 dargestellt, direkt die Kontrollsignale abgeleitet. Dies sind einmal zwei 3-Bit Busse für jeweils die HIGH und die LOW FETs. Dabei steht jedes Bit für den FET einer Phase. Ist es 1, wird dieser eingeschaltet, ist es 0, wird er ausgeschaltet. Weiterhin führt ein 2-Bit Bus zum Analogteil,

welcher dessen Multiplexer zur Auswahl der zu messenden Phase steuert. Weil es drei Phasen gibt, wird die auszuwählende Phase als 2-Bit Binärzahl im Bereich 0-2 dargestellt. Das letzte Signal legt fest, ob für die nächste zu erkennende Kommutierung auf eine ansteigende oder eine abfallende Flanke des Signals am Ausgang des Analogkomparator reagiert wird. Dazu wird dieses Signal mit dem Ausgangssignal des Analogkomparators mit dem XOR Gatter verknüpft. Soll die nächste Kommutierung bei ansteigender Spannung erfolgen, ist das *SensePolarity* LOW, und das Gatter invertiert das Ausgangssignal nicht. Das XOR hat also eine steigende Flanke, wenn das Ausgangssignal des Komparators eine steigende Flanke hat. Soll die nächste Kommutierung bei abfallender Spannung erfolgen, ist *SensePolarity* 1 und das Gatter invertiert das Ausgangssignal des Komparators. Das Ausgangssignal des XOR besitzt nun eine steigende Flanke, wenn das Signal des Komparators eine fallende besitzt. Das Ausgangssignal des XOR wird nun an den Timercounter weitergeleitet.

4.2.6 PWM

Um die am Motor anliegende Spannung und infolgedessen die Drehzahl zu beeinflussen, werden die Ansteuersignale der FETs mit einem PWM Signal moduliert. Dazu generiert das PWM-Modul kontinuierlich ein PWM-Signal. Das Modul wird in den Fixed-Function Timercounter Blöcken des Psoc implementiert. Diese haben eine Auflösung von bis zu 16-bit und es gibt vier davon. Und alle vier sind also durch die Instanziierung von vier BLMC alle belegt.

Die Frequenz des PWM-Signals muss wesentlich größer als die Kommutierungsfrequenz sein. Die Auflösung ist dann bei gegebener Frequenz durch die maximale Taktfrequenz des Psoc limitiert und sollte möglichst groß sein, um den Motor in feinen Schritten steuern zu können.

Das Tastverhältnis, welches die Stellgröße des Motors darstellt, wird durch Software gesetzt. Die PWM Frequenz beträgt bei anderen Systemen wie z.B. bei Mikrokopter 16 khz. Hier wird die doppelte Frequenz von 32 khz angestrebt. Dies verringert die Restwelligkeit der zu messenden Back-EMF Signale. Versuche haben gezeigt, dass dies dem unter Kapitel 4.2.4 beschriebenen Effekt der Fehlerkennungen von Kommutierungen positiv entgegenwirkt. Bei der maximal möglichen Taktfrequenz des Psoc von 66 Mhz ergibt sich bei 2048 Schritten eine PWM-Frequenz von rund 32,23 khz.

Das PWM-Tastverhältniss kann also mit einer Auflösung von 11 Bit eingestellt werden.

Da es ausreichend ist, entweder die LOW oder die HIGH FETs mit PWM zu modulieren und der jeweils andere FET während des aktuellen Zyklus eingeschaltet bleiben kann, werden nur die drei Steuersignale der HIGH FETs mit dem PWM-Signal moduliert. Dies geschieht, indem die vom Kontrollmodul kommenden Signale für die HIGH FETs durch die UND-Gatter AND456 (Abb. 15) jeweils mit dem PWM-Signal verknüpft werden. Somit wird der HIGH FET, der während der aktuellen Phase gerade angesteuert werden soll, anstatt mit einem kontinuierlichen Signal mit einem PWM Signal angesteuert. Die Signale für die LOW FETs werden direkt vom Kontrollmodul zu den Pins der FETs geleitet.

4.2.7 Steuerregister

Um die Funktionen des BLMC zu steuern, gibt es ein Steuerregister (Block „Control Reg“ in Abb. 15). Dieses wird softwareseitig mit 8-Bit angesprochen, allerdings wird hardwareseitig nur 1 Bit benutzt.

Diese legt fest, ob der Motor sich im Normalbetrieb mit Kommutierung durch Back-EMF Auswertung oder sich im Anlaufbetrieb mit fest vorgegebener Kommutierung befindet. Im Normalbetrieb ist dieses Bit 0. Damit schaltet der Multiplexer (MUX) das vom Komparator kommende Signal auf den *commutate_clk* des Kontrollmoduls. Außerdem wird der reset-Eingang des Timercounters durch AND3(Abb. 15) gesperrt. Das Gatter AND2 wird mit dem durch das NOT-Gatter invertierten Signal des Bits angesteuert und gibt so den capture-Eingang des Timercounters frei. Dies ist wichtig, damit der Timercounter die unter Kap. 4.2.4 beschriebenen verschiedenen Aufgaben im Anlauf- und Normalbetrieb ausführen kann. Im Anlaufbetrieb wird dieses Bit auf 1 gesetzt. Nun schaltet MUX das vom Timercounter kommende feste Kommutierungssignal zum Kontrollmodul durch. Außerdem verhalten sich AND2 und AND3 genau umgekehrt, so dass der capture-Eingang des Timercounters deaktiviert und der reset-Eingang freigegeben ist.

In der vorliegenden Implementierung wird der Motor ausgeschaltet, indem das PWM-Tastverhältniss auf 0 gesetzt wird, so dass die HIGH-FETs nicht mehr angesteuert werden. Über ein weiteres Bit im Steuerregister könnte man erreichen, dass alle

FETs sicher abgeschaltet werden können, wenn man diese mit allen Signalen der LOW- und HIGH-FETs mit UND-Gattern verknüpfen würde. Ebenfalls kann ein weiteres Steuerregister-Bit benutzt werden, um in dem Kontrollmodul eine Drehrichtungsumkehr zu implementieren.

4.3 Physikalische Hardware Endstufe

4.3.1 Leistungsteil

Der in Abb. 16 zu sehende Leistungsteil ist für die Ansteuerung der einzelnen Motorphasen zuständig. Für jede der drei Phasen gibt es solch ein Paar aus N-Kanal(LOW) und P-Kanal(HIGH) MOSFET. Der N-FET(Q3) schaltet die Phase gegen Masse(FND), der P-FET(Q2) gegen die Versorgungsspannung(VCC). Es ist dabei dar-

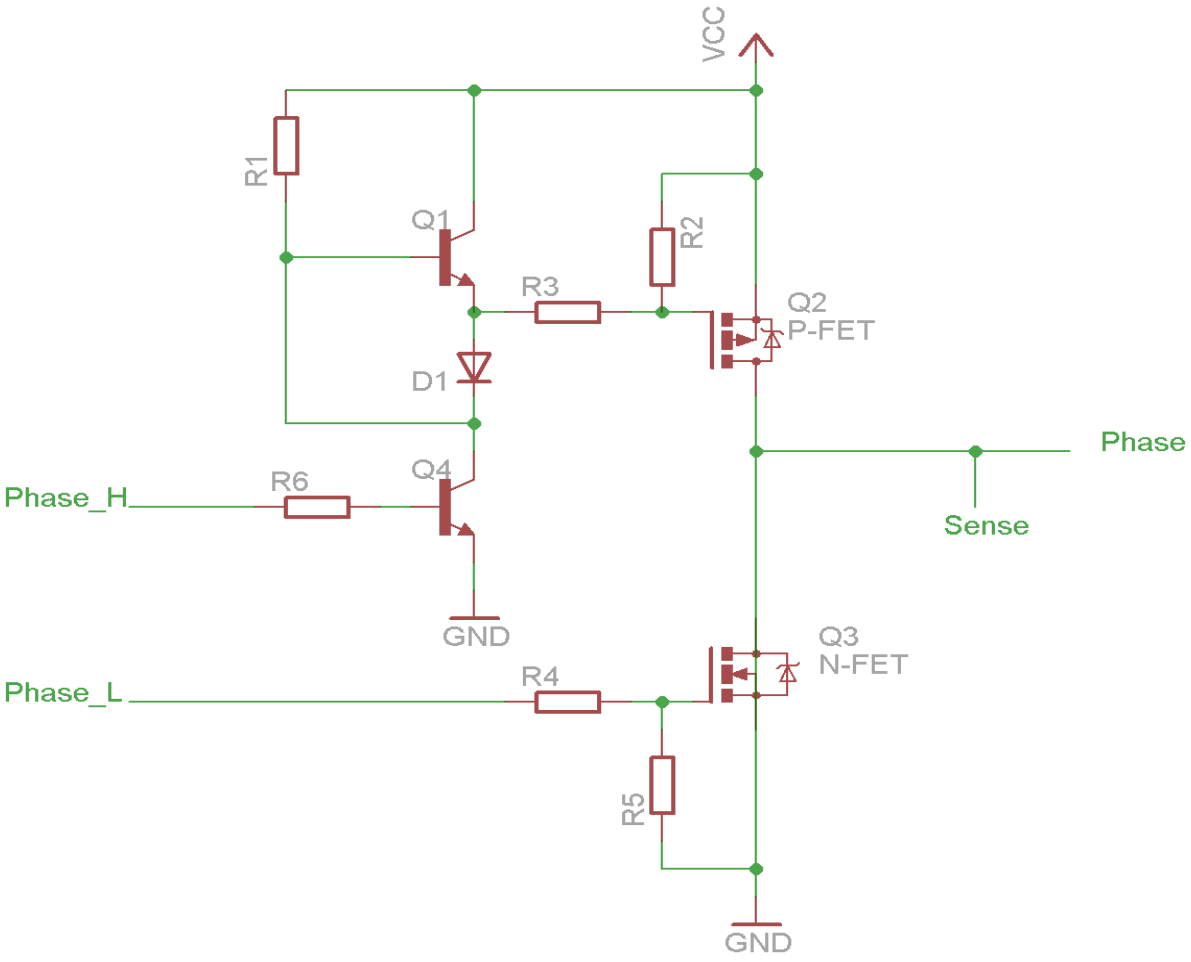

Abbildung 16: Schaltplan der Endstufe einer Motorphase

auf zu achten, dass niemals beide Transistoren einer Phase gleichzeitig eingeschaltet sind, da sonst ein Kurzschluss entsteht.

Die N-Kanal FETs sind Logiklevel-FETs, bei denen eine Spannung von 5V am Gate ausreicht, um sie komplett einzuschalten. Sie können deswegen direkt mit den Pins des Psoc verbunden werden. Es ist lediglich ein Widerstand von 100 Ohm (R4) in Serie geschaltet, um den Umschaltstrom zu begrenzen. Dadurch wird die Umschaltzeit zwar etwas langsamer, das ist aber bei den N-FETs weniger kritisch als bei den P-FETs, weil nur die P-FETs PWM moduliert werden und die N-FETs so deutlich weniger Schaltvorgänge ausführen. Ein Pulldown-Widerstand (R5) von 10 kOhm befindet sich am Gate des N-FET, so dass der Transistor sich nicht in einem undefinierten Zustand befindet, sondern sperrt, wenn der entsprechende Pin des Psoc hochohmig ist.

Da das Gate der P-FETs auf das Potenzial der Versorgungsspannung gebracht werden muss, damit der Transistor sperrt, können diese nicht einfach an einen Logikpin angeschlossen werden, sondern benötigen eine extra Ansteuerschaltung. Da die P-FETs durch die PWM Modulation viele Umschaltvorgänge durchführen, muss deren Umschaltung sehr schnell erfolgen, damit die entstehende Verlustleitung minimiert wird. Eine herkömmliche Open-Collector Schaltung mit dem Transistor Q4 mit Basisvorwiderstand R6 und R1 als Pullup würde zwar funktionieren, hätte aber das Problem, dass beim Ausschalten die Gatekapazität des FETs Q2 über den hochohmigen Widerstand R1 entladen werden würde, was zu einer längeren Umschaltzeit führt. Daher wird eine komplexere Schaltung benutzt. Diese ist dem Schaltplan der 2.0 Version des BLMC von Mikrokopter [Mikrokopter] entnommen.

Zum Einschalten wird der Pin an Phase_H auf High-Pegel(5V) geschaltet, so dass der Transistor Q4 einschaltet. Jetzt wird das Gate über R3,D1, und Q4 geladen. Der Strom ist hier nur durch R3 begrenzt. Dieser ist mit 10 Ohm sehr niederohmig und dient dazu Schwingungen zu dämpfen. Das Einschalten des FETs geht also sehr schnell. Da der Emitter von Q1 wegen des Spannungsabfalls an D1 auf höherem Potenzial als die Basis von Q1 liegt, sperrt Q1 sicher.

Zum Ausschalten wird Phase_H auf LOW-Pegel(0V) gelegt und Q4 sperrt. Jetzt kann der Strom über R1 durch die Basis von Q1 und R3 in das Gate fließen. In Q1 fließt nun ein Basisstrom. Somit kann nun ein um den Verstärkungsfaktor von Q1 grö-

ßerer, nicht durch das hochohmige R1 begrenzter Strom, durch den Collector von Q1 fließen und das Gate schnell aufladen. R2 ist analog zu R5 ein Pullupwiderstand.

Bei der Auswahl der FETs muss man vor allem auf den Widerstand im eingeschalteten Zustand achten. Dieser muss klein genug sein, dass die bei dem zu erwartendem Motorstrom entstehende Verlustleistung aller FETs an die Umgebung abgeführt werden kann. Ein kleinerer Widerstand bedeutet in der Regel einen höheren Preis der FETs und es muss hier eine Abwägung getroffen werden.
[Mikrokopter]

4.3.2 Back-EMF Aufbereitung

Damit der Psoc das Back-EMF Signal der Phasen auswerten kann, muss dieses erst aufbereitet werden. Dies geschieht mit der in Abb. 17 dargestellten Schaltung. Dazu muss zum einen die Spannung auf einen Wert reduziert werden, den der Psoc verträgt (max. 5,5 V), zum anderen muss die PWM-Frequenz herausgefiltert werden.

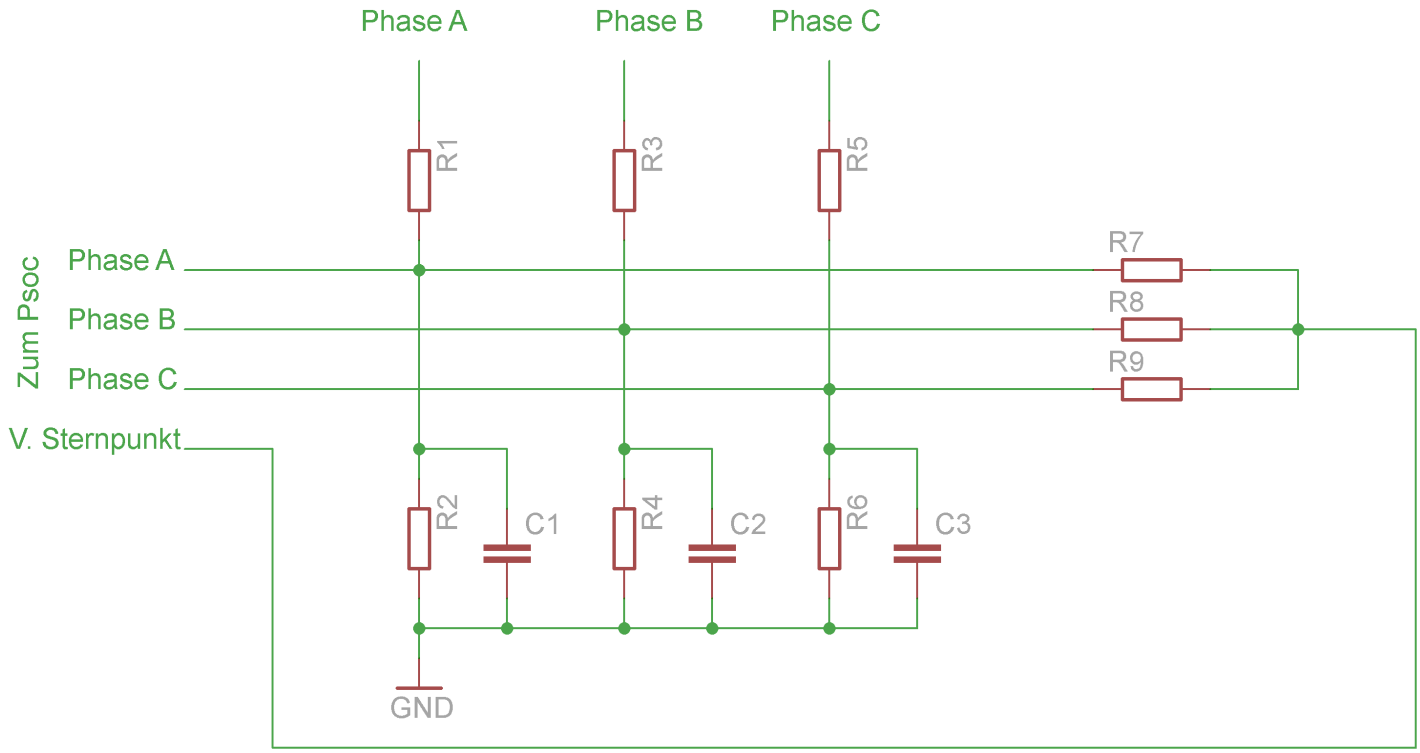

Abbildung 17: Schaltplan der Back-EMF Filterschaltung für alle drei Phasen

Das Reduzieren der Spannung geschieht mit einem Spannungsteiler. Für Phase A besteht dieser aus den Widerständen R1 und R2. Beide haben einen Wert von 4.7 kOhm, die Spannung wird also halbiert. Die maximale Betriebsspannung des Motors beträgt also in diesem Fall 11V.
Die PWM-Frequenz wird mit einem RC-Tiefpassfilter herausgefiltert und das Signal geglättet. Für Phase A wird dieser Filter aus dem Widerstand R1, der gleichzeitig Teil des Spannungsteilers ist, und dem Kondensator C1 gebildet. R2 beeinflusst allerdings ebenfalls das Verhalten des Filters. Die Grenzfrequenz diese Filters lässt sich mit der Formel (2) [Atmel] ausrechnen.

$$f = \frac{R_1 + R_2}{2\pi R_1 R_2 C} = \frac{4{,}7\,kOhm + 4{,}7\,kOhm}{2\pi\, 4{,}7\,kOhm\, 4{,}7\,kOhm\, 100\text{nF}} = 677\,Hz \qquad (2)$$

Sie liegt mit 677 Hz deutlich unter der PWM Frequenz von 32 kHz, die damit gut ausgefiltert wird.
Eine weitere Aufgabe dieser Schaltung ist es, dem Psoc den Sternpunkt des Motors zur Verfügung zu stellen. Mit diesem Sternpunkt wird die aktuelle Phase verglichen. Der echte Sternpunkt im Motor wird meistens nicht als eigener Anschluss herausgeführt. Deshalb wird im BLMC der Sternpunkt als sogenannter virtueller Sternpunkt nachgebildet. Dies geschieht mit den drei gleichen Widerständen R7, R8 und R9. Ihr Wert beträgt 4,7 kOhm.
[Atmel] [Mikrokopter] [Mikrocontroller]

4.3.3 Strom- und Temperaturüberwachung

Um eine Überlastung zu verhindern und den Nutzer zu informieren, soll der aktuelle Stromfluss eines jeden BLMC und die Temperatur auf der Platine erfasst werden.

Der Stromfluss wird dabei über eine Shuntwiderstand gemessen, der sich zwischen der Masse der FETs eines BLMC und der allgemeinen Masse befindet. Die Spannung, die über diesen Shunt abfällt, wird mit einem RC-Tiefpassfilter geglättet. Im Psoc wird sie mit einem Verstärker in den Messbereich des ADC gebracht und mit diesem digitalisiert. Nun kann man aus ihr, mit der Kenntnis des Widerstandswertes des Shunts, den fließenden Strom berechnen. Bei der Dimensionierung des Shunts muss darauf geachtet werden, dass einerseits der Spannungsabfall groß genug ist, um gemessen werden zu können, andererseits darf die Verlustleistung nicht zu groß werden. Der Shunt ist

ein 4mm breites und 77mm langes Stück Leiterbahn mit einer Kupferdicke von 35µm. Dies ergibt einen Widerstandswert von 9,8 mOhm.

Für die Temperaturmessung sind auf der Platine zwei NTC-Widerstände platziert. Diese sind jeweils mit einem 680kOhm Widerstand in Serie geschaltet, dass sich ein Spannungsteiler zwischen 5V und Masse ergibt. Diese Spannung wird nun auch über einen ADC im Psoc digitalisiert und daraus die Temperatur errechnet.

Nach der Fertigung der Platine hat sich, wie unter Kapitel 4.5.3 beschrieben, allerdings herausgestellt, dass die Wahl der Pins am Psoc für die analogen Signale ungünstig war. Bei der gegebenen Pinbelegung war es nicht möglich, alle analogen Signale gleichzeitig an ihre entsprechenden Komponenten im Posc zu routen. Es ist durch eine nachträgliche Änderung an der Platine noch möglich gewesen, alle für den Betrieb der vier BLMC notwendigen Analogsignale zu routen. Alle anderen analogen Funktionen wie Temperatur- und Strommessung mussten jedoch aufgegeben werden.

4.3.4 Testaufbau im Steckbrett

Vor der Fertigung der Platine sollte die grundlegende Funktion der Schaltung verifiziert werden. Dazu wurde die Schaltung für einen BLMC mit bedrahteten Bauteilen in einem Steckbrett aufgebaut (Abb. 18). Die Schaltung wurde mit eine Psoc5 Entwicklungsboard verbunden. In dem darauf enthaltenen Psoc-Chip wurde ein BLMC instantiiert und mit einem einfachen Softwaremodul versehen, so dass sich der Motor über ein serielles Terminal steuern ließ. Der Motor wurde nun mit verschiedenen Stellwerten angesteuert. Dabei wurde die Korrektheit der Ansteuersignale, sowie der Spannungsverlauf an den Motorphasen mit einem Oszilloskop untersucht. Das Ergebnis war, dass die Schaltung wie erwartet funktionierte. Um festzustellen, ob die Kommutierungen sauber in kontinuierlichen Abständen erfolgen, wurde der Ausgang des Komparators im Psoc auf eine Digitalausgang geroutet, um das Signal am Oszilloskop sehen zu können. Dadurch war zu sehen, dass der BLMC zwar bei hohen, für den Betrieb im Quadrokopter relevanten Drehzahlen sauber arbeitete, aber bei sehr niedrigen Drehzahlen falsche Kommutierungen erkannt wurden. Das Signal war dann sehr unregelmäßig. Die Blockierzeitspanne des Timercoutners war zu diesem Zeitpunkt noch nicht implementiert. Das Problem wurde auf die langen Kabelverbindungen zwischen den Bauteilen und das lange Flachbandkabel zwischen Steckbrett und Psoc-Board zu-

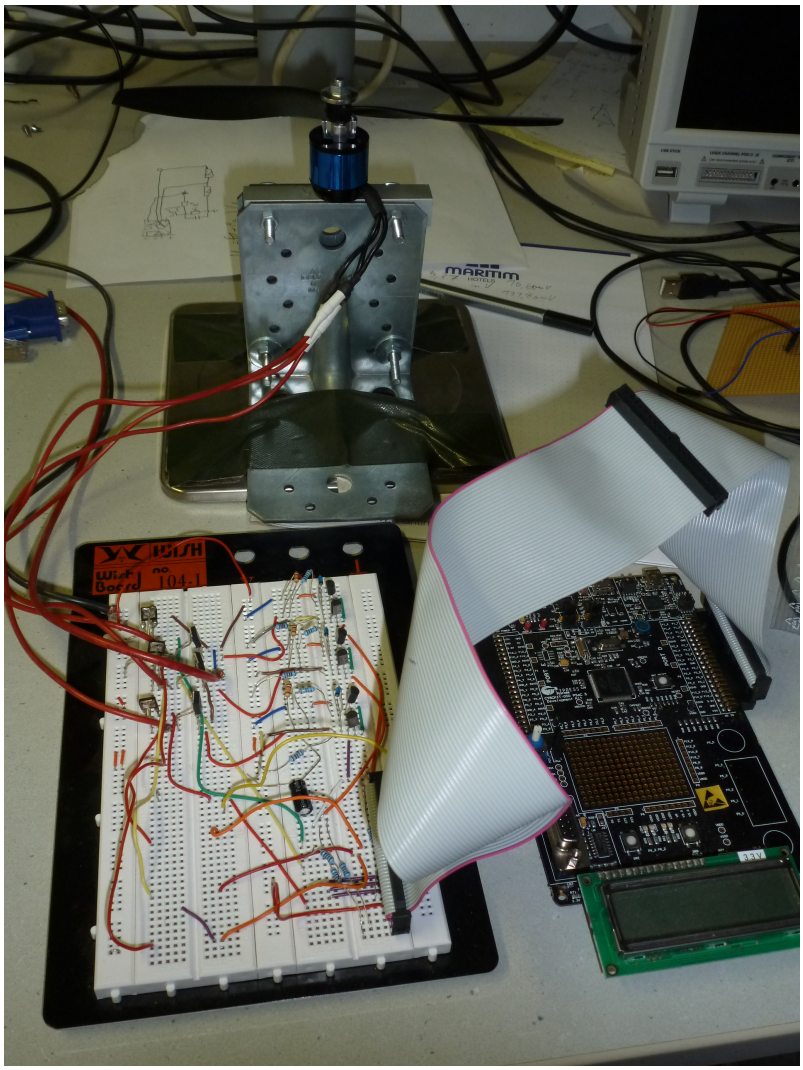

Abbildung 18: Testaufbau eines BLMC

rückgeführt. Da es für den Betrieb im Quadrokopter, wo hohe Drehzahlen erforderlich sind, nicht hinderlich war und erwartet wurde, dass es mit der fertigen Platine besser wird, wurden vorerst keine weiteren Maßnahmen ergriffen.

4.4 Software

4.4.1 Überblick

Der Software-Teil des BLMC ist die Schnittstelle zwischen dem in Software implementierten Regler und der BLMC Hardware. Er stellt eine komfortable API zur möglichst einfachen Benutzung bereit. Ebenso werden hier Funktionen, die nicht in Hardware implementiert werden sollen oder können, implementiert. Dies sind vor allem Überwachungsfunktionen, wie die Strom- und Temperaturüberwachung, die Überprü-

fung der Drehzahl, um festzustellen ob der Motor stehen geblieben ist, und die Steuerung des Anlaufvorgangs. Weiterhin ist es möglich hier noch weiter Features, wie beispielsweise eine Drehzahlregelung, zu implementieren.

Diese Funktionen werden in einer Funktion implementiert, die vom Code des Benutzers periodisch, in diesem Fall alle 1 ms, aufgerufen werden muss. Dies geschieht hier direkt in der Interrupt-Routine des Timers, die jede Millisekunde aufgerufen wird. Wird der Motorcontroller in einem anderen System eingesetzt, kann dies auch aus einer anderen Interrupt-Routine oder aus dem normalen Code heraus erfolgen, solange das 1ms Intervall eingehalten wird.

Diese Funktion ist Teil eines zentralen Softwaremoduls, das für alle im System vorhandenen Motorcontroller zuständig ist. Jede BLMC Komponente hat dann nochmal ihr eigenes Softwaremodul, welches den Verbindungscode zu den Hardwareregistern bereitstellt.

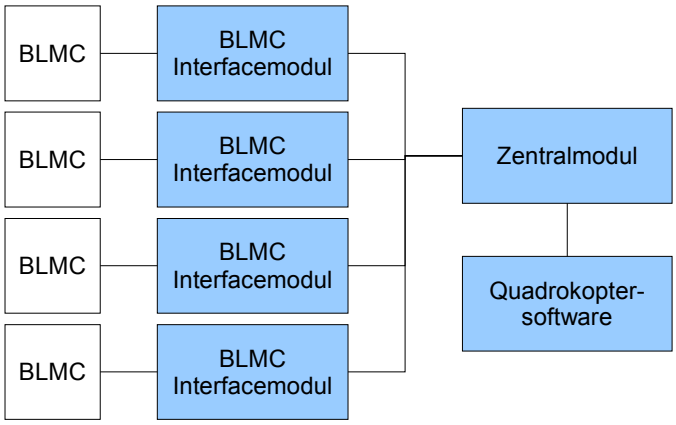

Abbildung 19: Schema der verschiedene Softwaremodule

4.4.2 Softwaremodul der einzelnen BLMC Komponente

Das Softwaremodul eines einzelnen BLMC wird im Psoc Creator als API zur BLMC Hardwarekomponente implementiert. Während des Compilierens wird somit für jede BLMC Instanz ein Modul mit individuellen, vom Instanznamen abgeleiteten Funktionsnamen generiert. Da nun jeder BLMC individuelle Funktionsnamen hat, ist es erst einmal nicht möglich, allgemeinen Code für eine beliebige Anzahl von Motorcontrollern zu schreiben. Deswegen gibt es eine Init-Funktion, die zum einen alle Unterkomponenten initialisiert und zum anderen eine C-Struktur mit Funktionspointern aller

vom zentralen Modul benötigten Funktionen zurückgibt. Die Strukturen aller Motorcontroller werden im Zentralmodul in einem Array gespeichert. Diese Architektur ermöglicht es auch auf einfache Weise Motorcontroller für andere Motortypen zu integrieren.

Über die zur Verfügung gestellten Funktionen kann das PWM-Tastverhältniss, der compare-Wert des Timercounters und der Betriebsmodus eingestellt werden. Die ersten beiden Werte werden dabei einfach an die Funktionen der Unterkomponenten weitergereicht. Die Betriebsmodusfunktion setzt je nach übergebener Konstante die Bits im Steuerregister. Außerdem kann der Wert des capture-Registers des Timercounters gelesen werden.

4.4.3 Zentrales Motorsteuermodul

Das Zentralmodul enthält wie bereits erwähnt ein Array von Strukturen mit Funktionspointern zu den Kontrollfunktionen der einzelnen Motorcontroller. Die Strukturen enthalten auch Variablen, um den Zustand und den Typ der Controller zu speichern. Es ist denkbar, nicht nur BLMC zu verwenden sondern beispielsweise auch PWM-Motorcontroller. In der aktuellen Implementierung wurden aber nur BLMC berücksichtigt. Das Array wird in der Initialisierungsfunktion des Zentralmoduls erstellt. Hier wird jede Init-Funktion von jedem vorhandenen Motorcontroller explizit aufgerufen, ab dann werden die Motorcontroller nur noch über die in den Strukturen gespeicherten Funktionspointer angesprochen. Dies ermöglicht den restliche Code allgemein zu halten.

Das Modul kennt für BLMC nun drei Zustände:

- Aus
- Anlaufbetrieb mit manueller, fester Kommutierung
- Normalbetrieb mit automatischer Kommutierungserkennung

Wird mit der *Motor_setPWM()* Funktion das PWM Tastverhältnis auf 0 gesetzt, geht der Motor grundsätzlich in den Aus-Zustand. Wird ein PWM-Wert größer 0 gesetzt und der Motor befindet sich im Normalbetrieb, wird der Wert einfach übernommen. Befindet sich der Motor dabei im Aus-Zustand, wird er in den Anlaufbetrieb versetzt.

Die regelmäßig in konstantem Zeitintervall aufgerufene *Motor_Update()* Funktion durchläuft in einer Schleife alle vorhandene Motoren. Ist ein Motor ausgeschaltet, werden für diesen weder Stellwerte übernommen, noch Überwachungsfunktionen durchgeführt.

Befindet sich ein Motor im Normalbetrieb, muss die Update-Funktion überprüfen, ob dieser stehengeblieben ist. Dazu wird der Wert des Timercounter ausgelesen, den dieser im letzten Kommutierungszyklus erreicht hat (capture-Register). Aus diesem wird ein gleitender Mittelwert über die letzten 10 Messwerte gebildet. Erreicht dieser den maximalen Wert des Counters, wurde keine Kommutierung mehr erkannt und der Motor steht. Es kommt aber auch häufig vor, dass das System unkontrolliert schwingt, weil bei stehendem Motor die durch eine Kommutierung ausgelöste Umschaltung der Phasen sofort die nächste Kommutierung auslöst und der Messwert somit ein Minimum, nämlich die eingestellte Blockierzeitspanne, erreicht. Es muss also auch auf Erreichen dieses Minimums geprüft werden. Wird erkannt, dass der Motor steht, wird dieser in den Anlauf-Zustand versetzt.

Befindet sich ein Motor im Anlauf-Zustand, wird mit jedem Aufruf der Update Funktion ein extra Zäher heruntergezählt. Dieser Zähler steuert den Ablauf des Anlaufvorgangs. Zu Beginn wird der BLMC auf manuelle Kommutierung eingestellt. Nun erzeugt der Timercounter ein fest vorgegebenes Kommutierungssignal, dessen Frequenz sich über den compare-Wert einstellen lässt. Nun wird ein niedriger PWM-Wert und eine langsame Kommutierung eingestellt. Dies wird die nächsten 300 ms beibehalten, damit sich der Motor auf das vorgegebene Drehfeld einstellen kann. Dann wird in Abständen von 100 ms der PWM-Wert linear und die Kommutierungsfrequenz exponentiell erhöht. Dieses Verhalten ist dem Mikrokopter-BLMC [Mikrokopter] nachempfunden. Die Werte wurden experimentell angepasst. Wichtig hierbei ist die PWM-Werte nicht zu niedrig zu wählen, damit der Motor sicher anläuft. Zu hohe PWM-Werte bedeuten allerdings hohe Stromspitzen und Erwärmung der Bauteile. Nach insgesamt 1s ist der Anlaufvorgang beendent und es wird in den Normalbetrieb mit automatischer Kommutierung umgeschaltet. Dazu wird auch das ursprünglich vom Nutzer gewünschte PWM-Verhältniss eingestellt. Der compare-Wert des Timercounter muss jetzt ebenso auf den Wert der Blockierzeitspanne eingestellt werden.

Die Strom- und Temperaturüberwachung würde ebenfalls in der Update-Funktion implementiert. Dazu würde die über dem Shunt des entsprechenden BLMC abfallende Spannung gemessen und in einen Stromwert umgerechnet. Dazu ist der genaue Widerstandswert des jeweiligen Shunts nötig, welcher durch eine einmalige Messung im Voraus bestimmt wird. Wird eine festgelegte maximale Stromstärke überschritten, kann man den Motor entweder direkt abschalten oder den PMW-Wert begrenzen. Die Messung der Temperatur kann mit einer deutlich geringeren Rate erfolgen als die Strommessung, weil sich die Temperatur nur sehr träge ändert. Wird die Maximaltemperatur der Platine überschritten, muss der maximal zulässige Strom begrenzt werden. Diese beiden Funktionen konnten aber wegen des in Kapitel 4.5.3 bereits angesprochenen Layoutfehlers nicht mehr implementiert werden.

Denkbar wäre auch an dieser Stelle mit einem Software-PID-Regler eine Drehzahlregelung zu implementieren.

4.4.4 API des BLMC

Im Folgenden soll auf die dem Nutzer zur Verfügung stehenden API Funktionen und Konfigurationsmöglichkeiten eingegangen werden.

```
#define MOTORMODULES_COUNT 4
BLMC_Motor_Module motormodules[MOTORMODULES_COUNT];

void Motor_Init()
{
      motormodules[0]=blctrl_1_Init();
      motormodules[1]=blctrl_2_Init();
      motormodules[2]=blctrl_3_Init();
      motormodules[3]=blctrl_4_Init();
}
```

Dies ist die Initialisierungsfunktion für alle BLMC und muss daher immer vor dem Aufruf anderer Funktionen aufgerufen werden. Gleichzeitig werden hier die im Psoc vorhandenen Instanzen der BLMC-Komponente dem Zentralmodul bekannt gemacht. Das Define `MOTORMODULES_COUNT` legt die Anzahl der vorhandenen Motorcontroller fest. In der Init-Funktion wird anschließend jedem Element des Arrays `motormodules` der Rückgabewert der Init-Funktion der entsprechenden BLMC-Komponente zugewiesen.

```
void Motor_setPWM(int motor,int pwm);
```

Diese Funktion weist dem Motor mit der ID `motor` den PWM-Wert zu. Die ID ist dabei im Bereich von 0 bis `MOTORMODULES_COUNT-1` Der PWM-Wert ist im Bereich 0 bis 2047. Das Anlaufen des Motors aus dem Stillstand erfolgt im Bedarfsfall automatisch.

```
int Motor_getRPM(int motor);
```

Diese Funktion gibt die Drehzahl des entsprechenden Motors in Umdrehungen pro Minute zurück. Dabei wird von einem einpoligen Motor ausgegangen. Hat der Motor mehr Pole muss der Wert durch die Anzahl der Pole dividiert werden.

```
void Motor_Update();
```

Diese Funktion muss regelmäßig in konstanten Intervallen von 1ms aufgerufen werden. Wird sie das nicht, drehen sich bereits laufende Motoren zwar weiter, aber alle Überwachungsfunktionen sowie die Anlauffunktion funktionieren dann nicht mehr. Wird das Aufrufintervall von 1 ms nicht eingehalten, wird das Zeitmanagement der Anlauffunktion gestört.

4.5 Platine

4.5.1 Layoutentwurf

Zur Aufnahme aller elektronischen Bauteile und des Psoc Chips wurde eine individuelle Platine hergestellt. Das Layout, also der Verlauf der Leiterbahnen, wurde mit der CAD Software DesignSpark für Platinendesign entworfen. Im Anschluss wurde eine Firma beauftragt, aus den Daten des Programms eine Platine herzustellen. Diese wurde dann in Eigenarbeit bestückt.

Die Platine enthält die Endstufen und Back-EMF Filter für vier BLMC. Weiterhin gibt es vier Status LEDS, einen Reset-Knopf und Buchsenleisten zur Aufnahme des Psoc Prozessormoduls. Als Anschlüsse sind zwei UARTs Schnittstellen und ein I2C-Bus vorhanden. Die übrigen noch nicht benutzten Pins des Psoc werden zur späteren

Verwendung ebenfalls auf Stiftleisten herausgeführt. Ebenso gibt es Anschlüsse für die 5V und 3,3V Versorgungsspannungen des Psoc, die durch extra Spannungsregler bereitgestellt werden müssen.

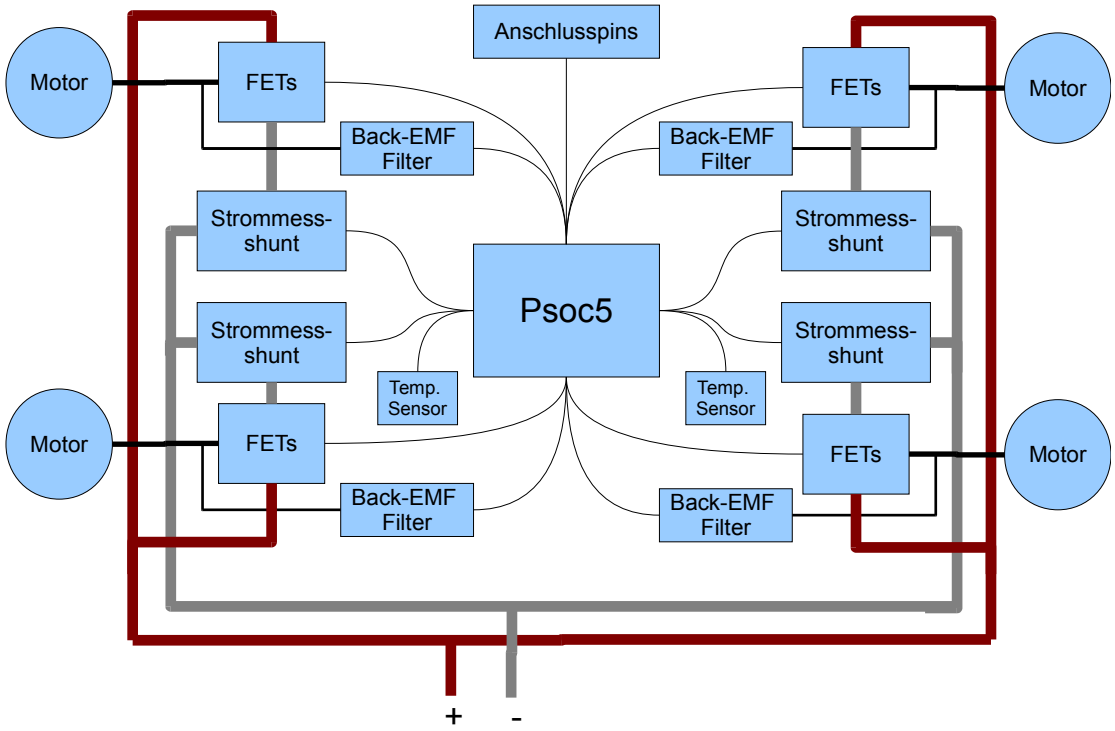

Abbildung 20: Schematische Übersicht der Verdrahtung der Komponenten auf der Platine

Die Platine soll auch den Strom vom Anschlusspunkt des Akkus zu den einzelnen BLMC verteilen. Die bisher im Quadrokopter benutzten Mikrokopter BLMC sind für einen Dauerstrom von 10A spezifiziert. Um eine Verbesserung zu erreichen, wurde für die Psoc-BLMC die Vorgabe von 20A Dauerstrom pro Motorcontroller gemacht. Daraus ergibt sich ein maximaler Gesamtstrom von 80A. Die Leiterbahn für diesen Strom bräuchte nach den gängigen Tabellen eine Breite von mindestens 4cm. Um die Platine von den Abmessungen daher so klein zu halten, dass sie in den Quadrokopter passt, wurde die Platine vierlagig ausgeführt. Die beiden inneren Lagen dienen dabei nur der Verteilung von Masse und Versorgungsspannung der Motoren. Die beiden äußeren Lagen nehmen die Bauteile auf und führen die Signale. Ein höherer Gesamtstrom als 80A wurde selbst bei diesem Aufbau der Platine als nicht beherrschbar eingeschätzt. Layout der Vorderseite ist in Abb. 21 in rot zu sehen. Die Rückseite ist türkis dargestellt.

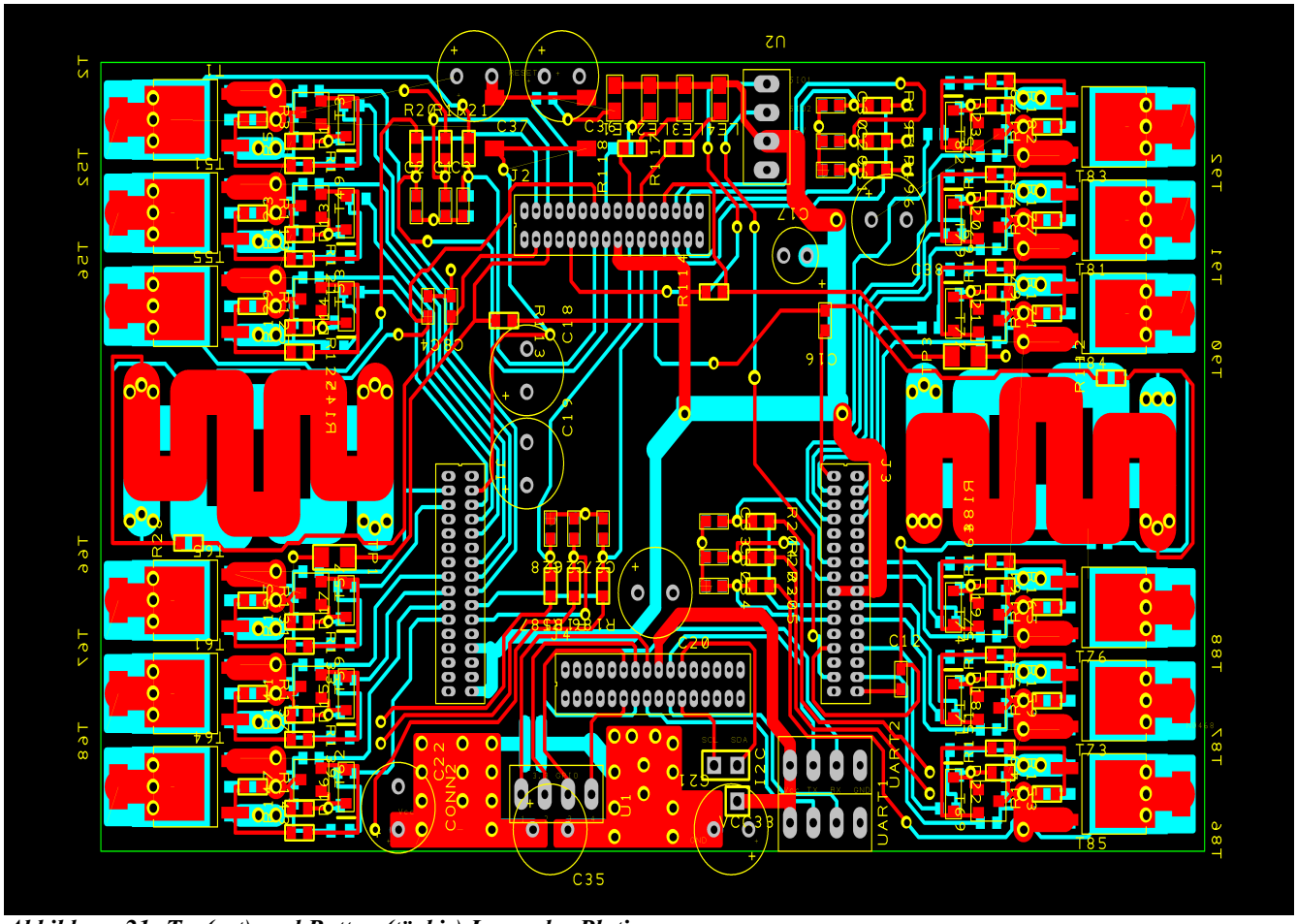

Abbildung 21: Top(rot) und Bottom(türkis) Layer der Platine

Die Bauteile sind überwiegend SMD-Bauteile. Diese sind sehr platzsparend und lassen sich mit einem Reflow-Lötofen schnell verlöten. Bedrahtete Bauteile zur Durchsteckmontage benötigen außerdem Bohrungen, welche die inneren Lagen durchbrechen. Dort steht dann weniger Kupferfläche zur Stromleitung zur Verfügung.

Anstatt den Psoc Chip direkt auf die Platine zu löten, wurde ein fertiges Modul (CY8CKIT-010 PSoC® CY8C58LP Family Processor Module Kit) benutzt. Diese Modul ist eigentlich zur Benutzung in einem Psoc-Evaluierungsboard gedacht. Darauf sind bereits alle wichtigen Bauteile wie Kondensatoren und Oszillatoren, die der Psoc für sich benötigt, enthalten. Außerdem steht ein Anschluss für das Psoc Programmiergerät zur Verfügung. Die Spannungsversorgung und die GPIO Pins des Psoc sind über Stiftleisten nach unten herausgeführt. Das Modul wurde gewählt, um den Layoutentwurf der Platine und die späteren Lötarbeiten zu vereinfachen. Das Modul wird auf die vier in der Mitte der Platine angeordneten Buchsenleisten J1-J4 (Abb. 21) gesteckt.

An den linken und und rechten Rändern der Platine befinden sich die FETs mit der dazugehörigen Ansteuerschaltung (Abb. 21). Die P-FETs (IPD042P03L3G) sind dabei zusammen mit den Anschlusspads der Motorphasen auf der Vorderseite, die N-FETs (IRLR7843) befinden sich auf der Rückseite. Die Back-EMF Filter befinden sich bei den oberen beiden BLMC direkt dahinter. Bei den unteren beiden sind sie unterhalb des Prozessormoduls platziert. Die Stromzufuhr zu den P-FETs geschieht über Vias direkt aus der spannungsführenden Innenlage. Vias sind mit einem leitenden Material versehene Bohrungen in der Platine. Sie stellen elektrische Verbindungen zwischen den einzelnen Lagen einer Platine her. Von den N-FETs wird der Strom über eine Leiterbahn auf der Masselage, die aber nicht mit der Masse verbunden ist, zum Shunt geführt. Der Shunt ist die breite Zickzackleiterbahn zwischen den FETs. Es befindet sich je einer auf der Vorder- und einer auf der Rückseite. Nach dem Shunt fließt der Strom in die Masselage.

Die Anschlusspads für den Akku sind am unteren Rand der Platine angebracht. Der Strom wird dort über mehrere Vias in die Innenlagen geführt. Um die großen Ströme zu bewältigen, sind auf der Vorder- und der Rückseite Anschlusspads vorhanden, so dass der Strom von beiden Seiten in das Innere der Platine geführt werden kann. Abb. 20 zeigt eine schematische Übersicht über den Stromkreislauf und die Verbindung der einzelnen Komponenten.

Durch die PWM-Modulation der Motoren entstehen starke Fluktuationen im Stromfluss. Um diese abzupuffern, ist eine große Kapazität zwischen Masse und Versorgungsspannung nötig. Daher sind am Rand der Platine und an freien Stellen in der Mitte neun Kondensatoren mit insgesamt 2870 µF verbaut (7x 220 µF; 1x 330 µF ; 1x 1000µF). Diese sind direkt mit den stromverteilenden Innenlagen verbunden.

Der Psoc besitzt vier IO-Port Gruppen, von denen jede mit einer unterschiedlichen IO Spannung betrieben werden kann. Die FETs sowie die Analogeingänge sind auf eine Spannung von 5V ausgelegt. Die an den UART und I2C anzuschließenden Komponenten arbeiten mit 3,3V. Aus diesem Grund wurden diese Anschlüsse und vier extra Pins auf eine eigene Portgruppe geschaltet und diese mit 3,3V versorgt. Die restliche Gruppen und die eigentliche Betriebsspannung des Psoc sind mit 5V verbunden.

In der Nähe der Shunts befindet sich auf jeder Seite noch ein Temperatursensor. So kann die Temperatur der Platine an der Stelle gemessen, werden, wo die Verlustwärme entsteht.

4.5.2 Herstellung

Zur Fertigung der Leiterplatte wurden die Daten an den Leiterplattenhersteller PCB-Pool übergeben. Vor der Fertigung ist es noch wichtig zu prüfen, ob das entworfene Layout den technischen Mindestanforderungen des Herstellers genügt. Dazu zählen vor allem Leiterbahnabstände, Leiterbahnbreiten und Bohrlochgröße. Diese Prüfung kann das Layout-Programm automatisch durchführen, wenn die entsprechenden Parameter korrekt eingestellt sind. Das Ergebnis der Fertigung ist in Abb. 22 zu sehen.

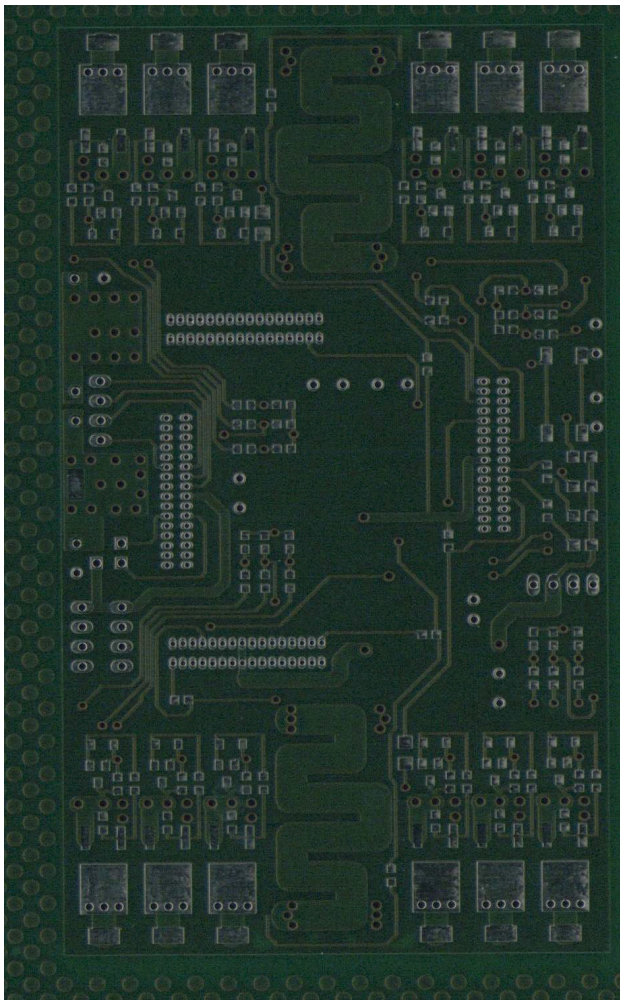

Abbildung 22: Platine nach der Fertigung noch ohne Bauteile

Das Auflöten der Bauteile wurde in Eigenarbeit durchgeführt. Die SMD-Bauteile, welche die überwiegende Mehrheit darstellen, wurden mit dem sogenannten Reflow-Verfahren verlötet. Dazu werden die Lötpads mit Hilfe einer vom Leiterplattenhersteller mitgelieferten Maske mit Lötpaste bestrichen. Dann werden alle Bauteile aufgesetzt. Die komplette Platine wird nun in einem Reflow-Ofen auf bis zu 260°C erhitzt, so dass die Lötpaste schmilzt und alle Bauteile verlötet werden. Die Bauteile zur Durchsteckmontage, wie Anschlussleisten und Kondensatoren, wurden am Ende mit dem Lötkolben von Hand verlötet.

4.5.3 Fehlerhafte Pinbelegung der Analogsignale

Nach der Fertigstellung der Platine wurde versucht, alle vier BLMC-Komponenten im Psoc zu instantiieren und mit ihren zugehörigen Anschlusspins zu verbinden. Dabei wurde festgestellt, dass dies in der vorhandenen Konfiguration nicht funktioniert. Der Psoc-Creator war nicht in der Lage, alle analogen Signale zu den internen Komponenten zu routen. Beim Entwurf der Platine wurde angenommen, dass der Psoc in der Lage ist, Analogsignale beliebig von den IO-Pins zu den internen Komponenten zu routen. Dies ist aber nicht der Fall. Verschiedene Pins teilen sich die internen Analogverbindungn. Es ist nicht möglich, dass sich Pins, die sich eine Verbindung teilen, mit verschiedenen internen Komponenten verbunden werden. Daher müssen beim Belegen der Pins mit Analogsignalen diese entsprechend verteilt werden.

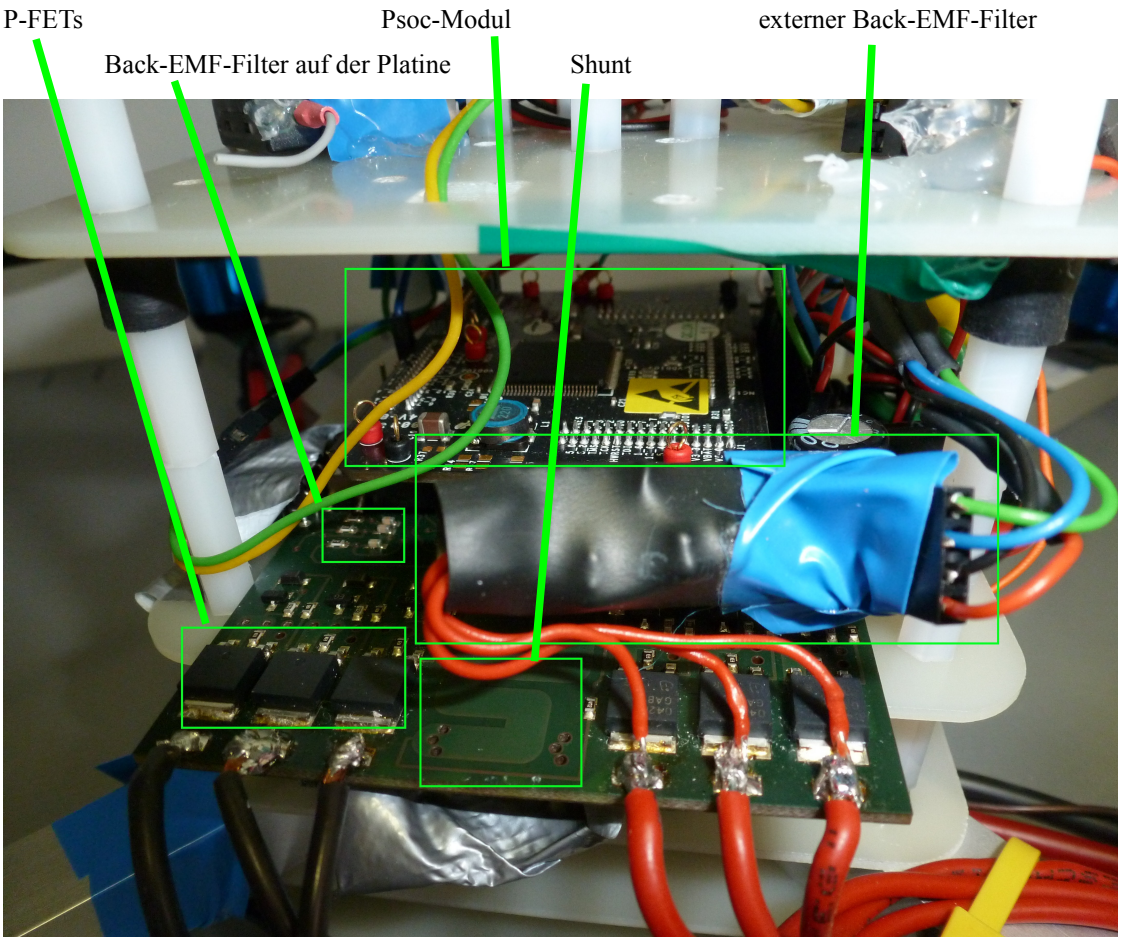

Abbildung 23: Detailansicht der Platine mit extern nachgebauter Back-EMF Filterschaltung

Es wurde jedoch eine Möglichkeit gefunden, dennoch alle vier BLMC zu betreiben, ohne eine neue Platine herstellen zu müssen. Die Back-EMF Filterschaltung für den unteren linken BLMC wurde auf einer kleinen Lochrasterplatine mit bedrahteten Bauteilen nachgebaut (Abb. 23). Die Eingänge wurden direkt an die Anschlusspads der Motorphasen gelötet. Die Ausgänge wurden an die bisher unbenutzte vierpolige GPIO-Stiftleiste angeschlossen. Weiterhin wurde der Anschluss des virtuellen Sternpunktes des oberen rechten BLMC mit dem für TX vorgesehenen Pin von UART 2 verbunden. Dieser Pin ist frei und wird für das UART nicht benötigt, weil an diesem der Empfänger für die Funkfernsteuerung angeschlossen wird und dieser nur unidirektional überträgt, also nur den RX Pin braucht. Durch diese Umbelegung ist ein Routing aller zum Betrieb der BLMC notwendigen Analogsignale, also aller Signale, die zur Back-EMF Auswertung benötigt werden, möglich. Auf die Analogeingänge zur Strom- und Temperaturmessung musste aber verzichtet werden. Ein weiteres Problem ist, dass nun Analogsignale mit einer Spannung von bis zu 5V an den Pins anliegen, die für 3,3V vorgesehen waren. Um den Psoc nicht zu beschädigen, müssen diese nun auch mit 5V Pegel betrieben werden. Dazu wurden die 3,3V Versorgung mit der 5V Versorgung verbunden. Die 3,3V Komponenten, wie IMU (I2C), Bluetoothtelemetriemodul (UART) und Fernsteuerempfänger (UART), müssen jetzt über zusätzliche Pegelwandler angeschlossen werden.
[Psoc5]

4.6 Portierung der Quadrokoptersoftware

Für die Fluglagenregelung und Steuerung des Quadokopoters wird die bereits vorhandene am Lehrstuhl entwickelte Software eingesetzt. Diese ist in C für die AVR32 Plattform geschrieben. Sie beinhaltet bereits alle Algorithmen zur Sensordatenaufbereitung, Sensordatenfusion, Lageregelung und Fernsteuerung des Quadrokopters. Ebenso ist sie in der Lage, Telemetriedatensätze zu einem PC zu übertragen. Zur Kommunikation mit der Funkfernsteuerung, dem PC, der IMU und den BLMCs werden die I2C und UART Schnittstellen des AVR32 benutzt. Diese werden über die im AVR-Software-Framework enthaltenen Treiber angesprochen. Damit die Software nun auf dem Psoc laufen kann, müssen hier ebenfalls kompatible Treiber für die entsprechenden Komponenten im Psoc zur Verfügung gestellt werden. Dies betrifft vor allem die Komponenten

UART für den Fernsteuerempfänger und die Telemetrieübertragung, I2C zur Kommunikation mit der IMU und den Timercounter zur Zeitmessung.

Die Psoc-Entwicklungsumgebung stellt für die I2C und UART Komponenten bereits einfach zu benutzende APIs bereit, die viel Funktionalität integrieren. So ist es auf dem Psoc im Gegensatz zum AVR32 nicht nötig, eigene Interrupt-Routinen zu programmieren, die die Abwicklung der Kommunikation übernehmen. Es genügt die Funktionsaufrufe der I2C und UART Treiber in der Quadrokoptersoftware, an die entsprechenden Funktionen der Psoc Komponenten zu delegieren. Diese führen das Übertragen oder Empfangen von Daten aus den oder in die entsprechenden Datenpuffer dann selbstständig durch. Die zurückgelieferten Status- und Fehlercodes müssen dann noch in die von der Quadrokoptersoftware benutzten Codes übersetzt werden.

Der Timercounter im AVR32 wird von der Quadrokoptersoftware zur Zeitmessung benutzt. Er ist hierzu so eingestellt, dass er Interrupts mit einer Frequenz von 1 khz auslöst. Wegen dieser konstanten Frequenz ist es nicht notwendig, dazu im Psoc einen vollwertigen Timercounter zu benutzen, welcher UDB Blöcke belegen würde. Es reicht aus, ein 1 khz Taktsignal mit den ausreichend vorhandenen Clockdividern zu erstellen und mit einem Interrupt-Eingang zu verbinden. Die Software-Routine diese Interrupts erhöht nun wie auf dem AVR32 den Softwaretimer des Systems. Zusätzlich wird hier auch die Motor_Update() Funktion der BLMCs aufgerufen.

Die BLMCs müssen auf dem Psoc nicht mehr über I2C angesteuert werden, sondern werden über direkten Funktionsaufruf angesteuert.

Die Teile der Quadrokoptersoftware, die optionale Sensoren zur Höhen- und Positionsregelung sowie zur Kollisionsvermeidung über analoge und SPI Schnittstellen ansprechen, wurden entfernt. Wollte man diese später doch nutzen, muss eine Komponente für die entsprechende Schnittstelle im UDB Array des Psoc instantiiert und ein Treiber dafür geschrieben werden. Tests ergaben, dass für eine SPI-Schnittstelle noch genügen UDB-Resourcen verfügbar sind, für eine weitere UART- oder I2C-Schnittstelle jedoch nicht. Möchte man noch analoge Sensoren nutzen, sind zwar ADCs vorhanden, aber die internen Routingmöglichkeiten zu diesen sind beschränkt. Diese Analogeingänge müssen beim Platintnlayout berücksichtigt werden und mit passenden Pins des Psoc verbunden werden. Im Allgemeinen sind allerdings auch nur noch 6 GPIO Pins für weitere Sensoren frei.

5 Evaluierung

5.1 Vergleich mit Mikrokopter-BLMC

Um die Effizienz des BLMC zu beurteilen, soll dieser mit dem BLMC von Mikrokopter verglichen werden. Dazu wird der Testaufbau aus Kap. 4.3.4 benutzt (Motor mit Propeller, Kap. 4.3.4 Abb. 18.). Das Gestell, auf dem der Motor montiert ist, wird auf eine Waage gestellt. Die Waage wird auf Null kalibriert, wenn der Propeller steht. Läuft der Propeller und erzeugt Schub nach oben, entspricht die Gewichtsabnahme des Aufbaus der Schubkraft des Propellers.

Als erstes wird nun der BLMC von Mikrokopter angeschlossen und mit 8 verschiedenen Stellwerten von 12 bis 175 angesteuert. (12;25;50;75;100;125;150;175) Der maximale Stellwert ist 255. Für jeden Stellwert wird die Schubkraft und die Stromaufnahme mit einem Multimeter gemessen. Im Anschluss daran wird einer der vier Psoc-BLMC angeschlossen. Dabei ist die gleiche Software geladen, die für den Testaufbau benutzt wurde und eine Vorgabe der Stellwerte über ein serielles Terminal ermöglicht. Dem Controller werden nun Stellwerte vorgegeben, so dass er jeweils für jeden Messpunkt genauso viel Strom verbraucht wie der Mikrokopter-BLMC. Dann wird wieder der Schub abgelesen. Die Grafik Abb. 24 zeigt das Ergebnis.

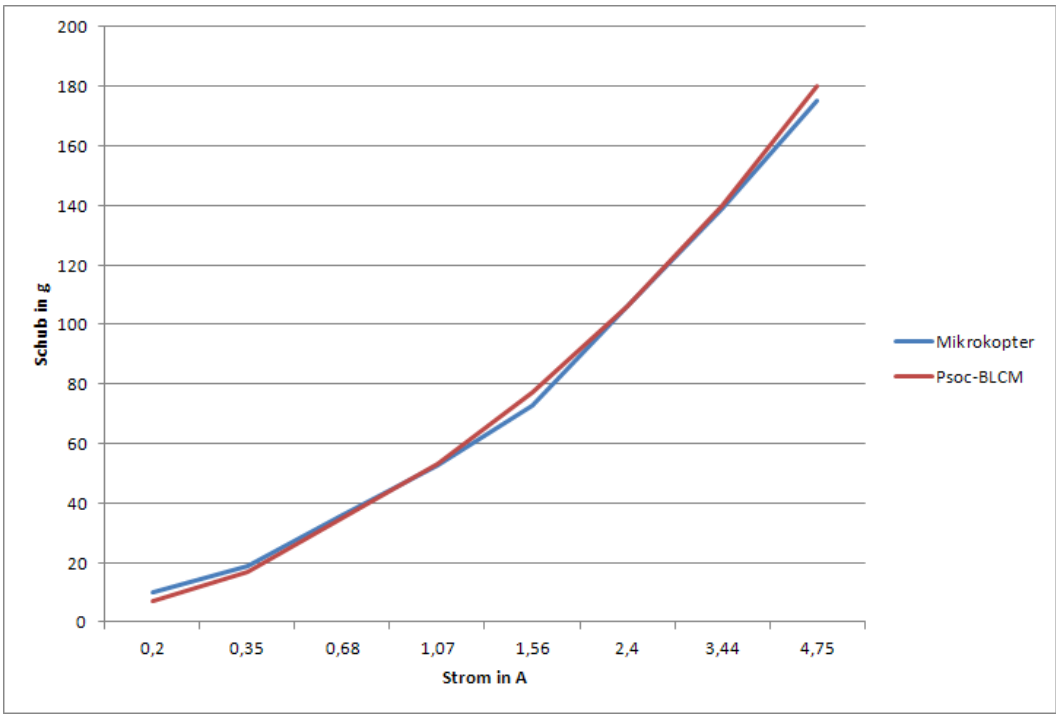

Abbildung 24: Vergleich der Effizienz von Mikrokopter und Psoc-BLMC

Man sieht, dass der Motor im unteren Leistungsbereich etwas mehr Schub erzeugt, wenn er am Mikrokopter-BLMC angeschlossen ist. Dieser Bereich geht bis etwa 10% des Stellwertes. Danach sind Psoc-BLMC und Mikrokopter-BLMC etwa gleich effizient.

In einem weiteren Versuch wurde die Zeit ermittelt, die vom Setzen eines neuen Stellwertes bis zu dem Zeitpunkt, wo sich der neue Schubwert einstellt, verstreicht. Dazu wurde die Möglichkeit der Waage genutzt, die Messwerte über eine RS232-Schnittstelle zu übermitteln. Es wurde am PC ein kleines Programm geschrieben, welches im BLMC einen neuen Stellwert setzt und dann die von der Waage gesendeten Messwerte aufzeichnet. Abb. 25 zeigt den Verlauf bei einer Stellwertveränderung von 30% auf 60%. Diese dauert bei beiden Controllern etwa 600 ms. Die direkte Übernahme der Stellwerte beim Psoc-BLMC bringt also keine Vorteile gegenüber der Mikrokopter-BLMC, die über den I2C-Bus angesteuert werden.

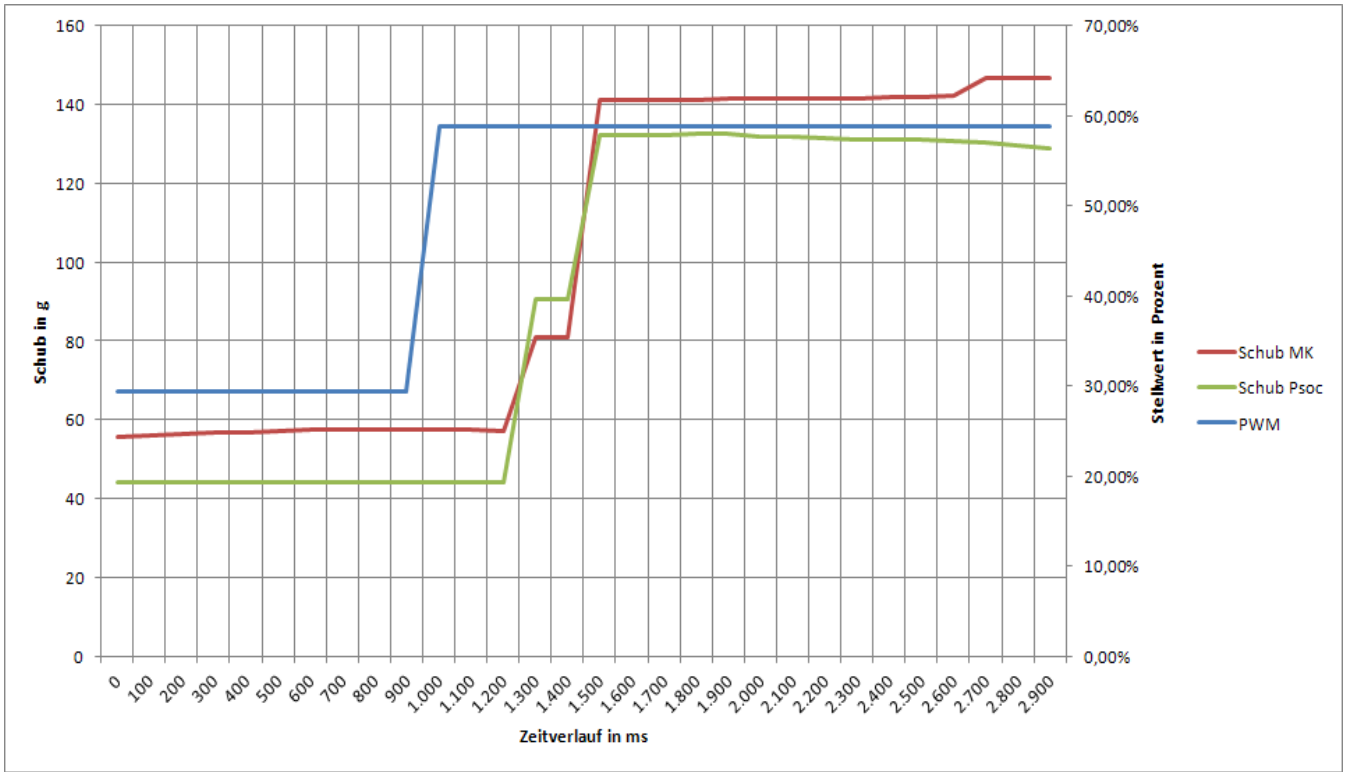

Abbildung 25: Zeitlicher Verlauf einer Stellwertänderung bei beiden BLMC

In Tabelle 3 sind die Kosten des voll integrierten Regel- und BLMC-Systems mit Psoc und die Kosten des Regelsystems mit AVR32 und vier Mikrokopter-BLMC gegenübergestellt. Bei AVR32 wird zwischen den alten(v1.2) und neuen BLMC (v2.0) von Mi-

krokopter unterschieden Beim AVR32 System gibt es noch zusätzlichen Aufwand durch die Verkabelung der einzelnen Komponenten. Beim Psoc-System müssen die Bauteile auf der Platine verlötet werden. Außerdem sind beim Psoc-System Kosten von wenigen Euro für Kleinteile nicht berücksichtigt. Ebenfalls aufgeführt ist ein Psoc-System mit den unter Kap. 6.2 Angegebenen Einsparmöglichkeiten.

Mikrokopter BLMC v1.2 + AVR32		Mikrokopter BLMC v2.0 + AVR32		Integriertes Psoc Regelsystem		Psoc Regelsystem mit Einsparungen	
4x MK-BLMC	160 €	4x MK-BLMC	220 €	Fertigung Platine	112 €	Fertigung Platine	72 €
AVR32-Borad	40 €	AVR32-Borad	40 €	Psoc5-Modul	70 €	Psoc5-Chip	32 €
				12x P-FETs	15 €	12x P-FETs	15 €
				12x N-FETs	13 €	12x N-FETs	13 €
Summe	**200 €**	**Summe**	**260 €**		**211 €**		**132 €**

Tabelle 3: Kostenvergleich zwischen MK v1.2/.2.0 +AVR32, Psoc und Psoc mit Einspaarungen

5.2 Flugtest im Quadrokopter

Um zu zeigen, dass das System tatsächlich flugfähig ist, wurde die Platine in einen der am Lehrstuhl benutzten Quadrokopter eingebaut (Abb. 26). Ein nun erfolgter erster Flugversuch zeigte, dass es dem System möglich ist, die Lage des Quadrokopters im Flug stabil zu regeln und Flugmanöver durchzuführen. Um genauere Aussagen über die Qualität der Regelung zu machen, wurden die Soll- und Ist-Werte der Roll- und

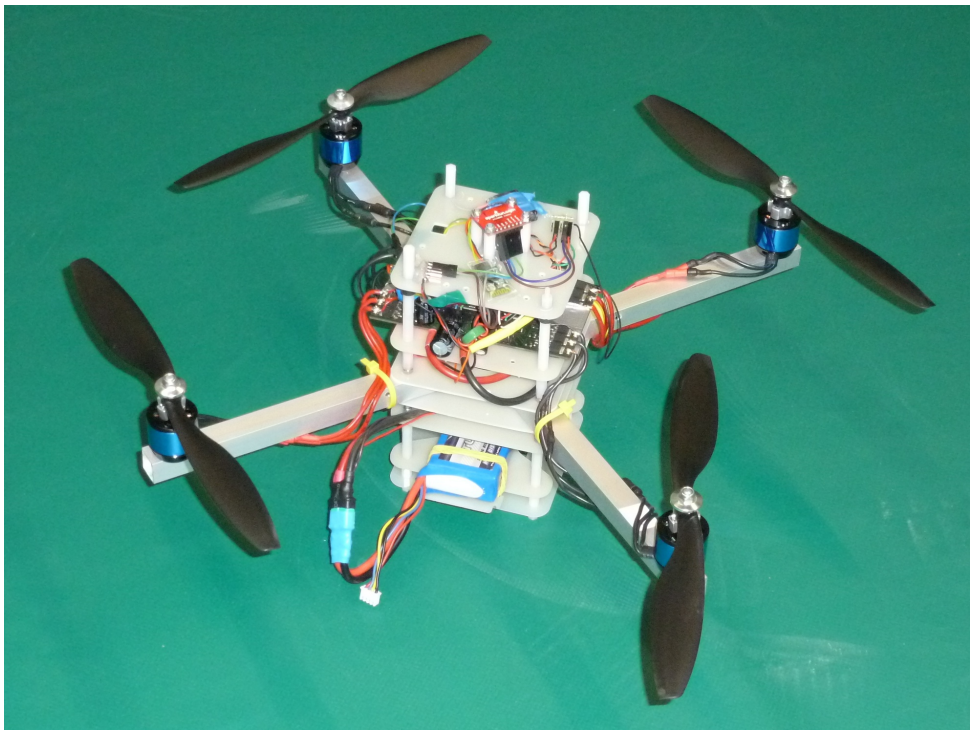

Abbildung 26: Fertig aufgebauter Quadrokopter

Pitchwinkel als Telemetriedaten über das Bluetoothmodul zum PC übertragen. Dort wurden sie mit dem Telemetrietool für den Quadkopter aufgezeichnet und grafisch dargestellt.

Der Quadrokopter wurde jetzt wiederholt vor und zurück bzw. links und recht geflogen. Abb. 28 zeigt dabei den Verlauf der Pitch-Soll- und Ist-Winkel beim Vor- und Rückwärtsflug. Abb. 27 zeigt den Verlauf der Roll-Soll- und Ist-Winkel beim Seitwärtsflug. Bei Pitch- und Roll-Winkel sieht man hier eine gute Nachführung der Ist-Werte mit wenig Überschwingen.

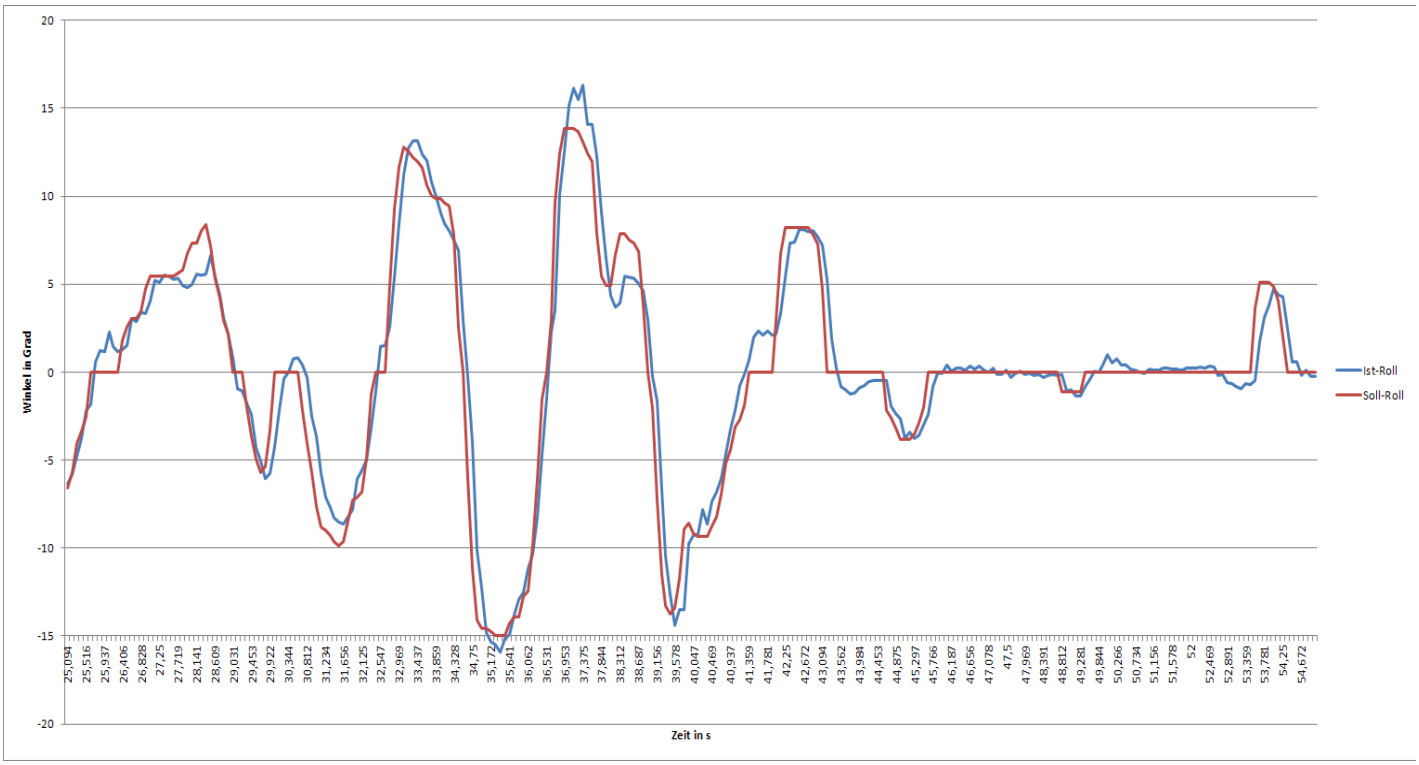

Abbildung 27: Zeitlicher Verlauf der Roll Soll- und Ist-Winkel

Während der Flugtests führte ein Problem mit dem Fernsteuerempfänger zu mehreren Abstürzen. Auf der UART Verbindung vom Fernsteuerempfänger zum Psoc traten Übertragungsfehler auf und die Software interpretierte daraufhin einen Steuerwert falsch und schaltete alle Motoren ab. Es wurde zuerst vermutet, dass die unterschiedlichen Pegel von Fernsteuerempfänger und Psoc dafür verantwortlich sind. Der Fernsteuerempfänger arbeitet mit 3,3V, der Psoc mit 5V. Laut Datenblatt erkennt der Psoc ab einer Spannung von 2,5V ein HIGH Signal, wenn der Port auf TTL-Pegel einge-

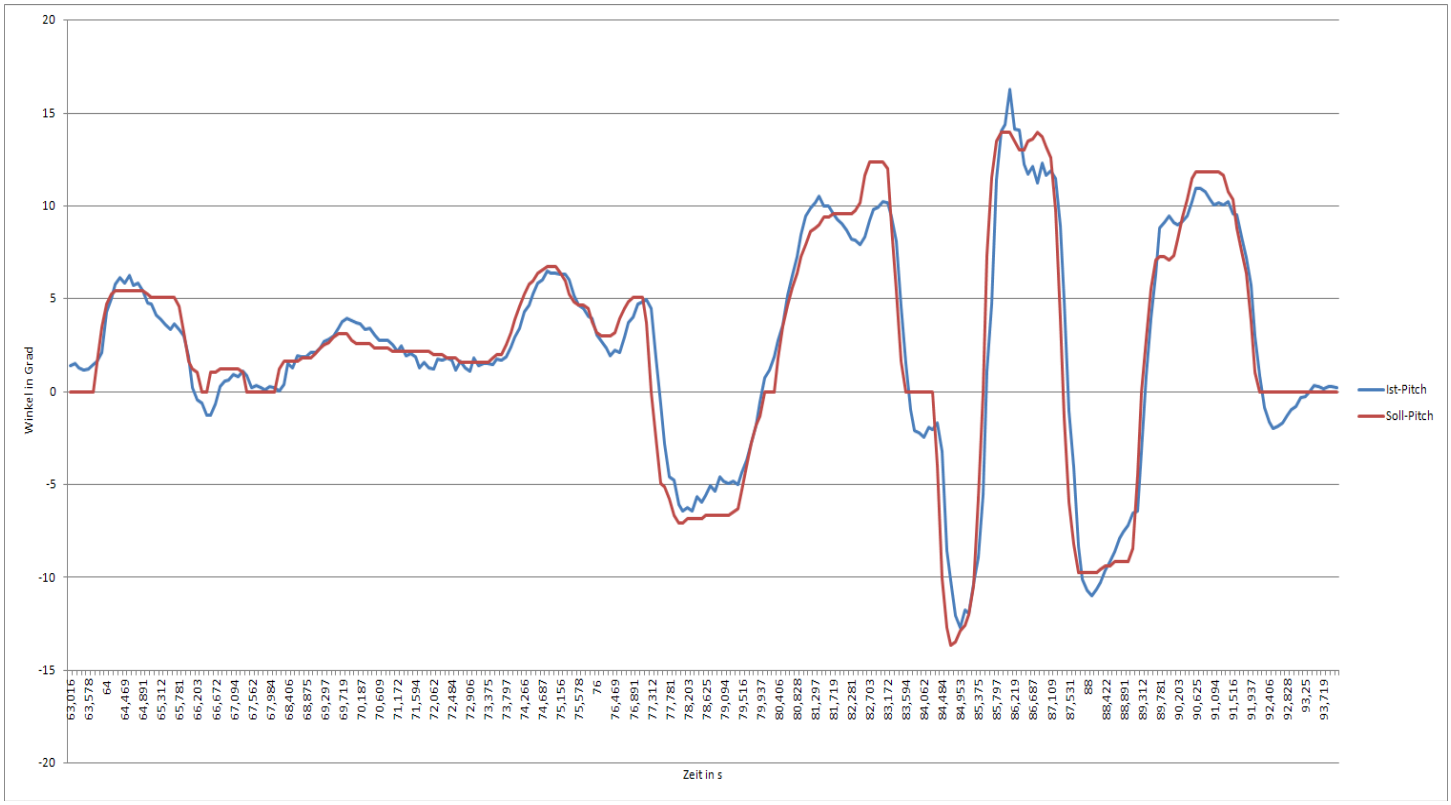

Abbildung 28: Zeitlicher Verlauf der Pitch Soll- und Ist-Winkel

stellt ist. Daher wurde auf einen extra Pegelwandler verzichtet. Ein nachträglicher Einbau eines solchen brachte allerdings auch keine Lösung. Die genaue Ursache des Problems konnte, auch wegen seiner schlechten Reproduzierbarkeit noch nicht ermittelt werden. Damit nicht sofort jeder falsch erkannte Steuerwert zu einem Absturz führt, wurde die Software so geändert, dass die Motoren erst abgeschaltet werden, wenn mindestens für 100 ms kein gültiger Wert erkannt wird. Aber selbst dies konnte Abstürze nicht restlos ausschließen.

6 Fazit / Ausblick

6.1 Fazit

Die in Kapitel 5 erzielten Ergebnisse zeigen, dass das System geeignet ist, die Fluglage eines Quadrokopters stabil zu regeln. Die Ansteuerung der Brushlessmotoren ist dabei genauso effizient wie bereits am Markt erhältliche Systeme. Der Platzbedarf im Quadrokopter beschränkt sich dabei aber auf eine Platine, die in einer einzigen Ebene Platz findet. Mit dem AVR32-Regelsystem wurde eine Ebene für das AVR32-Board und eine Ebene für die Mikrokopter-BLMC benötigt, wenn diese im Inneren des Quadrokopters untergebracht werden. Allerdings ist man dazu übergegangen, die Motorcontroller an den Auslegern zu befestigen, was nun keinen Platzvorteil des Psoc-Regelsystems mehr bedeutet. Betrachtet man die gesamte Fläche der Platinen der einzelnen Komponenten, kommt das AVR32-System mit vier Mikrokopter-BLMC auf 68 cm² und das Psoc-System auf 84 cm² Platinenfläche. Beim Preis ist das Psoc-System etwas günstiger im Vergleich zum AVR32-System mit den aktuell benutzten Mikokopter BLMC. Vergleicht man das Psoc-System mit den neuen Mikokopter-BLMCs, die mehr Strom aushalten und so technisch eher mit dem Psoc-System vergleichbar sind. Schneidet der Psoc besser ab. Beim Psoc-System gibt es aber auch noch die unter Kapitel 6.2 beschriebenen Einsparmöglichkeiten. Noch ein Vorteil des Psoc-Systems ist der Wegfall des I2C Busses zwischen Regelsystem und Motorcontroller. Dies beseitigt eine mögliche Fehlerquelle. Der verfügbare Quellcode und die daraus resultierende leichte Erweiterbarkeit sind ein weiterer Vorteil.

Ein Nachteil des Psoc-Systems ist, dass am Psoc nur noch 6 Pins verfügbar sind, was die Anschlussmöglichkeiten weiterer Sensoren und Komponenten, die nicht über I2C angesteuert werden, einschränkt. Ebenso sind die Features der Psoc-Creator Entwicklungsumgebung bei der Softwareentwicklung nicht auf dem Stand moderner Entwicklungsumgebungen wie z.b. Eclipse. Dies lässt sich jedoch als externe Codeeditor einsetzen. Das Compilieren erfolgt dann aber trotzdem mit Psoc-Creator.

Weiterhin ungelöst ist das Problem mit dem UART des Fernsteuerempfängers, das bis zuletzt zu Abstürzen führte.

6.2 Verbesserungsmöglichkeiten und Ausblick

Das Psoc-System funktioniert wie erwartet, bietet aber noch Raum für Verbesserungsmöglichkeiten. Dies wäre beispielsweise die Implementierung einer Drehzahlregelung. Das wäre einfach im Softwareteil des BLMC möglich, da die Drehzahl bereits gemessen wird und zur Verfügung steht.

Auch kann noch die Unterstützung für herkömmliche Gleichstrommotoren nachgerüstet werden. Diese würden dann an zwei der drei Phasen eines Controllers angeschlossen. Die Schaltung entspricht dann einer H-Brücke. Somit ist eine PWM-Ansteuerung und eine Drehrichtungsumkehr möglich. Zur entsprechenden Ansteuerung der Motoren muss für den Psoc dann ein neues Ansteuermodul erstellt werden. Beim Entwurf des Softwaremoduls wurde bereits darauf geachtet, dass sich neue Motoransteuermodule einfach einbinden lassen und dies gegenüber dem Nutzer transparent geschieht. Dieser kann also verschiedenartige Motoren über die gleiche API steuern und diese auch beliebig austauschen.
Wird das Routing der Platine noch dahingehend optimiert, dass alle Analogsignale mit geeigneten Pins verbunden sind, so dass wie ursprünglich vorgesehen die 3,3V Komponenten direkt mit 3,3V Pegel angesteuert werden können, vereinfacht dies den Aufbau weiterhin. Die Strom- und Temperaturüberwachung könnte dann auch implementiert werden.

Wenn der Psoc-Chip direkt auf die Platine verlötet wird, kann auch auf das aufgesteckte Psoc-Modul verzichtet werden. Dazu würden anstatt der Buchsenleisten für das Modul der Psoc-Chip, Oszillator, Kondensatoren und ein Programmieranschluss direkt auf der Platine verlötet. Dies würde zu einer geringeren Bauhöhe und einem geringeren Gewicht der Platine führen. Der einzelne Psoc-Chip ist auch wesentlich günstiger als das Modul.

Weiterhin kann die Strombelastbarkeit der Motorcontroller erhöht werden. Die Limitierung liegt hier nicht bei den einzelnen Controllern, sondern bei der Verteilung des Gesamtstroms aller vier Controller. Um diese Limitierung zu umgehen, könnte man den Strom anstatt über die Platine, mit Kabeln direkt zu den einzelnen Controllern führen. Dann könnte man auch auf eine vierlagige Platine verzichten und nur eine doppellagige benutzen, was die Herstellungskosten für diese um ca. 40 Euro reduziert.

7 Literaturverzeichnis

[Probst] Probst, Uwe: Servoantriebe in der Automatisierungstechnik
Vieweg+Teubner, 2011

[Mikrokopter] Mikrokopter: Homepage – URL
http://www.mikrokopter.de

[Microchip] Microchip Application Note AN857: Brushless DC Motor Control Made Easy, 2002

[Shao] Shao, Jianwen: Direct Back EMF Detection Method for Sensorless Brushless DC, Master Thesis, Faculty of the Virginia Polytechnic Institute and the State University, 2003

[Mikocontroller] mikrocontroller.net: Brushless-Controller für Modellbaumotoren – URL
http://www.mikrocontroller.net/articles/Brushless-Controller_für_Modellbaumotoren

[Atmel] Atmel Application Note AVR444:
Sensorless control of 3-phase brushless DC motors, 2005

[Torquemotor] Wikipedia: Torquemotor, 2013 – URL
http://de.wikipedia.org/wiki/Torquemotor

[Gleichstrommotor] Wikipedia: Gleichstrommaschine, 2013 – URL
http://de.wikipedia.org/wiki/Gleichstrommaschine

[Psoc5] Cypress Semiconductor: PSoC5: CY8C55 Family Datasheet, 2012

[PID] Edu Wiki: Regelung – URL
https://wiki.enterpriselab.ch/edu/workspace:modules:intro:ss08:summary:regelung

[UAVP] UAVP-NG Quad Brushless Controller (NGblc-4mini) - Quad Brushless Controller - URL http://ng.uavp.ch/shop/order

[Herkules] Andreas Baier: The new Quad Brushless ESC"HERKULES III" – URL
http://www.andreasbaier.de/index.php/en/herkules-iii-generation-3