Einbindung eines generischen Mikrocontrollers inklusive analoger und digitaler Peripherieelemente in einer FPGA-basierten Hardware-in-the-Loop Simulationsumgebung für Leistungselektronik.

Stefan Stark, BSc

FH Vorarlberg - University of Applied Sciences
stefan.stark@students.fhv.at

September 2017

Zusammenfassung

In diesem Artikel wird ein neuer Weg aufgezeigt wie ein generischer Mikrocontroller unter Berücksichtigung des Simulations-Taktes sowie der analogen und digitalen Peripherie in eine existierende Simulationsumgebung eingebettet werden kann, um damit Leistungselektronik-Schaltungen zu simulieren. Es wird erläutert, welcher Prozessorkern für den Mikrocontroller gewählt wurde, wie die analogen und digitalen Peripherien implementiert wurden und wie der generische Mikrocontroller mit der existierenden Simulationsumgebung kommuniziert. Zur Verifikation des entwickelten Konzeptes wird der generische Mikrocontroller gemeinsam mit einem Modell für einen Abwärtswandler getestet.

I. Einführung

In der Leistungselektronik haben Mikrocontroller (englisch: Microcontroller, kurz μC) die Aufgabe, schaltende elektronische Bauteile, welche für die Umformung der elektrischen Energie benötigt werden, anzusteuern. Der μC steuert die schaltenden elektronischen Bauteile mit Hilfe des implementierten Reglers so an, dass die Regelgröße (zum Beispiel die tatsächliche Ausgangsspannung eines Schaltnetzteils) der Führungsgröße (vom Benutzer eingestellte gewünschte Ausgangsspannung) entspricht.

Leistungselektronikhersteller modellieren und simulieren die Hardware (beispielsweise Schaltnetzteiltopologien wie einen Abwärtswandler) und die Software (Regelungskonzepte), um damit früh in einem Projekt Schwächen oder Fehler eines Konzeptes zu detektieren und gegebenenfalls zu beheben. Neben den daraus folgenden Kostenersparnissen ermöglicht eine Modellierung des Systems, Toleranzen von elektronischen Bauteilen und Temperatureinflüsse ohne großen Aufwand zu berücksichtigen. Die entwickelten Modelle können darüber hinaus zur Optimierung oder Fehlersuche von Prototypen / Produkten verwendet werden, da interne Systemgrößen im Modell ohne Probleme gemessen werden können. Zudem wird durch die Modellierung des Systems eine parallele Entwicklung von Hardware und Software ermöglicht, was zu einer drastischen Reduzierung der Entwicklungszeit und somit der Entwicklungskosten führt. Um all die erwähnten Vorteile nützen zu können, müssen die Modelle der Hardware und der Software möglichst detailgetreu und zueinander kompatibel sein.

Anhand einer Untersuchung des Stands der Technik wurden Ansätze herausgefiltert, die am besten die geforderten Ziele erfüllen:

- Der Lösungsansatz soll bereits vorhandene Simulationsumgebungen nützen (Simulink/PLECS).
- Die analoge und digitale Peripherie des μC soll im Lösungsansatz berücksichtigt werden.
- Beliebige μC sollen mit diesem Ansatz getestet werden können.
- Der nötige Aufwand, um eine Simulation

zu starten und zu bedienen, sollte gering sein.
- Die Kosten für einen Aufbau (Hardware, Lizenz, usw.) sollten gering sein, um eine Multiplizierbarkeit zu gewährleisten.
- Die Simulationsgeschwindigkeit sollte sich an der Geschwindigkeit des vorhandenen Systems orientieren.

Der Processor-in-the-Loop (PiL) Ansatz verwendet Simulink (STM32 MAT/Target), PLECS (PLECS-PIL), Virtual-Testbench ([1]) oder eine andere Simulationsumgebung auf einem PC für die Modellierung der Leistungselektronik (kurz LE). Die Software wird auf einem Evaluierungsboard ausgeführt, das mit einer seriellen Schnittstelle mit dem PC verbunden ist. Durch diese Verbindung wird keinerlei Peripherie des μC berücksichtigt. Gegen einen Lösungsansatz, der auf dem PiL-Prinzip basiert, spricht zusätzlich die Simulationsgeschwindigkeit und der Aufwand, eine solche Simulation aufzusetzen.

Die Artikel [2] und [3] modellieren den Prozessor und die LE auf einem FPGA. Der Prozessor wird als MicroBlaze (Soft-Core IP der Firma Xilinx) in den FPGA eingebettet. Eine Transformation der LE in den Zustandsraum ermöglicht anhand eines Lösungsalgorithmus das Ausführen der LE im selben FPGA. Die Kommunikation mit dem Benutzer (Bedienung und Visualisierung der Ergebnisse) geschieht über eine serielle Verbindung. Dieser Ansatz hat zwei Vorteile: die Geschwindigkeit und dass beliebige Prozessoren mit beliebiger Peripherie modelliert werden können. Da die analoge Peripherie nicht berücksichtigt wird, keine bestehende Simulationsumgebung benützt wird und der Aufwand, eine Simulation aufzubauen und zu bedienen, groß ist, ist diese Art zu simulieren nicht für einen Lösungsansatz geeignet.

Bei Hardware-in-the-Loop (kurz HiL) Simulationen wird der Software Algorithmus auf einer Hardware in der Schleife zur Simulationsumgebung ausgeführt. Die Artikel [4] - [8] führen, ähnlich wie die Artikel [2] und [3], die LE in einer Zustandsraumdarstellung auf einem FPGA aus. Die LE wird von Hand oder mit Tools von Drittherstellern (dSpace in Artikel [4]) in den Zustandsraum auf den FPGA gebracht (was einem hohen Aufwand entspricht). Die Software in [5] wird über eine Schnittstellenkarte auf einem Entwicklungsboard ausgeführt, dadurch gehen die Informationen der Peripherie nicht verloren. Wie in [4] werden in [5] allerdings nur bestimmte μC unterstützt. In den Artikeln [6], [7] und [8] werden die Steuersignale innerhalb ([6]) oder außerhalb des FPGA ([7] und [8]) für die LE generiert. In diesen Artikeln wird keine Peripherie berücksichtigt. Der Hauptgrund gegen den HiL Ansatz, bei dem die LE auf einem FPGA ausgeführt wird, ist der Aufwand, die LE in Zustandsform auf den FGPA zu platzieren und auszuführen.

Andere HiL Ansätze führen die LE in Simulink oder PLECS auf einem PC aus. Die Kommunikation zwischen der LE und dem μC wird mittels einer Schnittstellenkarte realisiert. Damit beide Systeme (Simulink und μC) den gleichen Ausführungstakt haben, wird der Takt von Simulink vorgegeben. Durch den variablen Solver von Simulink entstehen variierende Taktfrequenzen. Mit diesen variierenden Taktfrequenzen können nicht alle μC umgehen, deshalb eignet sich dieser Ansatz auch nicht für einen Lösungsansatz.

Der restliche Artikel ist in 5 Abschnitte unterteilt. Im Abschnitt II des Artikels wird das gewählte Konzept beschrieben. Die Modellierung der Peripherie wird in Abschnitt III erläutert. Die Funktion des entwickelten Systems wird in Abschnitt IV dargestellt. Eine Diskussion in Abschnitt V bildet den Schluss dieses Artikels.

II. Konzeptbeschreibung

Im gewählten Konzept (illustriert in Abbildung 1) wird der Mikrocontroller-Prozessor inklusive der digitalen Peripherie auf einem FGPA oder SoC modelliert. Über ein Bus-System tauscht der Prozessor Daten mit der entwickelten digitalen Peripherie aus. Die Kommunikation zwischen dem FPGA und Simulink, auf der die LE und die analogen Peripherien modelliert werden, ist über die PCIe-Schnittstelle und einen S-Function-Block in Simulink realisiert

worden. Wie im HiL Ansatz, wo die LE in Simulink ausgeführt wird, generiert Simulink den Takt für den Mikrocontroller-Prozessor und die digitale Peripherie. Der gewählte Prozessor und die Peripherie sollten deshalb unempfindlich gegenüber Taktänderungen sein.

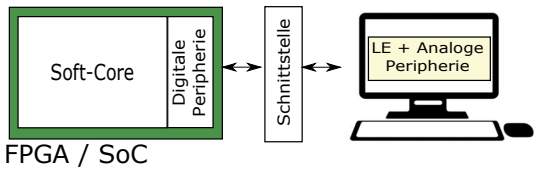


Abbildung 1: *Konzeptblockschaltbild*

Der NIOS II/e wurde als Soft-Core IP ausgewählt, da die Alternativen von ARM ([9] und [10]) nicht verfügbar waren und Tests zeigten, dass der NIOS II/e unempfindlich gegenüber einer Variation der Taktfrequenz ist. Neben dem Prozessor wird ein Speicher benötigt auf dem die abzuarbeitende Software platziert wird. Über das Bus-System (Avalon-MM) können der Speicher und selbst generierte digitale Peripherien angeschlossen und verwendet werden.

III. Peripherie

Die implementierten Funktionen der analogen und digitalen Peripherie wurden an die Funktionen der Peripherie des $STM32F0$ angelehnt.

i. Analoge Peripherie

Für das Betreiben eines Abwärtswandlers werden Analog-Digital-Umsetzer (kurz ADC) und Komparatoren (kurz COMP) benötigt. Die Modellierung der analogen Peripherie geschieht auf der Funktionsebene. Als Modellierungsumgebung wurde PLECS ausgewählt, in der mittels C-Script Blöcken ein Endlicher Automat (kurz FSM) aufgebaut wird. Mit dieser FSM wird das Verhalten der analogen Peripherie abgebildet.

Die entwickelte analoge Peripherie behandelt statische und dynamische Signale/Parameter. Dynamische Signale/Parameter haben einen Signal-Eingang in das Modell (grüne Pfeile in Abbildung 3), um direkt in die FSM einzuwirken. Die statischen Parameter können bei vorhandenen Peripherie-Konfigurationen mit dem Prozess in Abbildung 2 beim Starten der Simulation in das Modell übernommen werden.

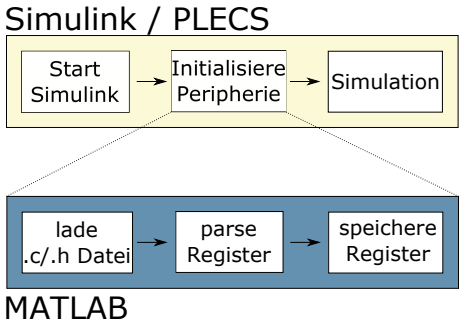
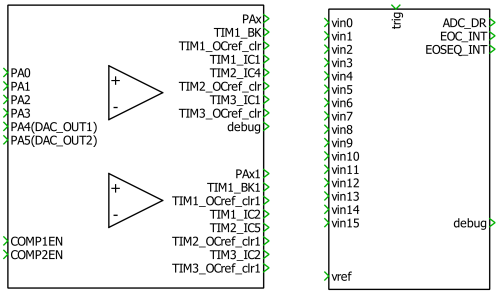


Abbildung 2: *Automatische Zuweisung der statischen Parameter*

i.1 Komparator

Wie in Abbildung 3a ersichtlich, wurden zwei Komparatoren mit zahlreichen Ausgängen (rechte Seite im Modell) entwickelt. Die Eingangsspannung und die Referenzspannung können an die Pins $PA0 - PA5$ angelegt werden. Mit den Eingängen $COMP1EN$ und $COMP2EN$ können die Komparatoren aktiviert werden.



(a) *Komparator* **(b)** *SA-ADC*

Abbildung 3: *Analoge Peripherie - PLECS Modelle*

Funktionen wie die Wahl der Ausgangspolarität, welche Hysterese zur Abschaltschwelle

zugeschaltet wird und welcher Ausgang verwendet wird können über statische Parameter eingestellt werden.

i.2 SA-ADC

Der ADC wird als Sukzessive-Approximation (kurz SA) ADC ausgeführt. Das Modell in Abbildung 3b weist neben zahlreichen Signaleingängen ($vin0 - vin15$) einen Triggereingang ($trig$) und einen Eingang für die Referenzspannung (ref) auf. Die umgewandelten Daten werden am ADC_DR Ausgang für jeden Kanal bereitgestellt. Die weiteren Ausgänge signalisieren ein Konvertierungsende eines einzelnen Kanales (EOC_INT) oder das Konvertierungsende einer Sequenz (mehrere Kanäle, $EOSEQ\_INT$).

Über die statischen Parameter kann man u.a. folgende Funktionen einstellen: welcher Eingangskanal ($vin0 - vin15$) soll umgewandelt werden, in welcher Reihenfolge sollen die Kanäle umgewandelt werden ($vin0$ nach $vin15$ oder $vin15$ nach $vin0$), welche Auflösung soll für den Umwandlungsprozess verwendet werden und wie lange soll ein Signal abgetastet werden.

ii. Digitale Peripherie

Für die Verifikation des Abwärtswandlers werden Timer benötigt um Referenzspannungen zu erzeugen, auf Signale vom Komparatoren zu reagieren und den Schalter des Abwärtswandlers zu bedienen. Um diesen Anforderungen gerecht zu werden wurde ein Timer implementiert, der diese Funktionen abdeckt. Der Timer und die implementierten Funktionen wurden anhand der $TIM2$ und $TIM15$ vom $STM32F0$ entwickelt. Es wurden Grundmodule ($counter\_stage$, $input\_stage$, $trigger\_stage$, ccr_stage und verschiedene $output\_stage$) erstellt, welche miteinander verknüpft in einem Top-Level einen Timer repräsentieren (illustriert in Abbildung 4).

Für das Betreiben des Abwärtswandlers wurden zwei Timer mit verschiedenen Ausgangsstufen entwickelt. Die Logik, die das Verhalten

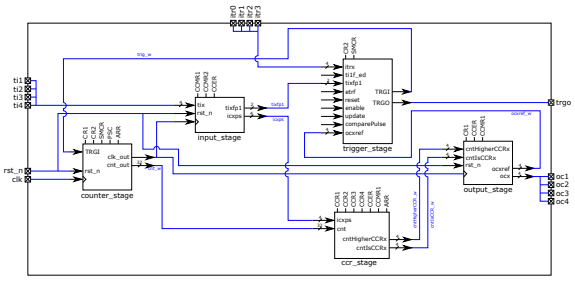


Abbildung 4: *Toplevel eines entwickelten General-Purpose Timer*

der Timer beschreibt ($Task\ Logic$), wurde mittels SystemVerilog in Form einer Verhaltensbeschreibung entwickelt.

Damit die Timer mit dem Soft-Core kommunizieren können, ist ein Register-File ($Register\ file$) und eine Avalon-Schnittstelle ($Avalon\ Slave\ Interface$) nötig. Das Register-File beinhaltet für jedes Register vom Timer eine spezifische Adresse (in Form eines Offsets). Die Avalon-Schnittstelle wandelt die Signale vom Avalon-Bus so um, dass die Register des Timers beschrieben oder gelesen werden.

Eine spezifische Komponente, wie z.B. der Timer, besitzt einen Aufbau wie in Abbildung 5. Durch diesen modularen Aufbau ist es möglich, dass eine spezifische Peripherie mit einem anderen Prozessor kommunizieren kann, indem die $Avalon\ Slave\ Interface$ Komponente ausgewechselt wird.

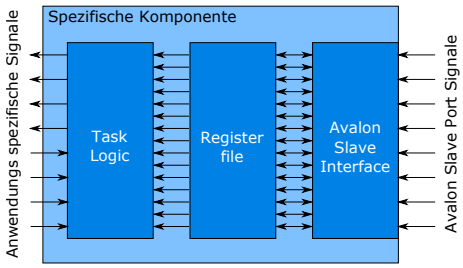


Abbildung 5: *Aufbau einer spezifischen Komponente*

Die digitale Peripherie wird über den Avalon-Bus ($Avalon - MM$) mit dem Prozessor verbunden. Der Prozessor führt die Software aus, die auf dem Speicher ($ON - CHIP\ RAM$) abgelegt wird. Ein solches Soft-Core-System ist in Abbildung 6 illustriert. Über Verbindun-

gen (*conductions*) können die implementierten Komponenten über die Pins des $FPGA - CHIP$, über die Pins der FPGA Karte (*Proce*) und über den S-Function-Block mit Simulink Daten austauschen. Der Takt für das Soft-Core-System wird von dem S-Function-Block in Simulink erzeugt und über den Pin *clk_in* in den $clk - tree$ geführt. Von dieser Komponente aus werden alle restlichen Komponenten mit demselben Takt (*clk*) versorgt.

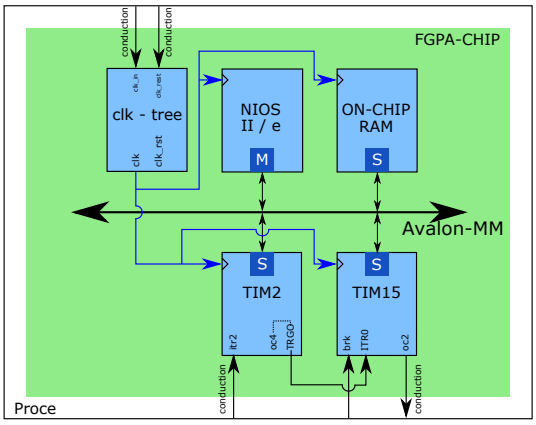


Abbildung 6: *Soft-Core-System*

IV. Verifikation

Die Verifikation des Systems soll sicherstellen, dass der NIOS II/e sowie die restlichen Komponenten in Abbildung 6 mit der sich ändernden Taktfrequenz zurecht kommen.

i. Abwärtswandler

Als Modell für die LE wird ein Abwärtswandler wie in Abbildung 7 in PLECS realisiert. Das PLECS Modell wird in Simulink eingebettet, damit eine Kommunikation mit dem S-Function-Block respektive dem Soft-Core-System möglich ist. Der Mittelwert des Stroms durch die LED beziehungsweise der Mittelwert des Stroms durch die Induktivität (L) soll durch das Setzen des maximalen Spitzenstroms (iL_ipeak) in L oder durch die Dauer der Öffnung ($toff + tdead$) des Schalters (m) vom Benutzer bestimmt werden.

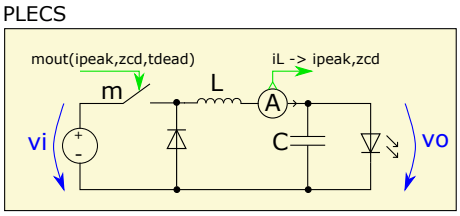


Abbildung 7: *Abwärtswandler mit relevanten Signalen*

Ein möglicher Ablauf wird in Abbildung 8 dargestellt. Der Schalter m wird von einem Timer $TIM15$ so angesteuert, dass sich eine Stromform durch die Induktivität iL wie in Abbildung 8 ergibt.

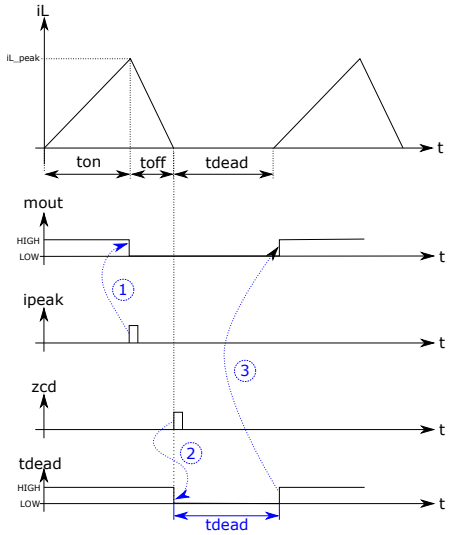


Abbildung 8: *Relevante Signale eines Abwärtswandlers*

Wenn $TIM15$ mit einem $HIGH$ den Schalter schließt, beginnt der Strom durch die Induktivität iL mit dem Faktor $\frac{vi-vo}{L}$ zu steigen. Ein Komparator vergleicht den Spulenstrom (iL) mit einem vom Benutzer eingestellten Schwellenwert. Beim Überschreiten des Schwellenwertes (iL_peak) wird das *ipeak* Signal von einem Komparator gesetzt. Dieses Signal führt über den *brk* Eingang von $TIM15$ zum sofortigen Öffnen des Schalters ($mout = LOW$, blauer Pfad 1). Der Strom sinkt daraufhin mit der Steigung $\frac{vo}{L}$. Ein weiterer Komparator detektiert den Nulldurchgang des Stromes und setzt das *zcd* Signal. Dieses Signal setzt den Zählwert

des Timers $TIM2$ auf Null zurück und das Ausgangssignal auf LOW (blauer Pfad 2). Sobald der Timer $TIM2$ die Totzeit $tdead$ erreicht hat, wird der Ausgang auf $HIGH$ gesetzt. Dies führt zum Starten des Timers $TIM15$ (blauer Pfad 3) und zum erneuten Schließen des Schalters.

Der Benutzer hat die Möglichkeit, mittels Schiebereglern in Simulink die Totzeit ($tdead$) und den Spitzenwert des Spulenstromes (iL_ipeak) einzustellen. Diese Einstellungen werden über Parallel-In-Out (kurz PIO) IP Komponenten im Qsys-System zum Prozessor geführt. Mit diesen Einstellungen setzt der Prozessor Register Timer und berechnet Vergleichsspannungen, die über ein PIO und die LE geschickt werden.

ii. Messergebnisse

Abbildung 9 zeigt bei verschiedenen Einstellungen des iL_ipeak und der $tdead$ den Ausgangsstrom des Abwärtswandlers während eines Simulationsdurchlaufes ($15ms$). Die Zahl vor dem Schrägstrich repräsentiert den gesetzten iL_ipeak Wert und die Zahl nach dem Schrägstrich stellt die eingestellte Totzeit dar.

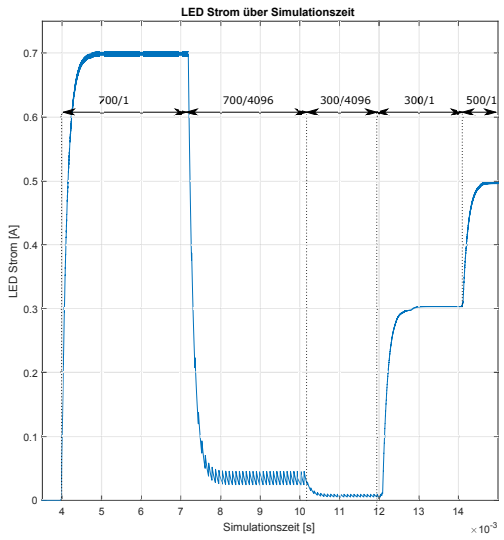


Abbildung 9: *Ausgangsstrom*

Der Ausgangsstrom beginnt nach $4ms$ zu steigen, da der Prozessor eine bestimmte Startprozedur durchläuft, bevor die ersten Befehle der Software (Setzen der Register der Timer und Aktivieren der Komparatoren) ausgeführt werden kann.

Der Mittelwert des simulierten Ausgangsstromes ($iled\_avg$) stimmt mit dem berechneten Wert aus Gleichung 1 (abgewandelt aus [11]) und den Erwartungen überein.

$$iled\_avg = \frac{iL\_ipeak}{2} * (1 - \frac{tdead}{Ts}) \qquad (1)$$

ii.1 Variation der Taktfrequenz

Um die Variation der Taktfrequenz des Qsys-Systems darzustellen wurde der Eingangstakt des Systems (clk_in in Abbildung 6) in Bezug zur Realzeit aufgenommen und in Abbildung 10 dargestellt.

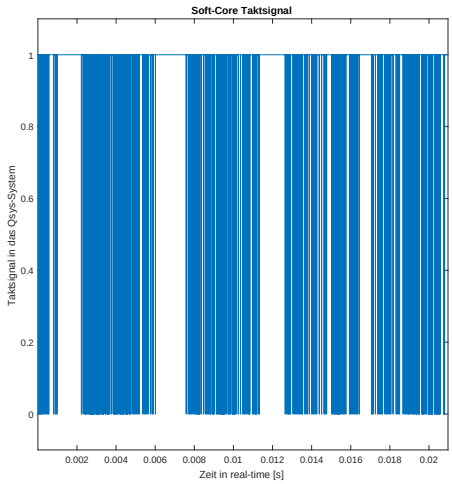


Abbildung 10: *Taktsignal des Qsys-Systems in Bezug zur Realzeit*

In Abbildung 10 erkennt man gut, dass bei einer nahezu konstanten Aus-Zeit des Taktes eine stark variierende Ein-Zeit (Signal ist bei 1 in Abbildung 10) am Qsys-System anliegt. Messungen ergaben, dass sich die Frequenz des Taktes innerhalb einer Bandbreite von $635Hz$ bis $41.1kHz$ befindet. Daraus ergibt sich ein Tastgrad im Bereich von 90% bis 99%.

V. Diskussion

Mit dem entwickelten Konzept konnten die gesteckten Ziele erreicht werden. Durch den verwendeten Prozessor (NIOS II/e) und dessen Entwicklungsumgebung (Eclipse) kann aber nicht ohne Aufwand ein beliebiges Cortex-M0 Projekt implementiert und getestet werden. Um den Workflow diesbezüglich zu verbessern, ist in einem nächsten Schritt die Umstellung des Prozessors auf den Cortex-M0 DesignStart Prozessor ([9]) angedacht. Verbesserungspotential besteht auch bei den analogen Peripherien, denn die implementiere analoge Peripherie kann aufgrund der fehlenden Detailtiefe nur bedingt für Toleranzanalysen verwendet werden.

Die Ergebnisse aus dieser Arbeit können genutzt werden, um weitere Simulationsaufbauten zu realisieren. Beispielsweise könnte die LE in einem Hard-Core ausgeführt werden (der Code kann beispielsweise mithilfe des PLECS-Coder aus dem PLECS-Modell erzeugt werden) und der Prozessor könnte mit einem Soft-Core im selben SoC simuliert werden.

Literatur

[1] Lentijo, S., A. Monti, E. Santi, C. Welch und R. Dougal: *A new testing tool for power electronic digital control*. In: *Power Electronics Specialist Conference, 2003. PESC '03. 2003 IEEE 34th Annual*, Band 1, Seiten 107–111 vol.1, Juni 2003.

[2] Jimenez, O., I. Urriza, L. A. Barragan, D. Navarro, J. I. Artigas und O. Lucia: *Hardware-in-the-loop simulation of FPGA embedded processor based controls for power electronics*. In: *2011 IEEE International Symposium on Industrial Electronics*, Seiten 1517–1522, Juni 2011.

[3] Lucia, O., I. Urriza, L. A. Barragan, D. Navarro, O. Jimenez und J. M. Burdio: *Real-Time FPGA-Based Hardware-in-the-Loop Simulation Test Bench Applied to Multiple-Output Power Converters*. Band 47, Seiten 853–860, März 2011.

[4] Liu, Chen, Xizheng Guo, Fei Gao, E. Breaz, P. Damien und F. Gechter: *FPGA based real-time simulation of high frequency soft-switching circuit using time-domain analysis*. In: *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Seiten 5832–5837, Oktober 2016.

[5] Majstorovic, D., I. Celanovic, N. D. Teslic, N. Celanovic und V. A. Katic: *Ultralow-Latency Hardware-in-the-Loop Platform for Rapid Validation of Power Electronics Designs*. Band 58, Seiten 4708–4716, Oktober 2011.

[6] Blanchette, H. F., T. Ould-Bachir und J. P. David: *A State-Space Modeling Approach for the FPGA-Based Real-Time Simulation of High Switching Frequency Power Converters*. Band 59, Seiten 4555–4567, Dezember 2012.

[7] Abourida, Simon, Sebastien Cense, Christian Dufour und Jean Belanger: *Hardware-in-the-Loop Simulation of Electric Systems and Power Electronics on FPGA using Physical Modeling*. In: *International Conference on Power Engineering*, Seiten 775–780, Mai 2013.

[8] Cuyper, K. De, M. Osee, F. Robert und P. Mathys: *A digital platform for real-time simulation of power converters with high switching frequency*. In: *Proceedings of the 2011 14th European Conference on Power Electronics and Applications*, Seiten 1–10, August 2011.

[9] Burr, Phil und Tim Menasveta: *An introduction to ARM Cortex-M0 DesignStart*. Technischer Bericht, ARM, März 2017.

[10] Yiu, Joseph: *Designing a System-on-Chip (SoC) with ARM Cortex-M processor - A starter guide*. Technischer Bericht, ARM, April 2017.

[11] Erickson, Robert W. und Dragan Maksimovic: *Fundamentals of Power Electronics*. Springer, 2012.