Einbindung einer FPGA basierten Motorregelung in ein industrielles Automatisierungssystem

Oliver Stefan - Dirk Rabe
Hochschule Emden-Leer
{oliver.stefan | dirk.rabe}@technik-emden.de

1 Abstract

Eine Einbindung eines leistungsfähigen Bauteils in ein industrielles Automatisierungssystem kommt zum Einsatz, um zeitkritische Regelungs-, Steuerungs- und Automatisierungsaufgaben zu übernehmen. Solche Aufgaben treten in der heutigen Praxis immer häufiger auf, da viele Prozesse immer schneller ablaufen und die Genauigkeitsanforderungen gleich bleiben oder steigen.
Solche Aufgaben können oft nicht von einer sequentiell arbeitenden, industriellen Steuerung verarbeitet werden, da die Reaktionszeiten zu groß sind. Abhilfe können hier FPGAs schaffen, die über parallel verarbeitende Logik verfügen. Somit können diese auch den sehr harten Echtzeit-Anforderungen genügen. Des weiteren stellen die frei konfigurierbaren Logikelemente eines solchen FPGAs eine sehr flexible Basis dar. Es können sowohl eigene Schaltungen implementiert, als auch vorgefertigte Hardwareschaltungen (z.B. CPUs) verwendet werden. Allerdings besitzen FPGAs keine standardisierte Schnittstelle zur Verknüpfung mit einem industriellen Automatisierungssystem.
Neben dieser Brücke zwischen den unterschiedlichen Systemen wird auch eine Beispielanwendung eines solchen Systems in dieser Arbeit vorgestellt.
Die Verbindung von FPGA und Automatisierungssystem ist über den CAN Bus realisiert worden.
Die Beispielanwendung stellt eine Linearachse dar, die durch einen Schrittmotor angetrieben wird. Die Generierung der Signale für den Schrittmotor und die Verbindung mit dem CAN Bus ist als Hardware im FPGA implementiert. Das Automatisierungssystem übernimmt übergeordnete Aufgaben und stellt dem Benutzer eine visuelle Kommunikationsmöglichkeit (Touchscreen) bereit.

2 Notwendigkeit des Einsatzes neuer Technologien im Automatisierungsumfeld

Immer zeitkritischer werdende Automatisierungsaufgaben fordern spezielle Lösungen, die sich nach Möglichkeit direkt in ein bestehendes Automatisierungssystem einbinden lassen und eine sehr hohe Verarbeitungsgeschwindigkeit besitzen. Des weiteren wird eine hohe Flexibilität von Automatisierungsgeräten gefordert, da die Produktlebenszyklen immer kürzer werden.

Die Einbindung eines solchen hoch flexiblen, sehr schnellen Gerätes ist die Aufgabenstellung dieses Projektes, welches an der Hochschule Emden-Leer durchgeführt wurde.

Durch eine ungebundene Struktur und die Möglichkeit der Parallelverarbeitung ist ein FPGA eine gute Basis für die Einbindung in ein Automatisierungssystem. In einem FPGA lassen sich sowohl komplette Steuerungen und Regelungen in Hardware ausführen, als auch Softcores[1] für die flexible Programmierung mit Hochsprachen implementiert. In diesem Projekt stellt ein Altera Cyclone II FPGA [Cyc2] - verbaut auf einem DE2 Board[2] - die Basis der Entwicklung dar. Allerdings ist durch die Anwendung von Hardwarebeschreibungssprachen zur Schaltungsimplementierung die Kompatibilität mit nahezu allen anderen FPGA-Systemen gewährleistet.

Das Automatisierungsgerät wird in diesem Projekt durch einer Siemens S7-1200 SPS [spss7] dargestellt. Dieses Gerät besitzt eine PROFINET Schnittstelle, über die es parametriert und programmiert wird. Des weiteren dient diese Schnittstelle zur Kommunikation mit eventuell vorhandenen, anderen Automatisierungssystem-Teilnehmern. In diesem Projekt ist die Bedienung der SPS über ein Touchscreen KTP-400 [tsktp] von Siemens realisiert worden. Dieses Gerät besitzt ebenso wie die SPS eine PROFINET Schnittstelle zur Programmierung und Kommunikation.

Für die Verbindung des FPGA mit der SPS sollte ein standardisiertes Bussystem zum Einsatz kommen, da somit die flexible Verwendung der einzelnen Komponenten in folgenden Projekten möglich ist. Die Wahl fiel auf den CAN Bus, da dieser sehr störunanfällig ist und sehr weit verbreitet ist. Leider besitzt

[1] CPU im FPGA als Hardwarebeschreibung
[2] Development and Education Board mit Altera Cyclone II FPGA von Terasic [de2bm]

die S7-1200 keine Möglichkeit direkt mit dem CAN Bus zu kommunizieren. Daher ist ein CAN-PROFINET Gateway[3] zum Einsatz gekommen.
Am FPGA kommt der weit verbreitete CAN Controller MCP2515 mit nachgeschaltetem Transceiver MCP2551 zum Einsatz. Beide Chips werden von Microchip hergestellt und sind optimal aufeinander abgestimmt. Der CAN Controller wird über den SPI[4] Bus angesprochen und hat zusätzlich verschiedene Interrupt Ein- und Ausgänge zur optimierten Kommunikation mit dem SPI Master.

In der folgenden Abbildung 1 werden die Vernetzungstruktur und die Kommunikationswege zwischen den einzelnen Komponenten ersichtlich.

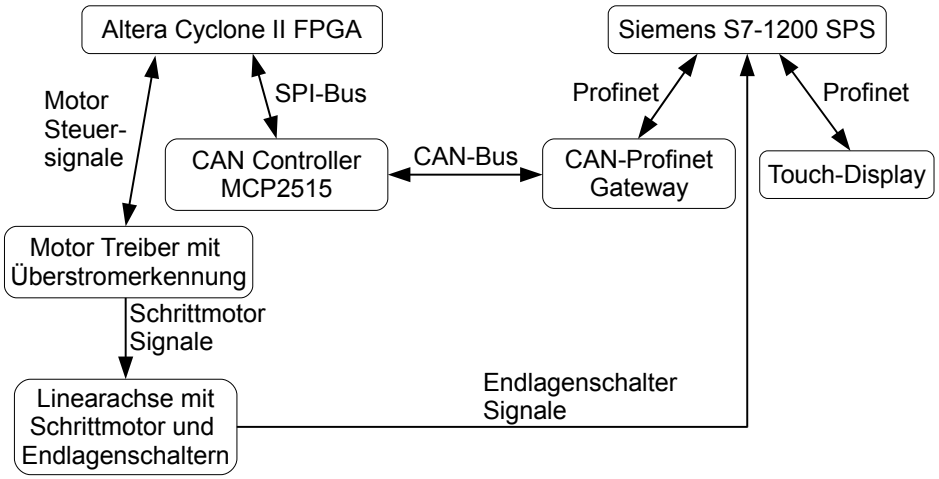

Abbildung 1: Die Vernetzung des Gesamtsystems

Des weiteren ist in der Abbildung 1 auch die Kommunikation mit der im Projekt realisierten Beispielanwendung, einer Linearachse, gezeigt. Diese Anwendung dient der Visualisierung der Kommunikation zwischen der SPS und dem FPGA und steht nur Beispielhaft für eine große Anzahl von möglichen Anwendungen eines solchen Systems.

[3]Protokoll-Konverter
[4]Serial Peripheral Interface

3 CAN Anbindung am Cyclone II FPGA

Die Verknüpfung des FPGA mit dem CAN Bus erfolgt mittels des CAN Controllers MCP2515 von Microchip. Dieser übernimmt die Versendung und das Empfangen von CAN Nachrichten. Des weiteren ist es möglich mit diesem Chip die empfangenen CAN Nachrichten zu filtern und somit nur die benötigten Daten an das übergeordnete System weiterzuleiten. Der CAN Controller hat insgesamt drei Sende Puffer und zwei Empfangs Puffer jeweils für die Zwischenpufferung einer Nachricht. Auch das gesamte, standardisierte CAN Bus Zugriffsverfahren CSMA/CA[5] ist in dem Chip implementiert.

Durch diese große Anzahl von Möglichkeiten ist es notwendig den CAN Controller bei Start des Systems zu initialisieren. Dazu ist im FPGA ein Hardware SPI Bus Master Controller implementiert worden. Dieser verfügt über die für die Kommunikation wichtigen Timing Informationen des SPI Bus.
Auch die Kommunikation mit den Sende- und Empfangspuffern des MCP2515 erfolgt über den SPI Bus. Zusätzlich ist eine Interrupt Leitung installiert worden (siehe Abbildung 2), die die schnelle Entnahme von empfangenen Nachrichten ermöglicht. Weiterhin ist eine schnelle Datenübertragung zwischen FPGA und CAN Controller durch die hohe SPI Busgeschwindigkeit von 1 MHz gewährleistet.

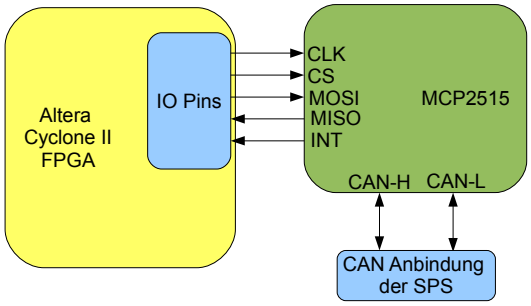

Abbildung 2: Die Verdrahtung des CAN-Controllers mit dem FPGA

Wie in Abbildung 2 zu sehen, werden für den SPI Bus 4 Leitungen und eine Interrupt Leitung, benötigt.
Der 1 MHz Takt zur Datenübertragung wird vom SPI Master erzeugt und über die CLK Leitung zum SPI Slave übertragen.

[5] Carrier Sense Multiple Access/Collision Avoidance

Da am SPI Bus mehrere Slaves angeschlossen sein können, ist es nötig eine Auswahl des Slaves zu treffen. Dieses verwaltet der Master über die CS[6] Leitung, die für jeden Slave des Bussystems eigenständig ausgeführt sein muss.
Die Daten in der Richtung vom Master zum Slave überträgt die MOSI[7] Leitung. Die entgegengesetzte Datenübertragung findet über die MISO[8] Leitung statt.
Somit kann über diesen Busaufbau eine Vollduplex-Verbindung aufgebaut werden, soweit die Busteilnehmer diesen Modus unterstützen.

Mit dem eingesetzten CAN Contoller mit Anbindung an das Cyclone II FPGA steht eine vollwertige CAN Schnittstelle zur Verfügung, worüber CAN Nachrichten mit variabler Größe und Identifier versendet und empfangen werden können.

4 CAN Anbindung an der S7-1200

Die Siemens S7-1200 besitzt keine CAN Anbindung. Daher kommt in diesem Projekt ein CAN-PROFINET Gateway des Herstellers ESD Electronics zum Einsatz. Dieses Gerät besitzt zwei PROFINET Schnittstellen und eine CAN Bus Schnittstelle. Parametriert wird das Gateway über die PROFINET Schnittstelle mithilfe der SPS Parametrierungs- und Programmierungssoftware [tia]. Dort kann das Gerät über eine GSD[9] Datei eingebunden werden. Für die Verwendung in Verbindung mit der SPS lässt sich das Gateway in den Netz-Manager der Software einladen. Mit einer virtuellen PROFINET-Leitung ist es möglich die SPS und das Gateway zu verbinden und somit zu einem Netz zusammenzuführen. Ab diesem Moment steht das Gateway in der SPS Software voll zur Verfügung und es können sogenannte Ein- und Ausgabemodule in das Gateway parametriert werden. Jedes Ein- oder Ausgabemodul steht für einen Nachrichtentyp bestimmter Länge und mit bestimmten CAN Identifier. Diese parametrierten Empfangs- und Versandnachrichten werden in das PAE und PAA[10] gemappt. Somit ist es im SPS Programm sehr einfach möglich auf die Empfangs- und Sendedaten zuzugreifen.

[6] Chip Select
[7] Master Output Slave Input
[8] Master Input Slave Output
[9] Gerätestammdaten
[10] Prozessabbild der Eingänge und Prozessabbild der Ausgänge

Da der CAN Bus standardisiert ist, reicht die Einstellung der Datenübertragungsrate für die Parametrierung dieser Schnittstelle aus. Die Nachrichtenbezogenen Parameter entnimmt die Software aus den Ein- und Ausgangsmodulen, die in der SPS Parametrierungssoftware dem Gateway beliebig hinzugefügt werden können.

5 Beispielanwendung: Linearachse

Zur Visualisierung der Kommunikationsfähigkeit der Systemteilnehmer sollte im Zuge des Projektes eine mögliche Anwendung realisiert werden. Die Ansteuerung eines elektromechanischen Antriebs ist eine gute Möglichkeit die Leistungsfähigkeit eines FPGA zu demonstrieren.
Zur besseren visuellen Wahrnehmung der Motoraktivitäten ist die Entscheidung auf eine Linearachse gefallen. Diese wird durch einen zweisträngiger Schrittmotor [Schörlin 1996] angetrieben und besitzt auf jeder Seite einen Endlagenschalter. Der Schrittmotor besitzt eine Schrittweite von 3,75° und wird mit bipolarer Ansteuerung bestromt. Durch eine Untersetzung von 10:1 erlangt der Motor ausreichend Kraft zur Beschleunigung der Linearachse.

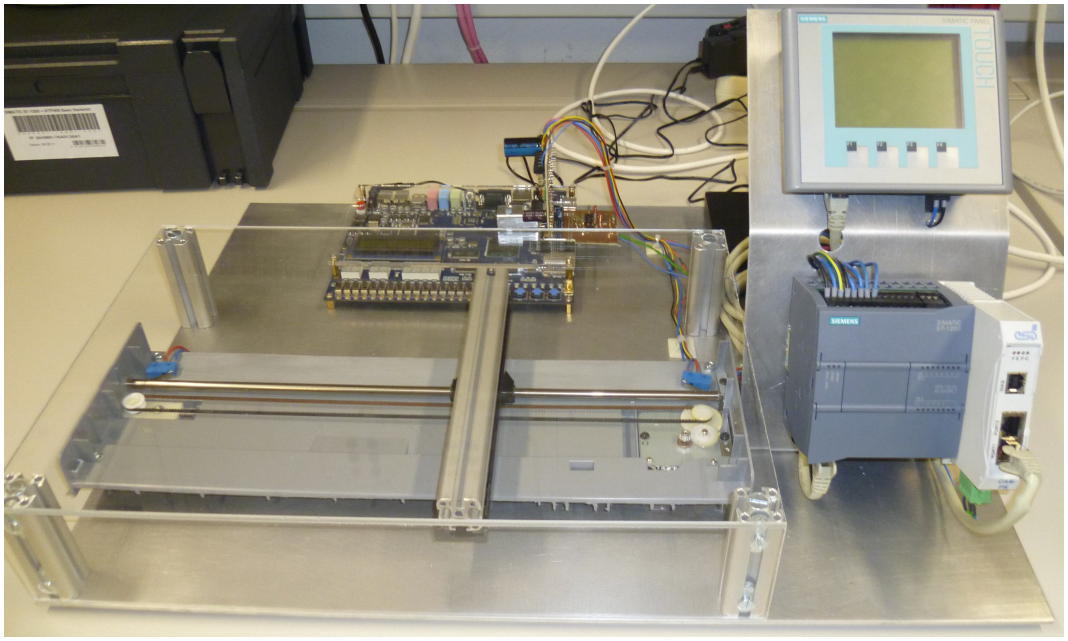

Abbildung 3: Die Linearachse (im vorderen Teil)

6 Schrittmotorsteuerung im Cyclone II FPGA

Die Ansteuerung des Schrittmotors ist komplett in Hardware abgebildet. Dadurch ist die Ansteuerung sehr schnell und kann verschiedene Aufgaben gleichzeitig wahrnehmen.

6.1 Rampe zur Beschleunigung und Verzögerung

Beim Beschleunigen des Körpers ist zu beachten, dass die Beschleunigung einen gewissen Schwellwert nicht überschreitet, weil es sonst zu einem sofortigen Abreißen kommt. Deshalb muss für hohe Drehzahlen die Drehzahl rampenförmig hochgefahren werden. Die Rampenform gibt das Verhältnis der Beschleunigungszeit zu erreichter Geschwindigkeit an. Eine lineare Rampe lässt sich sehr einfach implementieren. Die Beschleunigungszeit ist hierbei relativ lang, da der Motor im unteren Drehzahlbereich viel schneller beschleunigen kann als im oberen. Daher wurde für dieses Projekt eine „Wurzel-förmige“ Rampe gewählt, bei der die Beschleunigung mit steigender Geschwindigkeit abnimmt. Durch diese Eigenschaft benötigt der Antrieb zum Erreichen niedriger Drehzahlen nur eine sehr kurze Zeit. Außerdem können mit einer solchen Rampenform wesentlich einfacher sehr hohe Drehzahlen erreicht werden.
In der Abbildung 4 ist die im FPGA implementierte Rampe zu sehen.

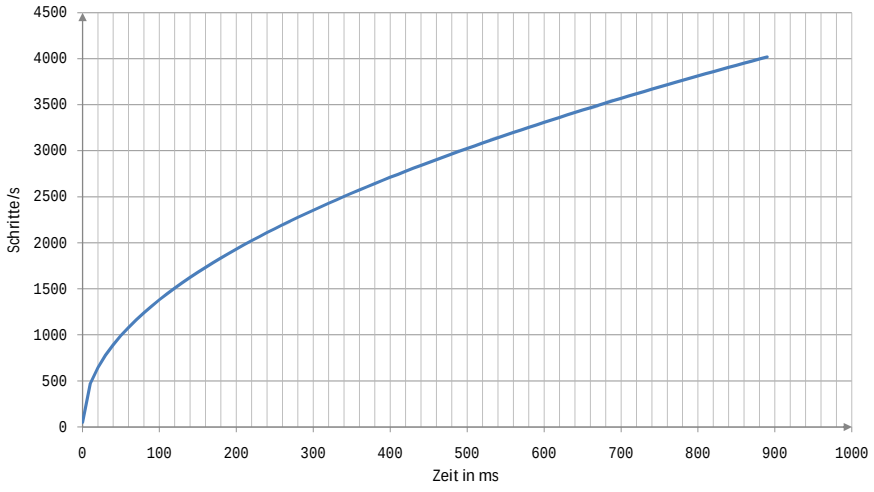

Abbildung 4: Die quadratische Beschleunigungsrampe

Mit dieser Rampe ist es möglich den Schrittmotor mit angebundener Linearachse auf maximal 3800 Schritte/s zu beschleunigen.

Die Verzögerung des Antriebs wird mithilfe der selben Rampe realisiert. Die Abbildung 5 zeigt ein Geschwindigkeits-Zeit-Diagramm, welches von der Schrittmotorsteuerung abgefahren wird, wenn eine Sollgeschwindigkeit von 3500 Schritte/s und die Schrittzahl mit 4350 Schritten vorgegeben wird.
In der Abbildung 5 kann die Sollgeschwindigkeit erreicht werden, da eine ausreichende Anzahl der zu verfahrenden Schritte vorgegeben ist.

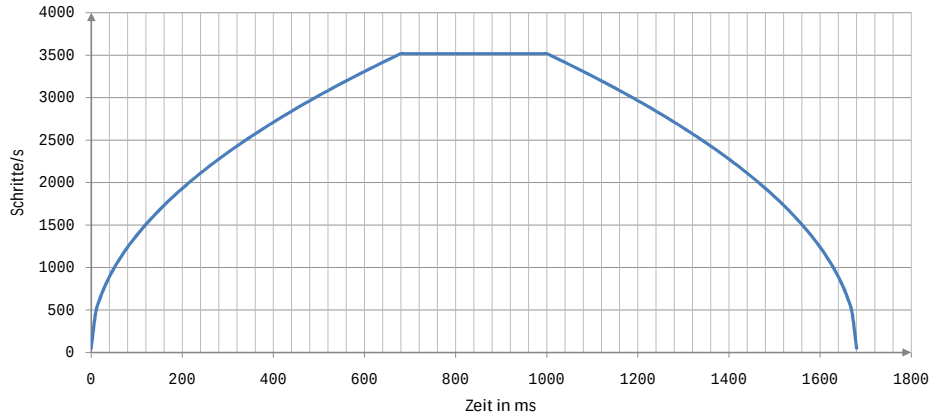

Abbildung 5: Geschwindigkeits-Zeit-Diagramm des Schrittmotors mit erreichter Sollgeschwindigkeit

Sollte die maximale Drehzahl bei der zurückzulegenden Strecke nicht erreicht werden, wird direkt aus der positiven Beschleunigung in die negative (Verzögerung) übergegangen.
Dieses Verhalten wird in der Abbildung 6 dargestellt. In diesem Fall sollten 1100 Schritte mit einer Geschwindigkeit von 3500 Schritte/s verfahren werden.

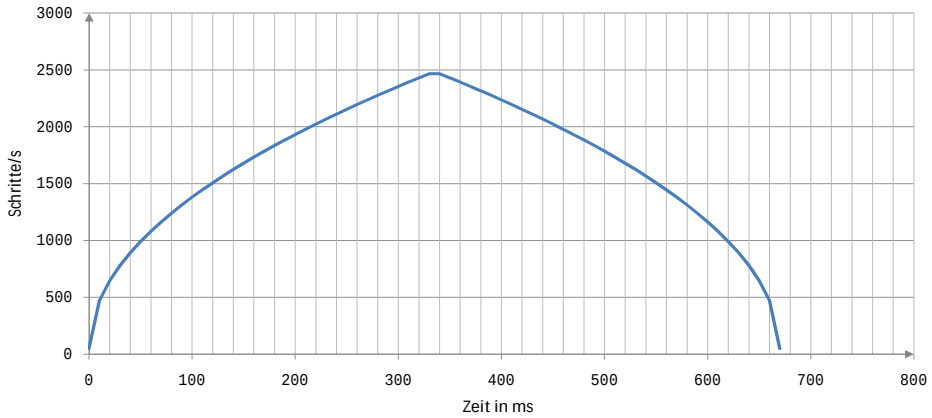

Abbildung 6: Geschwindigkeits-Zeit-Diagramm des Schrittmotors mit nicht erreichter Sollgeschwindigkeit

6.2 Hardwareaufbau der Schrittmotoransteuerung

Die Schrittmotorsteuerung wurde in Hardware realisiert. Dies bedeutet, dass kein (Hochsprachen-)Programm zum Einsatz kam, sondern das gesamte Verhalten der Schaltung in einer Hardwarebeschreibungssprache beschrieben wurde. Diese Beschreibung wird dann in eine lauffähige Hardware synthetisiert. In diesem Projekt ist die Hardwarebeschreibungssprache VHDL[11] zum Einsatz gekommen. Diese wird über das Synthese-Tool [quar2] von Altera in lauffähige Hardware übersetzt.
Durch diese Sprache kann jede Hardware erstellt werden, die vom Umfang in das FPGA geladen werden kann. Vorkonfigurierte Timer oder Zähler sind allerdings in einem solchen FPGA nicht zu finden. Solche, bei Mikrocontrollern und SPS selbstverständlichen Bausteine müssen in einer programmierbaren Hardware nach Bedarf implementiert werden.
Diese Umstände machen die Verwendung eines FPGA sehr komplex und aufwändig, aber auch sehr flexibel, da jede denkbare digitale Schaltung synthetisiert werden kann.
Für die Realisierung der Schrittmotoransteuerung mussten mehrere Zähler implementiert werden, die in diesem Fall auch teilweise als Timer angesehen werden können. Der Hauptbestandteil der Schaltung ist die Generierung eines Taktes für die einzelnen Schritte. Dieser Takt wird im Weiteren als **clk_step** bezeichnet.

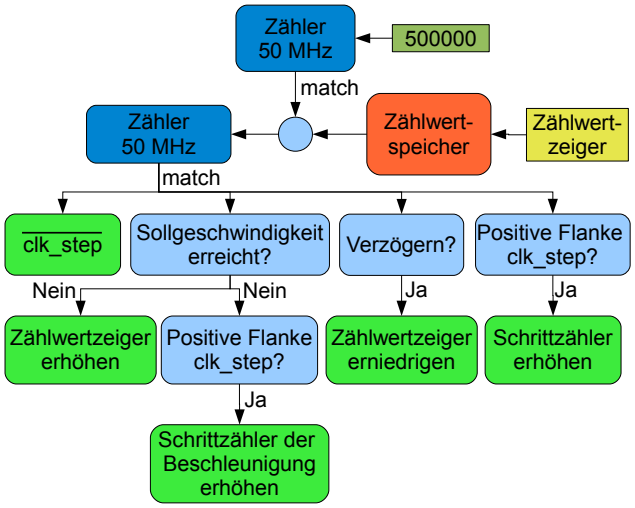

Abbildung 7: Schematische Darstellung der implementierten Schrittmotor Ansteuerung

[11] Very High Speed Integrated Circuit Hardware Description Language

Die Frequenz des clk_step Taktes ändert sich beim Beschleunigen oder Verzögern über die Zeit nach der in Abschnitt 6.1 beschriebenen Rampe.

In der Abbildung 7 sind zwei unabhängige Zähler gezeigt. Der obere ist als Timer verwendet, da dieser eine feste Frequenz erzeugt. In diesem Timer wird mit 50 MHz bis zum oberen Grenzwert von 500000 gezählt. Damit ergibt sich die Frequenz zu:

$$\frac{\text{oberer Grenzwert}}{\text{Zähler Taktfrequenz}} = \frac{500000}{50MHz} = 10ms \tag{1}$$

Somit wird nach jeweils 10 ms ein neuer Wert aus dem *Zählwertspeicher* geholt und als oberen Zählerwert in den zweiten Zähler geladen. Die Werte des *Zählwertspeichers* sind nach der Rampe in Abschnitt 6.1 generiert und als VHDL Konstanten in der Hardware Beschreibung realisiert.
Dieser zweite Zähler bestimmt die Frequenz der Schritte, die der Schrittmotor ausführen soll. Dieses geschieht durch die Invertierung des *clk_step* Signals.
In der Beschleunigungsphase wird der *Zählwertzeiger* nach jeweils 10ms inkrementiert, sodass die eine ansteigende Geschwindigkeit realisiert wird.

In der Beschleunigungsphase werden die Schritte über einen Schrittzähler mitgezählt. Dieses dient der Berechnung des korrekten Verzögerungszeitpunktes. Ist die Sollgeschwindigkeit des Motors erreicht, wird der *Zählwertzeiger* nicht weiterhin inkrementiert, sondern bleibt konstant auf dem bis dahin erreichten Wert. Ab diesem Zeitpunkt zählt der Schrittzähler für die Beschleunigung nicht mehr weiter mit. Ein anderer Zähler zählt allerdings die gesamten Schritte mit - auch wenn die Beschleunigungsphase abgeschlossen ist. Über diesen aktuellen Schrittzählerwert und die Angabe der Schritte, die für die Beschleunigung gebraucht wurden, kann der korrekte Zeitpunkt zur Einleitung der Verzögerungsphase berechnet werden (siehe obere Bedingung in Abbildung 8).

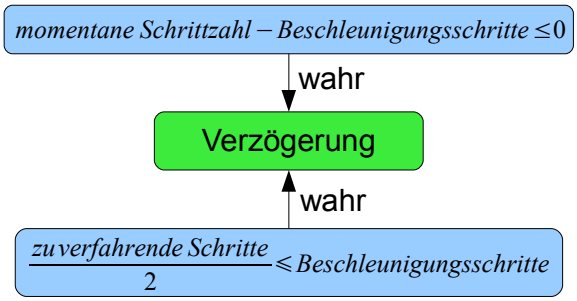

Abbildung 8: Bedingungen zur Einleitung der Verzögerungsphase

Ein Sonderfall liegt vor, wenn direkt aus der positiven Beschleunigung in die negative (Verzögerung) übergegangen wird (siehe Abschnitt 6.1 Abb. 6). Die untere Bedingung in Abbildung 8 fängt diesen Fall ab. Mit dieser Bedingung ist es möglich die Verzögerungsphase einzuleiten, bevor die Sollgeschwindigkeit erreicht wird.

Ist eine der beiden Bedingungen aus Abbildung 8 gegeben, wird die Verzögerungspahse eingeleitet und, wie in Abbildung 7 gezeigt, der *Zählwertzeiger* nach jeweils 10 ms dekrementiert.
Somit wird die Drehzahl spiegelsymmetrisch zum Hochfahren wieder heruntergefahren. Daher benötigt der Motor in der Verzögerungsphase auch exakt so viele Schritte wie in der Beschleunigungsphase.

Mit dieser Hardware ist die Taktgenerierung für die Schritte abgeschlossen. Es soll an einem kleinen VHDL Code Ausschnitt ein kleiner Einblick in eine solche Hardwarebeschreibungssprache gegeben werden.
In der Abbildung 9 ist der erste Zähler aus Abbildung 7 mit dem Signal *time_delay_counter* zu finden. Des weiteren ist auch die Zuweisung des *Zählerspeicher* Wertes und die Abfrage der Erreichung der Sollgeschwindigkeit abgebildet. Des weiteren sind diverse Abfragen vorhanden, da alle Signale parallel verarbeitet werden und somit ein wechselseitiger Ausschluss einiger Abfragen erfolgen muss.

```
entity stepper_driver is
  port (
    clk_i   : in  std_ulogic;
    reset_i : in  std_ulogic;
    ... );
end stepper_driver;

architecture behav of stepper_driver is
begin  -- behav

 stepper: process (clk_i, reset_i)
  begin  -- process
    if reset_i = '0' then               -- asynchronous reset (active low)
      ...
    elsif clk_i'event and clk_i = '1' then  -- rising clock edge
      case clk_gen_state is
        when idle =>
        ...
        when start_up =>
        ...
        when run =>
        ...
        when shut_down =>
        ...
        when others => null;
      end case;
    end if;
  end process;
```

Abbildung 9: Einblick in die Hardwarebeschreibungssprache VHDL

Aus dem, mit der beschriebenen Hardware generierten, Takt werden über eine weitere Hardware die Steuersignale für den Schrittmotor generiert. Diese kann für zwei unterschiedliche Ansteuerungsverfahren die korrekten Signale erzeugen. Für die exakte Positionierung ist das Halbschrittverfahren (siehe Abbildung 10) implementiert.

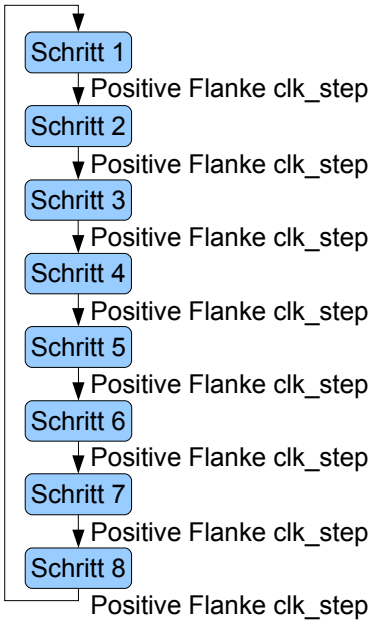

Abbildung 10: Signalgenerierung für den Halbschrittbetrieb

Da im Halbschrittverfahren zeitweise nur ein Strang des zweisträngigen Schrittmotors bestromt wird (siehe Abbildung 12), hat dieser in der Zeit nur das halbe Drehmoment. Dadurch kann nicht die maximale Geschwindigkeit erreicht werden.

Zur Vermeidung dieses Problems ist auch der Vollschrittbetrieb in die Schrittmotor Ansteuerung implementiert worden.

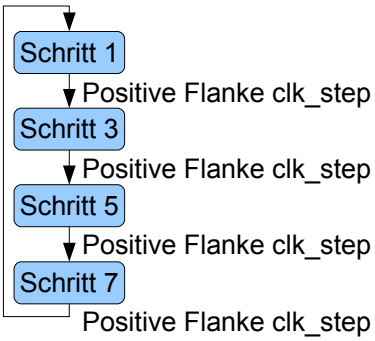

Abbildung 11: Signalgenerierung für den Vollschrittbetrieb

Im Vollschrittbetrieb (siehe Abbildung 11) sind immer beide Stränge voll bestromt und somit ergibt sich kein Einbruch des Drehmomentes.

Die Polarität der einzelnen Wicklungen in jedem Schritt sind in der Abbildung 12 eingetragen.

	Spule 1		Spule 2	
Schritt	Anschluss A	Anschluss B	Anschluss A	Anschluss B
1	+	-	+	-
2	+	-	0	0
3	+	-	-	+
4	0	0	-	+
5	-	+	-	+
6	-	+	0	0
7	-	+	+	-
8	0	0	+	-

- steht für Minus
+ steht für Plus
0 steht für unbelegt

Abbildung 12: Bestromung der Wicklungen in den einzelnen Schritten

Mit dieser oben beschriebenen digitalen Hardware werden nun die vier Signale für den Schrittmotor generiert und von einer Verstärkerstufe verstärkt. Diese beinhaltet noch eine Stromüberwachung, sodass auch bei niedrigen Drehzahlen, bei denen der Strom sehr hoch werden würde, die Bestromung der Wicklung unterbrochen wird. Durch diesen Aufbau ist es möglich den Motor mit einer relativ hohen Spannung zu versorgen und dadurch ein großes Drehmoment zu erhalten.

7 Kommunikation der Systemteilnehmer

Die SPS ist in diesem System der Master, also der aktive Teilnehmer. Dieser kann eine Aktion bei dem Slave, also dem passiven Teilnehmer veranlassen. In diesem System ist das FPGA der Slave.
Zur Kommunikation dieser beiden Teilnehmer muss ein immer geltendes Verfahren eingeführt werden, an das sich beide Teilnehmer halten müssen. Dieses in diesem System implementierte Verfahren ist in Abbildung 13 dargestellt.

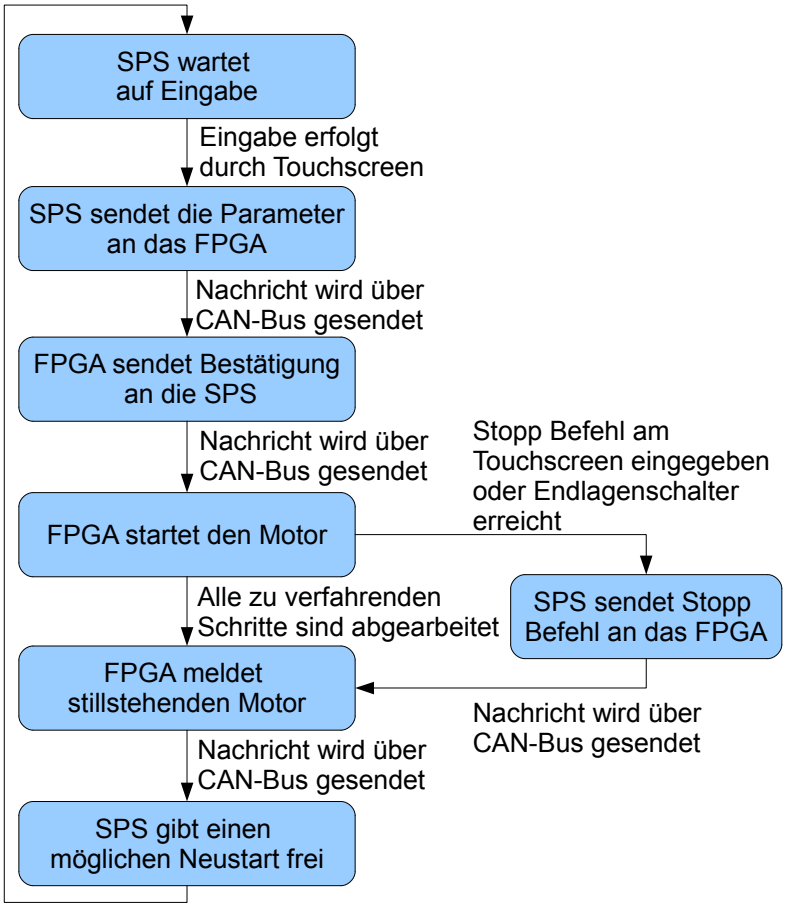

Abbildung 13: Der Kommunikationsablauf zwischen der SPS und dem FPGA

In dem Kommunikationsverfahren nach Abbildung 13 ist zu erkennen, dass eine Mitteilung an den Slave immer quittiert werden muss. Dies dient der Verhinderung von Verklemmungen und sorgt für eine sichere Überprüfung des aktuellen Slave-Status.
Außerdem ist auch die Möglichkeit des sofortigen stoppens des Antriebs zu erkennen. Dieses ist zum Beispiel bei Erreichen eines Endlagenschalters notwendig.

8 Ergebnisse

Durch diese Arbeit konnte eine voll funktionsfähige Anbindung eines FPGA an eine SPS-basierte Automatisierungsplattform erfolgreich umgesetzt werden. Durch die gewählten Kommunikationsnetze CAN Bus und PROFINET sind industrielle Standards eingehalten worden.
Die Beispielanwendung einer Schrittmotor-angetriebenen Linearachse konnte ebenfalls vollständig realisiert werden. Durch diese Anwendung kann die Kommunikation zwischen Automatisierungsgerät und FPGA visualisiert werden.
Die Implementierungen der Funktionen im FPGA haben die hohe Komplexität eines solchen Hardwareentwurfs aufgezeigt. Im Gegensatz zu höheren Programmiersprachen (C++, Java...) wurden alle grundlegenden Strukturen selber erstellt. Dadurch können in einem FPGA jedoch wesentlich zeitkritischere Funktionen implementiert werden als z.B. auf einem Mikrocontroller/SPS.

9 Ausblick

Diese Arbeit wurde vor dem Hintergrund der Ausbaufähigkeit erstellt. Die gewählten Kommunikationsnetze CAN Bus und PROFINET sind beide sehr einfach erweiterbar, sodass mehrere Teilnehmer beider Systeme untereinander kommunizieren könnten. Es ist also möglich, über nur ein CAN-PROFINET Gateway mehrere Automatisierungssysteme mit mehreren FPGAs zu verbinden.
Des weiteren ist die Linearachse nur eine mögliche Beispielanwendung eines solchen Systems. Bedingt durch die hohe Flexibilität des FPGAs können jegliche Schaltungen oder auch Softcores implementiert werden. Somit steht der Realisierung verschiedenster Aufgaben wie z.B. optischer Mustererkennung, Raumzeigermodulation von Asynchrommaschinen usw. nichts im Wege.

Literaturverzeichnis

[de2bm]

ALTERA (Hrsg.): *DE2 User Manual*. 1.42. Altera, http://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=English&No=30&FID=ab73908ea64e51be175534c8101942b7

[Cyc2]

Altera: *Allgemeine Daten über das Cyclone II FPGA*. http://www.altera.com/devices/fpga/cyclone2/cy2-index.jsp. – zuletzt abgerufen am 14.01.2012

[quar2]

Altera: *Quartus II Web Edition Software*. https://www.altera.com/download/software/quartus-ii-we. – zuletzt abgerufen am 14.01.2012

[Schörlin 1996]

SCHÖRLIN, Felix: *Mit Schrittmotoren steuern, regeln und antreiben*. Franzis, 1996

[spss7]

SIEMENS (Hrsg.): *S7-1200 Easy Book*. 11/2011. Siemens, http://support.automation.siemens.com/WW/llisapi.dll/csfetch/39710145/s71200_easy_book_de-DE_de-DE.pdf?func=cslib.csFetch&nodeid=39710146&forcedownload=true

[tsktp]

SIEMENS (Hrsg.): *SIMATIC HMI Panels*. 2011. Siemens, http://www.automation.siemens.com/salesmaterial-as/brochure/de/brochure_panels_de.pdf

[tia]

SIEMENS (Hrsg.): *Totally Integrated Automation Portal*. 2010. Siemens, http://www.industry.siemens.com/topics/global/en/tia-portal/Documents/e20001-a340-p230.pdf