Bachelorthesis

Andreas Arvidsson

Analog-Interface-Modul für einen FPGA-Prototypen der Signalverarbeitung bei ABS-Sensoren

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Andreas Arvidsson

Analog-Interface-Modul für einen FPGA-Prototypen der Signalverarbeitung bei ABS-Sensoren

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
Studienrichtung Informationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Karl-Ragmar Riemschneider
Zweitgutachter: Prof. Dr.-Ing. Franz Schubert

Abgegeben am 28. Februar 2011

Andreas Arvidsson

Thema der Bachelorthesis

Analog-Interface-Modul für einen FPGA-Prototypen der Signalverarbeitung bei ABS-Sensoren

Stichworte

ADC, DAC, FPGA, RS232-Interface, ABS-Sensor, VHDL, Signalverarbeitung

Kurzzusammenfassung

Diese Bachelorthesis befasst sich mit der Hardwareentwicklung eines Moduls, das die analogen Signale eines ABS-Sensors in digitale Werte konvertiert. Parallel wird mittels eines FPGAs ein Vorverstärker-Modul angesteuert und die Signale der ABS-Sensoren im FPGA weiterverarbeitet. Weiterhin besteht eine Kommunikation zwischen dem PC und dem FPGA, sodass eine Steuerung und Datenauswertung mittels des PCs durchgeführt wird. Diese Bachelorthesis ist Teil von Forschungsarbeiten, die eine Sensoreigendiagnose für ABS-Sensoren zum Ziel hat. Sie steuert einen weiteren Schritt zur Chipimplementation eines neuartigen Verfahrens bei.

Andreas Arvidsson

Title of the paper

Analog-interface-module for a FPGA-prototype with signal processing of ABS-sensors

Keywords

ADC, DAC, FPGA, RS232-interface, ABS-sensor, VHDL, signal processing

Abstract

This bachelorthesis deals with the hardware development of a module that converts analogue signals of ABS-sensors into digital ones. At the same time a FPGA processes these converted signals and actuates an amplifier-module. The FPGA communicates with a PC. Thus the information can be analysed both in PC and FPGA. The FPGA is also controlled by the PC. This bachelorthesis is a part of research which intends to build a self-diagnosis for ABS-sensors to contribute to chipimplementation.

Danksagung

An dieser Stelle geht ein besonderer Dank an Herrn Prof. Dr.-Ing. Karl-Ragmar Riemschneider, der als betreuender Erstprüfer, für das Bereitstellen des Themas und der tatkräftigen Unterstützung bei Fragen und Problemen für diese Bachelorarbeit gute Arbeit geleistet hat.

Weiterer Dank gilt Herrn Prof. Dr.-Ing. Franz Schubert, der sich als Zweitgutachter zur Verfügung gestellt hat.

Dipl.-Ing. Martin Krey erhält einen besonderen Dank für seinen fachlichen Rat und seine tatkräftige Unterstützung während meiner Bachelorarbeit.

Besonderer Dank gilt meiner Familie für die erhaltene Unterstützung während meines Studiums.

Inhaltsverzeichnis

1 Einführung

ESZ-ABS ist die Bezeichnung für ein Projekt der HAW-Hamburg (Hochschule für Angewandte Wissenschaften) und steht für „Experimentelle digitale Signalverarbeitung und Zustandserkennung für ABS-Sensoren“. Es werden ABS-Sensoren auf ihre Spezifikationen hin getestet. Es besteht das Ziel, einen neuen besseren Controller für ABS-Sensoren zu entwickeln.

Die ABS-Sensoren befinden sich an jedem der vier Räder eines Autos. Sie sind eingegossen in ein Kunststoffgehäuse und messen die Magnetfeldänderung der Encoderräder. Jeder Zahn erzeugt einen sinusähnlichen Verlauf. Dieser wird digital ausgewertet. Im Rahmen des Projektes ESZ-ABS gibt es bereits verschiedene Diagnosefunktionen, die mit Hilfe von Microcontrollern durchgeführt werden.

Es wurden bereits mehrere unterschiedliche Messplätze entwickelt und für verschiedene Messungen eingesetzt. Dazu gibt es einen Kreuzspulenmessplatz, an dem die ABS-Sensoren auf magnetische Veränderungen getestet werden ([20] und [15]). Es werden in diesem Aufbau vier Spulen verwendet, die paarweise angesteuert werden. Weitere Messplätze sind RMP1[1], RMP2 und RMP3. Der letzte ist ein Präzisionsmessplatz (siehe [18]), der den Sensor zum Encoderrad auf $10\mu m$ genau heranfahren und den Sensor in sechs Achsen positionieren kann. Durch den geringen Abstand von $10\mu m$ werden hochauflösende Bauelemente benötigt, die sehr empfindliche Spannungen im mV-Bereich messen können. An den verschiedenen Messplätzen werden die Harmonischen bestimmt, um eine Aussage über die Signalqualität der ABS-Sensoren unter verschiedenen Bedingungen treffen zu können. Dazu zählen Veränderungen von Magnetfeldern und die Positions- bzw. Verkippungsänderungen von ABS-Sensoren.

Die abschließende Zielsetzung des Projektes ist die Chipimplementation. Hierfür sind die zuvor entstandenen Schaltungen und Controllerboards schrittweise durch diskrete Bauteile bzw. analoge Schaltkreise oder durch programmierbare Logik abzulösen. Zum einen wird dadurch den zukünftigen Schaltungsstrukturen auf dem Chip genähert und zum anderen werden wesentlich schnellere Verarbeitungsschritte erreicht.

[1] Radmessplatz1

Diese Arbeit beschäftigt sich mit der programmierbaren Logik und ist somit ein weiterer Schritt zur Näherung der Chipimplementation. Die vorherige Diplomarbeit von JH-Dreschhoff [10] behandelt die Berechnung des THDs[2] im FPGA.

Die programmierbare Logik wird in der Beschreibungssprache VHDL[3] erstellt.

1.1 Aufgabenstellung

Diese Arbeit befasst sich mit der Entwicklung einer Platine (ADC-Platine), mit dem die Signale des ABS-Sensors über das Vorverstärker-Modul von N.Jegenhorst [11] entgegengenommen und in digitale Werte umgewandelt werden. Es handelt sich dabei um ein Differenzsignal und zwei Halbbrückensignale. Weiterhin soll eine Offsetkompensation des Vorverstärker-Moduls durch einen DAC realisiert werden sowie für weitere andere Hardware-Komponenten wie dem Komparator.

Zusätzlich zu dem zu erstellenden Interface für das Vorverstärker-Modul wird ein Interface zur Kommunikation mit einem FPGA der Familie Spartan3E Nexys2 realisiert werden. Mindestens ein Komparator soll in seiner Funktion umgesetzt und ausgewertet werden.

Nach der Entwicklung soll die ADC-Platine in Betrieb genommen werden. Dies soll mit Hilfe mehrerer VHDL-Module umgesetzt werden. Dabei gilt es, die Signale vom Vorverstärker-Modul über den ADC zu konvertieren und diese sowohl in VHDL als auch in der Software Matlab zu bearbeiten und darzustellen.

Die verschiedenen Komponenten sind sowohl auf ihre Funktionen und Aufgaben mit ModelSim zu simulieren als auch in der Hardware mit Hilfe von Testsignalen und Logicanalyzer zu testen. Zu der Inbetriebnahme sollen ebenfalls eigene VHDL-Module entwickelt werden, die die einzelnen Komponenten auf ihre Funktionalität überprüfen.

Weiterhin soll ein Interface entwickelt werden, das die Kommunikation mit dem PC führt. Hier soll eine Auswertung und Darstellung der entgegengenommen Daten des ADCs stattfinden. Weiterhin soll es von Matlab aus möglich sein, Parameter und Einstellungen vorzunehmen, die im FPGA umgesetzt werden. Somit gilt es, eine Steuerung zu entwickeln, die das FPGA kontrolliert.

Es soll mit dieser zu entwickelnden Platine erreicht werden, die Verstärkung für das Vorverstärker-Modul zu bestimmen und diesem zu übergeben. Weiterhin wird das ADC-Board von einem FPGA gesteuert. Die konvertierten Daten des ADCs

[2] total harmonic distortion; dt. Gesamte harmonische Verzerrung
[3] Very-High Speed Integratet Circuit Hardware Description Language

werden im FPGA aufgenommen, ausgewertet und sollen an den PC gesendet werden. Hier sollen sie einer weiteren Verabeitung unterzogen und grafisch dargestellt werden.

Diese Funktionen und Entwicklungen sollen abschließend im Rahmen der Gesamtfunktion der Sensorsignalverarbeitung erprobt werden und somit den Funktionsnachweis dieser Arbeit liefern.

1.2 Grundlagen

Der AMR-Effekt, Anisotropischer Magnetoresitiver Effekt, beruht auf die Änderungen des von außen herrschenden Magnetfeldes. Sie beeinflusst ferromagnetische Metalle, wodurch sich der elektrische Widerstand ändert. Diese Änderung elektrischen Widerstandes wird genutzt, indem eine Wheatstonesche Messbrücke aus vier Widerständen eingesetzt wird [1]. Diese Messbrücke befindet sich im Sensorkopf. Da es bei passiven Encoderrädern keine Magnete gibt, befindet sich hinter dem Sensorkopf ein Dauermagnet. Die Feldlinien des Dauermagneten werden durch die Zähne des Encoderrads umgelenkt, die von der Messbrücke registriert werden. Es entstehen elektrische Ströme, aus denen drei Signale entnommen werden können. Es handelt sich dabei um zwei Halbbrückensignale und einem Differenzsignal (siehe Bild 1.1).

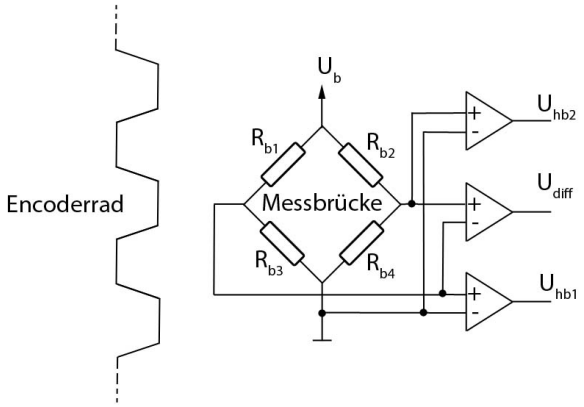


Bild 1.1: *Wheatstonesche Messbrücke*

Der AMR-Effekt weist eine große Nichtlinearität der Kennlinie auf (siehe Bild 1.2). Liegt das Signal im angenäherten linearen Bereich, ist das Signal sinusähnlich. Geht das Signal in den nichtlinearen Bereich über, wird das Ausgangssignal verzerrt. Um die Verzerrungen zu vermeiden, muss im angenäherten linearen Bereich gearbeitet werden. Dies wird dadurch erreicht, dass der Sensor in einem ganz bestimmten Abstand zum Encoderrad positioniert wird.

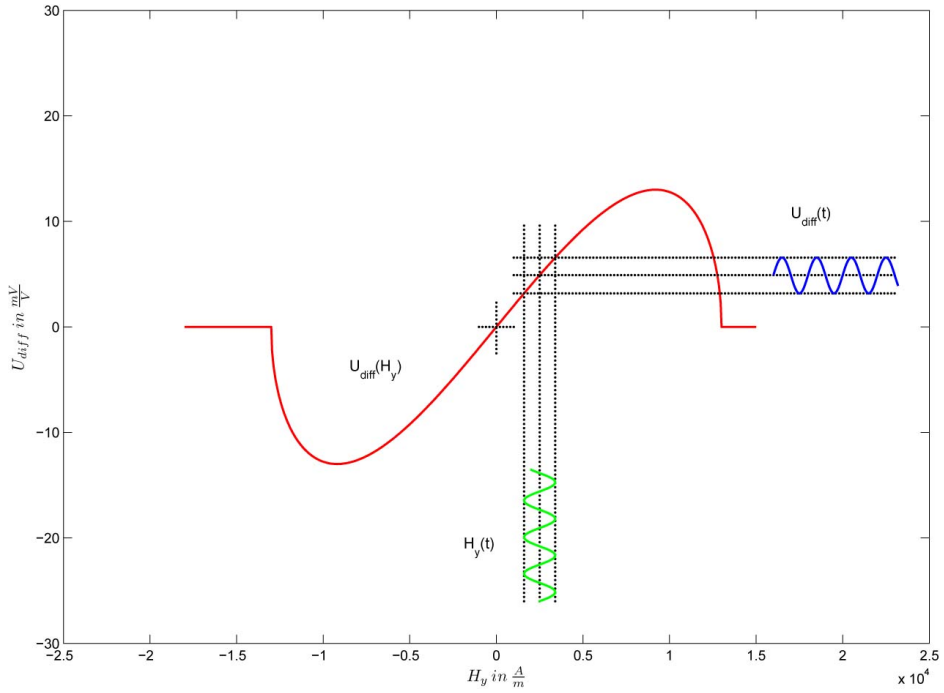


Bild 1.2: *Magnetisch-elektrische Kennlinie des AMR-Sensors im angenäherten linearen Bereich; entnommen aus [1]*

Über dieses Thema gibt es bereits mehrere Arbeiten, die den Sensor auf ihre Spezifikationen testen. Es werden die Harmonischen dazu berechnet, Untersuchungen unter verschiedenen magnetischen Änderungen sowie unterschiedlichen Anordnungen des Sensors in Bezug auf das Encoderrad unternommen.

Die Erkenntnisse sind bisher meist mit Microcontrollern gewonnen. Da für die Messungen und Untersuchungen nicht der ganze Sensor verwendet wird, sondern nur der Kopf selber, werden Funktionen benötigt, die das Protokoll auslesen und umsetzen können. Dazu dienen bisherige Microcontroller [11] oder auch die erstmals in VHDL implementierte Version [6].

Nichtideale Sinussignale, wie es bei den Sensorsignalen der Fall ist, besitzen nicht nur die Grundfrequenz sondern ebenfalls Oberschwinungen. Anhand der Grund- und Oberschwingungen werden die Harmonischen bestimmt. Dadurch lässt sich der THD (total harmonic distortion) berechnen. Dieser THD gibt Auskunft darüber, wie gut das aktuelle Signal ist.

Es werden weitere Untersuchungen unternommen, wie der Fall der Frequenzverdopplung [7], die Harmonischenanalyse bei magnetischen Winkelsensoren [15] oder die Fehlersichere Automatisierung eines Encoder-Messplatzes [12], welches laufende Arbeiten sind.

Wie bereits erwähnt werden die Messungen und Untersuchungen nur mit dem Sensorkopf durchgeführt, ohne den Controller zu benutzen. Ziel ist es, einen eigenen Controller zu entwickeln. Um diesen Schritt erreichen zu können, müssen mehrere Schritte durchlaufen werden. Der erste Schritt mit den Microcontrollern ist bereits im fortgeschrittenen Zustand.

Der nächste Schritt ist die Umsetzung in programmierbare Logik. Die erste VHDL-Implementierung gewonnener Erkenntnisse ist bereits umgesetzt [10]. Durch die programmierbare Logik lassen sich die Aufgaben und Berechnungen um ein Vielfaches beschleunigen. Es wird weiterhin die Kombination mit dem PC geben.

Weitere erreichte Erkenntnisse in diesem Projekt sowie zur Zeit bearbeitende Aufgaben siehe [14], [20], [11], [6], [19], [21], [13], [18], [10], [15], [7] und [12].

2 Hardware - Schaltungsaufbau

2.1 Analyse

In diesem Abschnitt soll auf die verschiedenen Rahmenbedingungen der zu entwickelnden Platine eingegangen werden, welche Hardware zur Verfügung steht und welche noch benötigt wird. Dazu gehören die gegebenen Hardware-Module.

2.1.1 Rahmenbedingungen

Die zu entwickelnde Platine soll sich vor dem FPGA befinden. Es sollen ein Differenzsignal und zwei Halbbrückensignale vom Vorverstärker-Modul entgegengenommen werden. Weiterhin ist eine Abtastrate des Signals von mindestens 64 Werten pro Schwingung gefordert. Für den Vorverstärker ist eine Offset-Kompensation von $1,25V$ notwendig.

Die Auflösung sollte $12bit$ betragen, um dem präzisen Messplatz, der in der Diplomarbeit C. Schörmer [18] entstanden ist, gerecht zu werden. Dieser positioniert den Sensor an das Encoderrad auf $10\mu m$ genau, was für das Ausgangssignal eine Auflösung im mV-Bereich erfordert.

Es soll vom FPGA ein Interface erstellt werden, das die Kommunikation mit dem PC ermöglicht.

Weitere Vorgaben für die ADC-Platine ist die Größe von $7x14cm$.

2.1.2 Hardware-Aufbau

Bei der Analyse der Hardware wird der gesamte Aufbau betrachtet (siehe Bild 2.1). Im kleinen roten Fenster ist das Vorverstärker-Modul auf der ADC-Platine um $180°$ gedreht dargestellt.

Das nächste Bild zeigt das dazugehörige Blockschaltbild. Es soll näher auf die einzelnen Hardware-Elementen eingegangen und näher beschrieben werden.

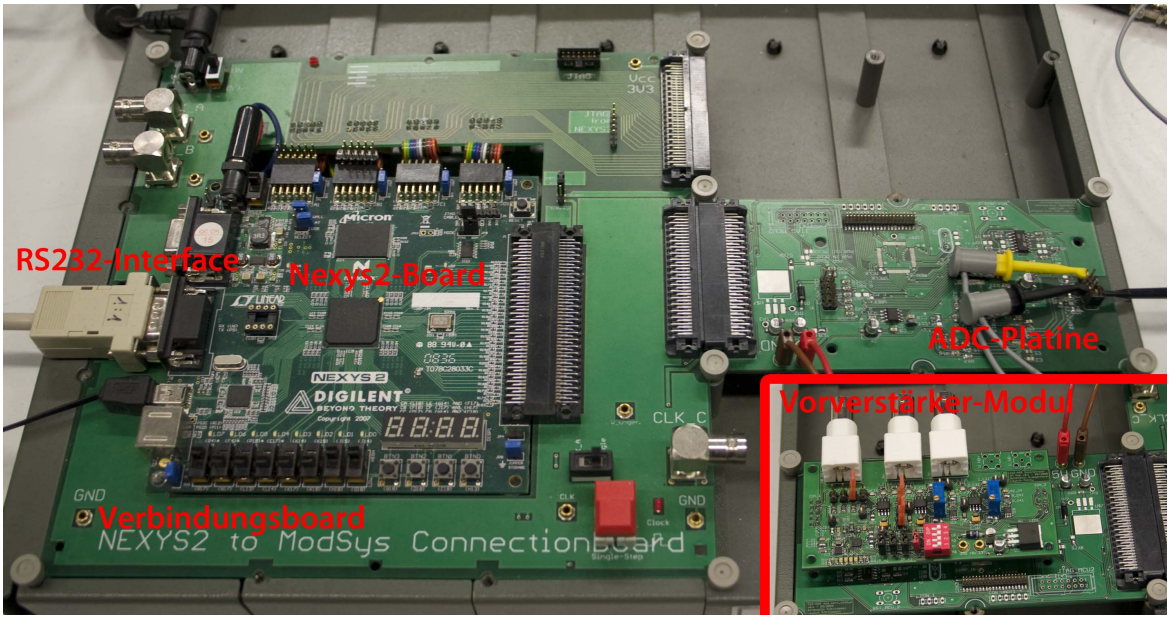


Bild 2.1: *Aufbau der gesamten Hardware; im roten Rahmen das Vorverstärker-Modul auf der ADC-Platine*

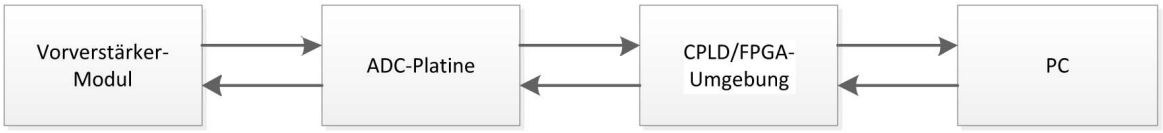


Bild 2.2: *Blockschaltbild der Hardware*

Wie aus dem Bild 2.2 zu erkennen ist, lässt sich der gesamte Aufbau in vier einzelne Elemente zerlegen. Das erste Element ist das *Vorverstärker-Modul*. Sie umfasst die Datenerfassung des ABS-Sensors und liefert die Sensordaten nach einer Verstärkung an das zweite Element. Dieses zweite Element entspricht der zu entwickelnden Platine (im weiteren *ADC-Platine* genannt). Sie nimmt die Daten des ersten Elements entgegen und konvertiert die analogen Signale in digitale Signale um und führt sie weiter zu dem dritten Element. Das dritte Element ist die *CPLD/FPGA-Umgebung*, die weitere Bearbeitungen und Auswertungen der Daten unternimmt. Dieses Element kommuniziert mit dem vierten Element, dem *PC*.

2.1.3 Vorverstärker-Modul

Bei dem Vorverstärker-Modul (siehe Bild 2.3) handelt es sich um eine Entwicklung aus einer früheren Diplomarbeit von N. Jegenhorst [11]. Die Aufgabe des Moduls war es, die Sensor-Signale entgegenzunehmen und diese durch eine Funktion zu schicken, der die Signale verstärken kann. Dies ist notwendig, da das Signal

schwächer wird, desto größer der Abstand zwischen dem Zahnrad (Encoderrad) und dem Sensor wird. Mit kleinen Signalamplituden sind Aussagen über den Zustand des Signals nur schwer zu ermitteln, weshalb eine Verstärkung vorgenommen wird.

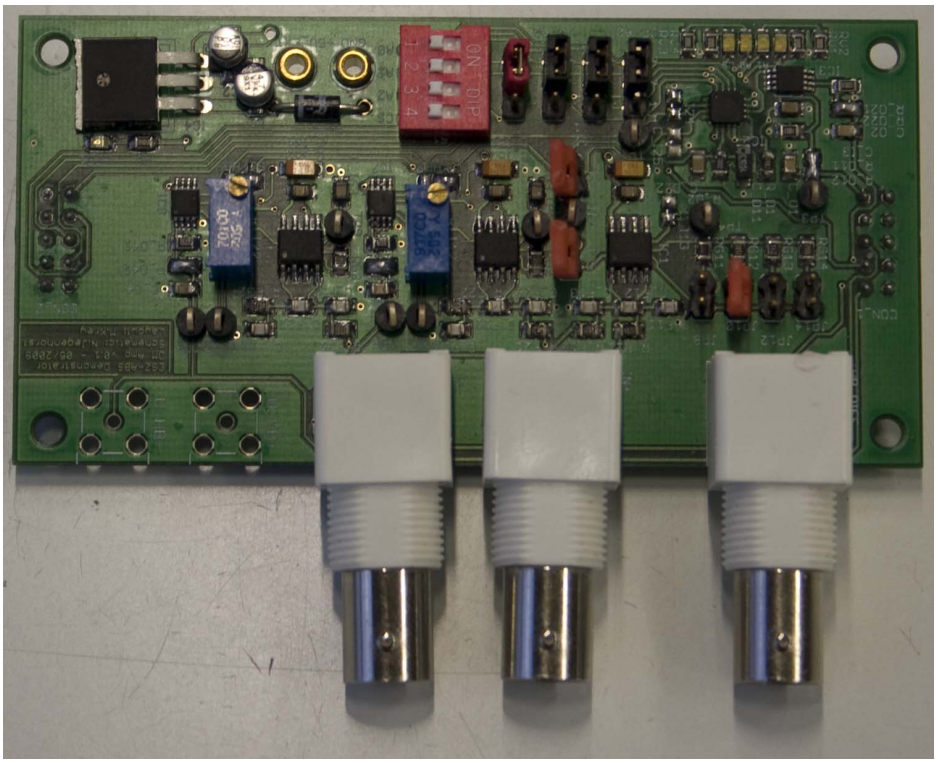


Bild 2.3: *Vorverstärker-Modul von [11]*

Die Steckverbindung zwischen dem Vorverstärker-Modul und der ADC-Platine sind vom Typ *Con-Amp*. Davon werden zwei benötigt, wie aus der Tabelle 2.1 zu entnehmen ist. Die Signale *U_DIFF*, *U_HB1* und *U_HB2* kommen vom Vorverstärker-Modul über den ersten Steckverbinder *Con-Amp1* und werden zur ADC-Platine geführt. Diese müssen dort entgegengenommen und ausgewertet werden (siehe 2.1.4). Das Signal *U_OFF* wird von der ADC-Platine an das Vorverstärker-Modul geführt. Es handelt sich dabei um die Offset-Kompensation der Signale vom Vorverstärker-Modul. Bei dem zweiten Steckverbinder befinden sich insgesamt vier Signale, die von der ADC-Platine gesteuert werden. Es geht um digitale Signale, die die Verstärkung der Signale des Verstärker-Moduls steuern. Diese Steuerung wird automatisiert und im FPGA mit Hilfe von VHDL realisiert. Die Verstärkung wird über die drei Signale *AMP_DB* eingestellt und mit dem Chip-Select-Signal gespeichert.

Pin-Nummer	Con-Amp1	Con-Amp2
1	U_DIFF	+5V
2	AGND	AMP_DB0
3	AGND	+5V
4	AGND	AMP_DB1
5	U_HB1	+5V
6	AGND	AMP_DB2
7	U_OFF	GND
8	AGND	AMP_CS
9	U_HB2	GND
10	AGND	GND

Tabelle 2.1: *Pinbelegung der Verbindungsstecker zum Vorverstärker-Modul*

2.1.4 Datenerfassung

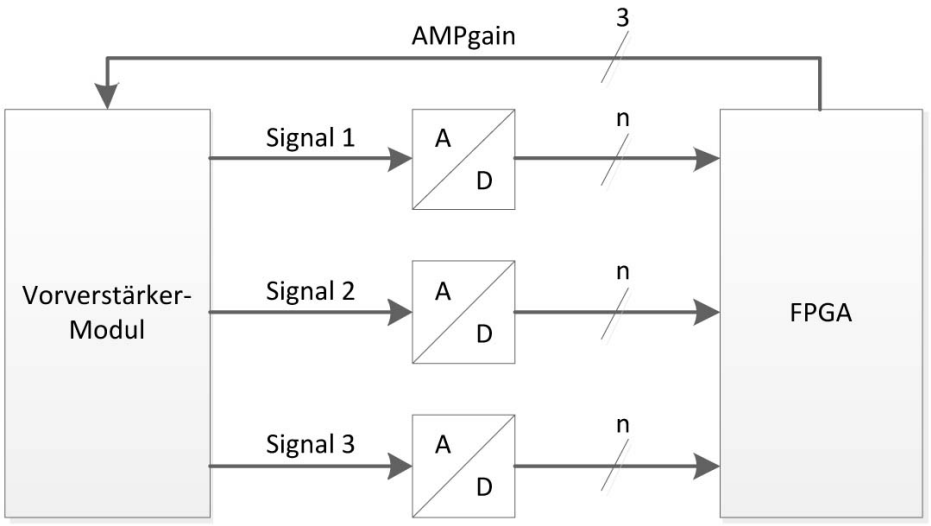


Bild 2.4: *Einsatz des ADCs in Kombination mit dem Vorverstärker und dem FPGA*

In Bild 2.4 ist der Einsatz des Vorverstärkers in Kombination mit ADC und FPGA zu erkennen. Das Vorverstärker-Modul liefert insgesamt drei Signale, die die Eingangssignale *U_DIFF*, *U_HB1* und *U_HB2* liefert. Die drei Signale werden einzeln durch je einen ADC geschickt, damit diese in digitale Signale bzw. Werte konvertiert werden. Die Ergebnisse werden dem FPGA übergeben, wo sie ausgewertet und bearbeitet werden. Die Auswertung und Bearbeitung im FPGA umfassen u.a. die Bestimmung der Verstärkung. Ist die Verstärkung zu groß, so wird sie hier mittels des 3-bit breiten digitalen Verstärkersignals *AMPgain* um eine Verstärkungseinheit verringert. Dieser Bus wird dem Vorverstärker-Modul übergeben. Das in

Bild 2.4 nicht aufgeführte Signal *AMP_CS* gibt dem Modul das Zeichen, wann die Verstärkung im Vorverstärker-Modul geändert bzw. aktualisiert werden soll. Sie ist hier immer auf '0', sodass der Wert sofort übernommen wird, sobald dieser geändert wird.

2.1.5 Periodendauer

Zur Bestimmung der Periodendauer des Differenzsignals wird eine Kombination aus Hard- und Software eingesetzt. Dazu wird ein Komparator-Signal benötigt. Die Auswertung und Berechnung des Komparator-Signals wird im FPGA erledigt. Da es sich um drei Signale handelt, werden mindestens drei Komparatoren eingesetzt.

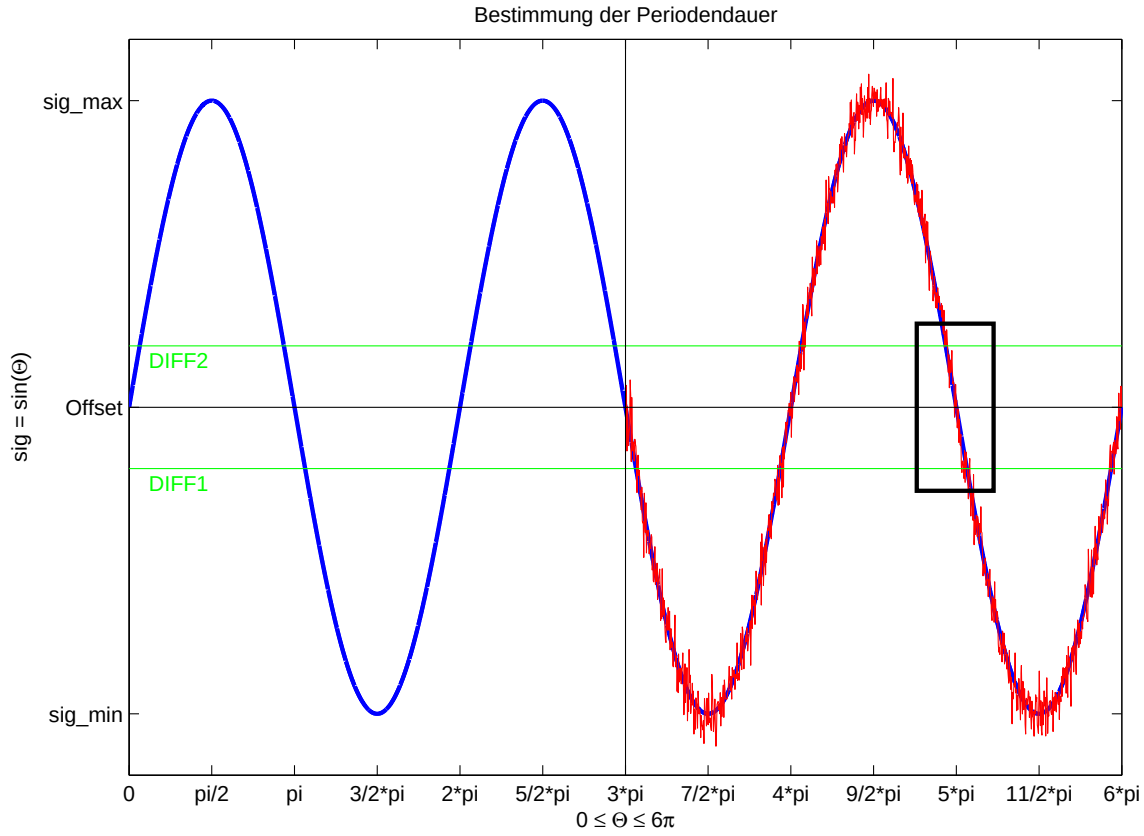


Bild 2.5: *Bestimmung der Periodendauer; links ideales Signal, rechts zur Verdeutlichung Rauschen hinzugefügt*

Für den Vergleich des Signals UDIFF mit den Schwellenspannungen DIFF1 und

DIFF2 werden zwei Komparatoren eingesetzt. Der Grund dafür sind die Abweichungen der konvertierten digitalen Werte, die vom ADC geliefert werden. Diese Abweichungen haben zur Folge, dass der Komparator nicht gleichmäßig ausschlägt, welches sich bei der Berechnung einer Periode bemerkbar macht. Dies kann kritische Folgen haben. Fehlerhafte Daten führen zu Auswertungen und Darstellungen, die kritische Auswirkungen auf weiterführende Steuerungen haben. Es muss sicher vermieden werden, dass das Bremssystem mit unsicheren bzw. fehlerhaften Sensor-Informationen versorgt wird.

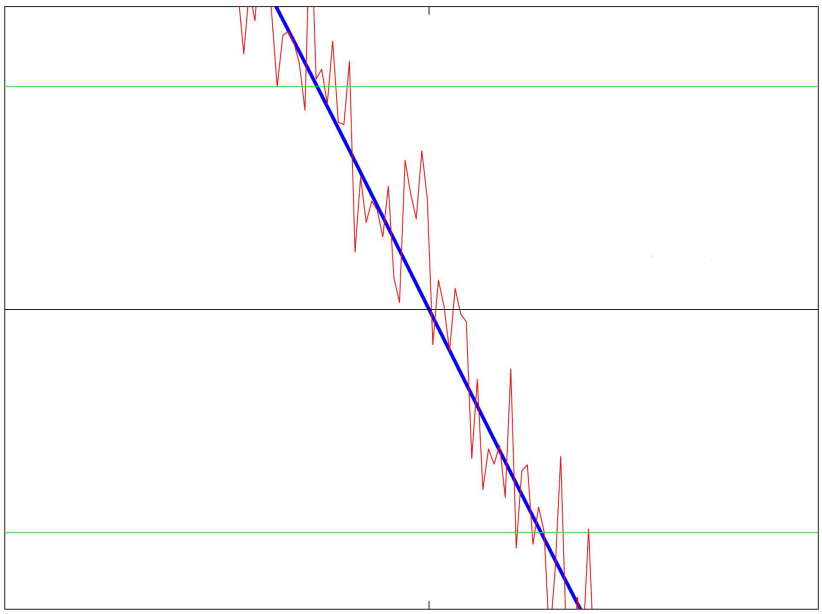


Bild 2.6: *Zoom der Periodendauer im Bereich der Schwellen*

Das Bild 2.6 verdeutlicht das Problem, wenn Rauschen auf dem Signal liegt.

Wird nur eine Schwellenspannung als Referenzspannung verwendet, kommt es zu Schwankungen des Komparatorsignals (siehe Bild 2.7). Es gibt vier Varianten, dieses Problem zu lösen:

1. Variante: Wird nur ein Komparator verwendet, so kann ein Zustandsautomat erstellt werden. Sobald der Komparator einmal umgeschaltet hat, dann muss eine gewisse Zeit vergehen, bevor sie wieder umschalten darf. Hier wird es mit der Bestimmung der Zeit schwierig, diese richtig zu dimensionieren.
2. Variante: Die zweite Variante ebenfalls mit einem Komparator-Signal ausgestattet, wobei hier das Signal entprellt werden muss. Dadurch muss sie eine gewisse Zeit konstant anliegen, damit sie als das entprellte Signal ’0’ oder ’1’ angesehen wird. Hier wird ebenfalls ein Zustandsautomat benötigt, damit das Hin- und Herschwingen zwischen den Signalen ’0’ und ’1’ verhindert wird.

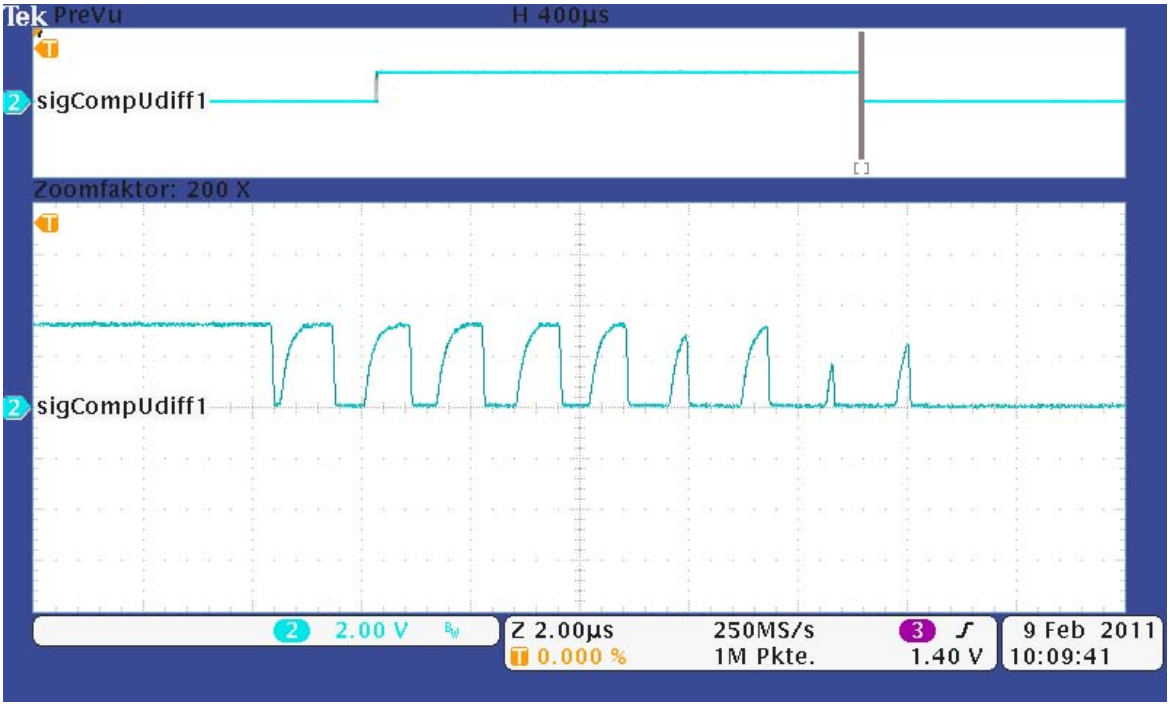


Bild 2.7: *Darstellung des Komparator-Signals, oberer Bereich: Komparator-Signal als ganzes, unterer Bereich: Zoom auf die fallende Flanke des Komparator-Signals*

3. Variante: Diese Variante könnte komplett in der Software bzw. in VHDL umgesetzt werden. Dazu würde der *Smart-Komparator* (siehe [10]) zum Einsatz kommen.

4. Variante: Es werden zwei Komparatoren verwendet, die auf zwei verschiedene Schwellenspannungen eingestellt sind (siehe Bild 2.5 die Spannungswerte *DIFF1* und *DIFF2* jeweils grün dargestellt). Es wird ein Zustandsautomat eingerichtet, der darauf wartet, dass beide auf '1' ausschlagen und in den nächsten Zustand wechselt. Wechseln beide auf '0' wechselt erneut der Zustand. Dies wiederholt sich ununterbrochen, solange ein Signal mit ausreichender Amplitude anliegt (siehe Abschnitt 3.2.3.2).

Die vierte Variante bietet eine optimale Lösung durch die Kombination eines Zustandsautomaten mit zwei Komparatorsignalen des Eingangssignals *U_DIFF*. Durch diese Lösung ist die Umsetzung stabil und sicher, da beide Komparatorsignale auf 'high' bzw. 'low' sein müssen. Weiterhin kann eine Entpreller-Funktion in der Hardware eingespart werden.

Die beiden übrigen Komparator-Signale *HB1* und *HB2* werden zur Zeit nicht verwendet. Sie können für die Bestimmung der Drehrichtung zum Einsatz kommen, auf die in dieser Arbeit nicht näher eingegangen wird.

2.1.6 Gegebene Hardware-Komponenten

Die ADC-Platine soll mit dem Modsys-Aufbau funktionieren können. Dies ist eine Entwicklung des Hauses der HAW. Es besitzt den CPLD „XC2C256 CoolRunner-II“ und besitzt spezielle Steckverbinder des Typs *Modsys-Connector*. Eine andere Diplomarbeit (siehe [10]) verwendet das Nexys2-Board mit dem FPGA Spartan3E. Da die Vorgabe das Nexys2-Board auch für diese Arbeit ist, wird ein Zwischenmodul gebraucht, das die Verbindung zwischen der ADC-Platine und dem Nexys2-Board herstellt, worauf in diesem Abschnitt eingegangen wird.

Nexys2-Board

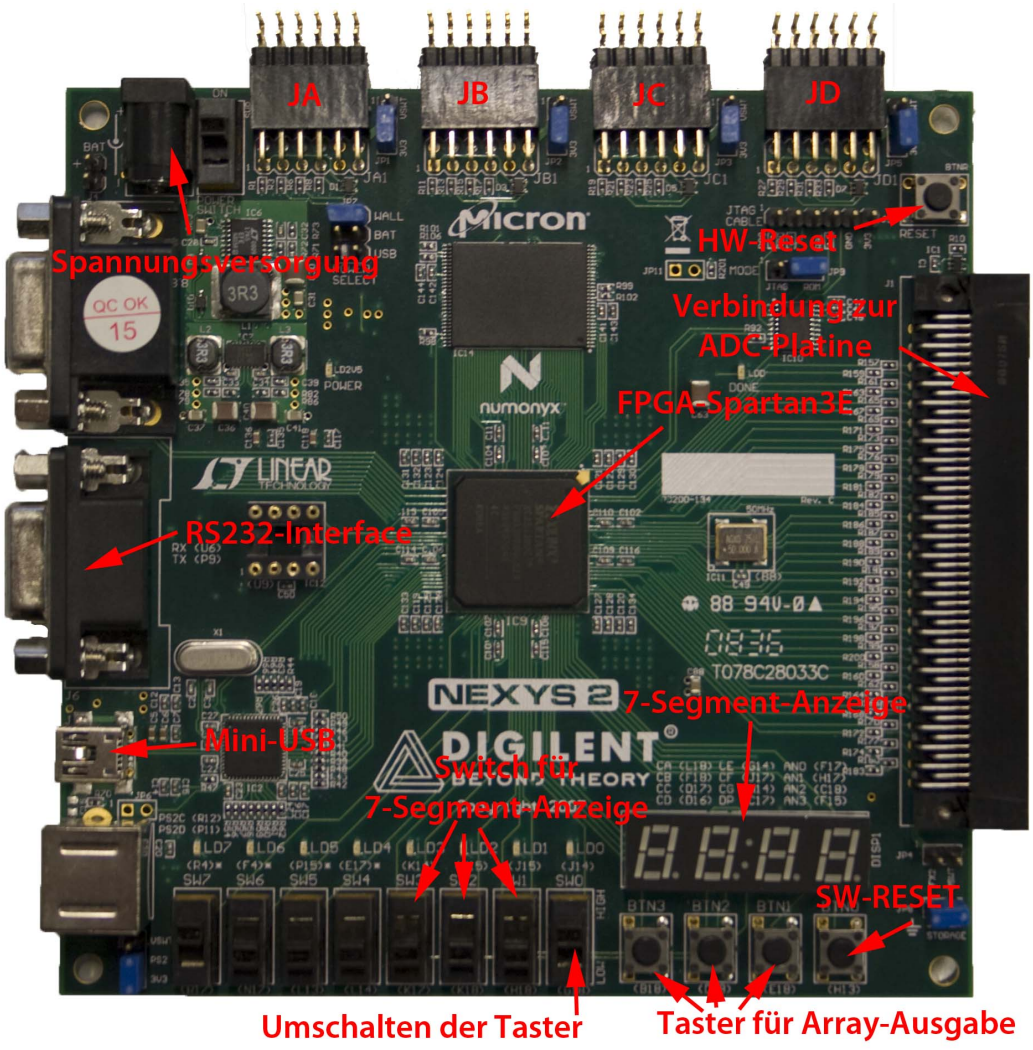


Bild 2.8: *Die eingesetzte FPGA-Umgebung: Das NEXYS2-Board mit dem Spartan3E-FPGA*

Bei dem *NEXYS2-Board* (siehe Bild 2.8) handelt es sich um einen von der Firma Digilent entwickelten Entwicklungsumgebung bezüglich FPGA-Programmierung. Es wird mit VHDL beschrieben.

- Systemtakt: $50MHz$
- 7-Segment-Anzeige mit 4 Anzeigen
- 8 Switches
- 4 Taster
- 1 Mini-USB
- RS232-Interfaces
- mehrere Steckverbinder für 2 Modsys-Platinen wie der ADC-Platine
- Spannungsversorgung

Für die Kommunikation zwischen der ADC-Platine und dem PC wird die RS232-Schnittstelle via des Nexys2-Boards verwendet. Hierfür muss im FPGA eine entsprechende Komponente erstellt werden. Auf nähere Einzelheiten siehe Abschnitt 3.2.2.

Verbindungsmodul

Bei dem Verbindungsmodul (siehe Bild 2.9) handelt es sich um eine Platine, die das Nexys2-Board mit der ADC-Platine verbindet. Diese ADC-Platine wird in dieser Arbeit entwickelt und näher beschrieben. Die Verbindungsplatine besitzt einen Anschlussstecker passend zum Nexys2-Board und zwei Modsys-Stecker der männlichen Variante.

2.1.7 Benötigte Bauelemente

Aus den oben genannten Angaben kann schlussgefolgert werden, dass mindestens folgende Bauelemente benötigt werden:

- 1x ADC mit mind. 3 Eingängen (Udiff, HB1 und HB2)
- 1x DAC mit mind. 4 Ausgängen (Offset-Kompensation und Komparatoren)
- mind. 3 Komparatoren
- 2x Steckverbinder vom Typ *Con-Amp* mit je 10 Pins (für das Vorverstärker-Modul)

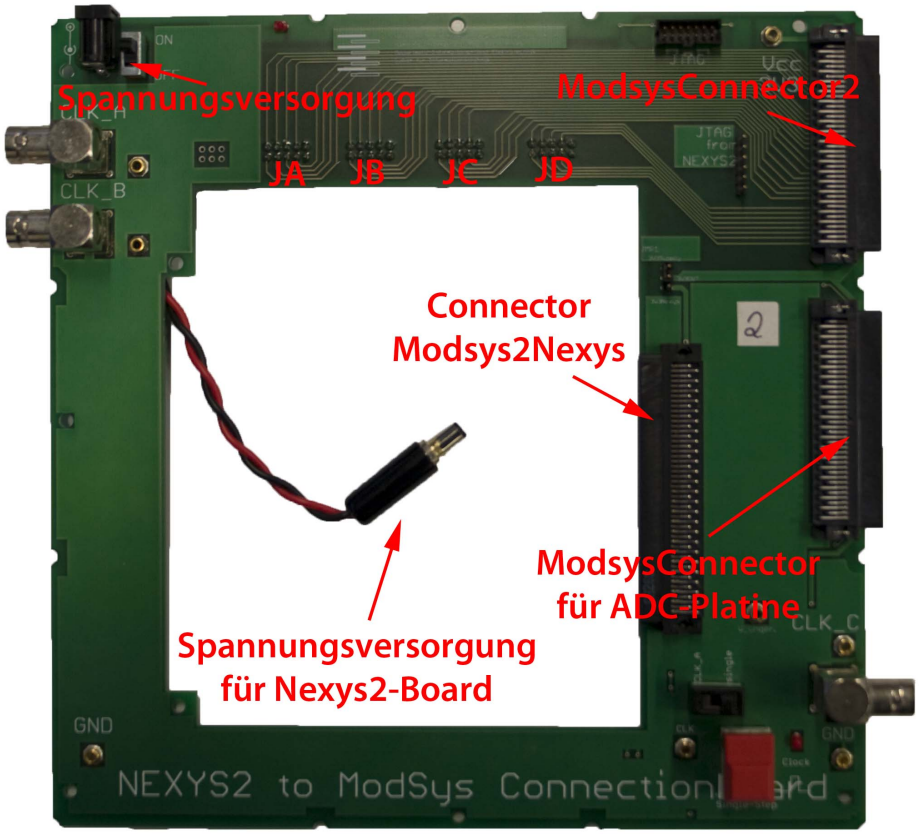


Bild 2.9: *Verbindungsmodul NEXYS2 To Modsys Connectionboard*

- 1x CPLD-/FPGA-Board, Vorgabe: NEXYS2-Board mit dem FPGA Spartan3E-1200
- RS232-Interface zur Kommunikation zwischen FPGA und PC
- Verbindungsmodul zwischen der ADC-Platine und dem FPGA

Da das Nexys2-Board als Vorgabe gilt, so wird noch ein neues Interface benötigt, das die beiden Module (ADC-Platine und Nexys2) miteinander verbindet. Im Hause der HAW gibt es eine weitere interne Entwicklung, die dafür das Verbindungsstück *NEXYS2 to ModSys Connectionboard* gebaut haben. Dieses kommt für diese Arbeit zum Einsatz. Es benötigt eine Versorgungsspannung von $5V$, um sowohl sich selbst als auch das Nexys2-Board zu versorgen. Somit besteht ein Steckverbinder für die Spannungsversorgung. Es wird ein weiterer Steckverbinder *Connector Modsys2Nexys* für das Nexys2-Board benötigt. Die Leitungen sind mit dem Steckverbinder des Typs *ModsysConnector* für die ADC-Platine verbunden. Das Nexys2-Board besitzt weitere Stecker, die mit der Bezeichnung *JA* bis *JD* versehen sind. Diese weisen pro Stecker insgesamt acht verwendbare Pins, die wiederum zusammen für

einen weiteren *ModsysConnector2* verwendet werden können. Die einzelnen Steckverbindungen können im obigen Bild 2.9 näher betrachtet werden. Der Steckverbinder *ModsysConnector* für die ADC-Platine besitzt insgesamt 80 Pins, von denen nur 37 frei zu verwenden sind. Einige von den übrigen Pins sind für die Masse und für die Spannungsversorgungen von $3,3V$ sowie $5V$ vorgesehen.

2.2 Umsetzung

In diesem Abschnitt wird darauf eingegangen, welche Bauelemente eingesetzt und wie diese auf der Hardware in Funktion gebracht werden.

2.2.1 Auswahl der Bauelemente

2.2.1.1 Analog-Digital-Converter

Aufgrund der erwünschten präzisen Auswertung der Sensordaten ist eine hohe Auflösung von $12bit$ unumgänglich (siehe 2.1.1). Wird auf die erste Auswertung der Analyse des Abschnitts 2.1.7 Bezug genommen, so werden mindestens drei Eingänge benötigt. Um viele Daten in kurzer Zeit digital zu erhalten, ist eine hohe Sample-Rate notwendig.

Sie sollte mindestens

$$minimale\ Sample\text{-}Rate = 2,5kHz \cdot 64\ Werte \cdot 3\ Kanäle = 0,48MSPS \tag{2.1}$$

betragen. Um die Anzahl der Werte für eine Periode variabel zu halten, sollte die Sample-Rate für die doppelte Anzahl von Werten gewählt werden, sodass der ADC mindestens eine Sample-Rate von $0,96MSPS$ erfüllen sollte.

In der Tabelle 2.2 sind drei verschiedene Modelle aufgelistet, zwei von der Firma Texas Instruments und eines von der Firma Linear Technology, die für diese Arbeit in Frage kommen.

- **ADC 1:** ADS7865
  Das besondere an diesem Bauelement der Firma Texas Instruments ist das parallele Daten-Interface. Es werden zwar insgesamt zwölf Leitungen für die Verbindung benötigt, aber sie werden alle zeitgleich an das FPGA und umgekehrt übertragen. Dadurch lässt sich viel Zeit nur bei der Datenübertragung einsparen. Weiterhin besitzt dieser ADC vier Eingangskanäle. Da nur drei benötigt werden, erfüllt dieser ADC die Bedingung. Die externe Taktung ist eine

Daten	ADS7865	LTC1851	ADS7863
Hersteller	Texas Instruments	Linear Technology	Texas Instruments
Auflösung	12 bit	12 bit	12 bit
Sample-Rate	2MS/s	1.25MS/s	2MS/s
Daten-Interface	parallel	parallel	Serial, SPI
Verbrauch	5.6mA	8mA	6.5mA
Taktung	extern	intern	extern
Eingangskanäle	4	8	4

Tabelle 2.2: *Auswahl des ADCs*

Besonderheit bei diesem Modell. Es kann zwischen $1MHz$ und $32MHz$ getaktet werden, sodass die Taktung des ADCS dem FPGA angepasst werden kann. Der Verbrauch von $5.6mA$ ist im Vergleich zu den anderen ADCs relativ gering.

- **ADC 2:** LTC1851
  Dieses Bauelement von der Firma Linear Technology besitzt ebenfalls das parallele Daten-Interface. Weiterhin weist dieses Bauelement insgesamt acht Eingangskanäle auf. Diese können frei miteinander kombiniert werden, sodass verschiedene Signale mit verschiedenen Referenzen verglichen werden können, ohne die Leitungen umplatzieren zu müssen. Dies ist in dieser Arbeit nicht erforderlich. Der Verbrauch ist $2,4mA$ höher als der vom ersten ADC. Die Taktung ist bei diesem Bauelement intern realisiert. Sie ist fest auf $1,25MHz$ eingestellt, sodass bei der Realisierung mit dem FPGA mehr Aufwand betrieben werden muss, um alle Daten zum richtigen Zeitpunkt auszulesen.

- **ADC 3:** ADS7863
  Dieses Bauelement ist ebenfalls von der Firma Texas Instruments. Der Verbrauch liegt mit $6,5mA$ zwischen den anderen beiden ADCs. Die Taktung ist ebenfalls extern und weist insgesamt vier Eingangskanäle wie der erste ADC auf. Der Unterschied liegt bei dem Daten-Interface. Es ist eine serielle Schnittstelle eingesetzt, die zwar an Leitungen und Anschlüssen spart, aber längere Übertragungszeiten benötigt.

Es wird für diese Arbeit der erste ADC *ADS7865* von der Firma Texas Instruments gewählt. Das besondere an diesem Bauelement ist der Besitz von zwei internen ADCs. Somit lassen sich zwei Kanäle gleichzeitig bearbeiten. Da hier vier Kanäle sind, können nach einer Umwandlungsphase zwei neue Werte abgeholt werden. Dadurch ist eine Bearbeitung von bis zu $2MSPS$ möglich. Dieser ADC lässt sich in einem Frequenzbereich von $1MHz$ bis $32MHz$ betreiben, sodass mit der Entscheidung der Taktfrequenz eine gewisse Variabilität besteht. Da in dieser Arbeit

mit einer Frequenz von $12,5MHz$ gearbeitet wird, ergibt sich eine Sample-Rate von $0,78125MSPS$. Damit ist die Sample-Rate von mindestens $0,48MSPS$ eingehalten.

Eine weitere Besonderheit des ADCs *AD7853* ist der interne DAC. Dieser wird für die Referenzspannung des ADCs eingesetzt. Der gewünschte Wert wird in das DAC-Register geschrieben und deckt einen Bereich von $0,5V$ bis $2,5V$ ab, wobei die $0,5V$ die Mindestspannung ist. Die $2,5V$-Spannung entsprechen $10bit$ und ergibt programmiert einen Wert von $0x3FF$. Standardmäßig sind die $2,5V$ bei der Initialisierung des ADCs eingestellt.

Der Aussteuerbereich liegt bei $-V_{REF}$ bis $+V_{REF}$. Bei einer Referenzspannung von $2,5V$ ergibt sich somit eine Spannungsdifferenz von $5V$. Es wird aber nur der positive Bereich verwendet, da die Signale von dem Vorverstärker-Modul nur positive Werte liefert.

Falls der ADC nicht mehr benutzt wird, kann dieser in verschiedene Arten von Ruhezuständen (*power down mode*) heruntergefahren werden, um Strom zu sparen. Es gibt vier Einstellungen dafür, die mit Hilfe von VHDL angesteuert werden können. Es handelt sich dabei um die Tiefe des Schlafes, wie sie erneut in den Betrieb zurückgerufen werden und wann sie betriebsbereit sind. Dies wird in dieser Arbeit nicht weiter vertieft, da der ADC ohne Unterbrechung arbeiten soll, was bei einem ABS-System erwartet wird.

Die verschiedenen Abläufe wie das Setzen der Einstellungen des ADCs und das Abholen der neuen konvertierten Daten werden komplett mittels VHDL im FPGA gesteuert. Diese Steuerung findet sich im FPGA in den Abschnitten 3.1.2 und 3.2.1 wieder.

DESCRIPTION	DIFFERENTIAL INPUT VOLTAGE (CHXX+) – (CHXX–)	INPUT VOLTAGE AT CHXX+ (CHXX– = V_{REF} = 2.5V)	BINARY CODE	HEXADECIMAL CODE
Positive full-scale	V_{REF}	5V	0111 1111 1111	7FF
Midscale	0V	2.5V	0000 0000 0000	000
Midscale – 1LSB	$-V_{REF}/4096$	2.49878V	1111 1111 1111	FFF
Negative full-scale	$-V_{REF}$	0V	1000 0000 0000	800

Bild 2.10: *ADS7865 Output Data Format; entnommen aus dem Datenblatt [9]*

Da die Signale vom Vorverstärker positiv sind und für die Referenzspannung $V_{REF} = 2,5V$ gilt (siehe Tabelle in Bild 2.10), bedeutet dies, dass nur der halbe Bereich von $0V$ bis $2,5V$ zur Verwendung kommt. Dies hat zur Folge, dass von den 12bit des ADCs nur 11bit gebraucht werden. Das MSB kann somit theoretisch ignoriert werden. Praktisch ist dies nicht möglich, aufgrund des auftretenden Rauschens auf das Signal bzw. der Abweichung bei der Konvertierung.

Eine erste Modifikation ist bei den analogen Eingangskanälen des ADCs bereits durchgeführt. Drei von den vier Eingängen werden genutzt und die vierte ist

offen geblieben. Um dies zu vermeiden wird dieser entweder auf Masse gezogen oder wie in dieser Arbeit genutzt, indem der Kanal ebenfalls mit dem ersten Signal *U_DIFF* versehen wird. Damit bestehen zwei Eingangskanäle mit dem Signal *U_DIFF* und jeweils ein Eingangskanal für *U_HB1* und *U_HB2*.

Da der ADC wie bereits erwähnt zwei interne ADCs besitzt, wird das Signal *U_DIFF* auf beide Kanäle des ersten internen ADCs gelegt. Da die internen ADCs die konvertierten Wert hintereinander ausgeben, wird das Signal *U_DIFF* nach jeder Konvertierung herausgegeben.

Bei der ersten Werteausgabe wird in dem ersten Zyklus, der erste Wert des ersten internen ADCs und der erste Wert des zweiten internen ADCs bereitgestellt. In dem zweiten Zyklus wird der zweite Wert des ersten internen ADCs und der zweite Wert des zweiten internen ADCs bereitgestellt. Dadurch gibt es nach jedem Zyklus immer einen neuen Wert vom Signal *U_DIFF* und jedes zweite Mal einen neuen Wert von den Halbbrückensignalen *U_HB1* und *U_HB2* (vgl. Datenblatt [9]).

2.2.1.2 Komparator

Bei der Auswahl des Komparators ist die Bedingung, dass das Modell über mindestens drei Komparatoren verfügt. Es sollen für jedes Signal des Vorverstärkers einer zum Einsatz kommen. Hier wird der *LM339* von der Firma ST eingesetzt, da sich dieser bewährt hat. Dieser verfügt über vier interne Komparatoren, sodass einer noch offen ist. Dieser offene wird zusätzlich für das Differenzsignal genutzt (siehe Abschnitt 2.1.5).

Zu der Spannungsversorgung ist festzuhalten, dass dieses Bauelement ebenfalls wie das Vorverstärker-Modul mit $5V$ betrieben wird.

Der Einsatz des Komparators *LM339* ist in Bild 2.11 dargestellt.

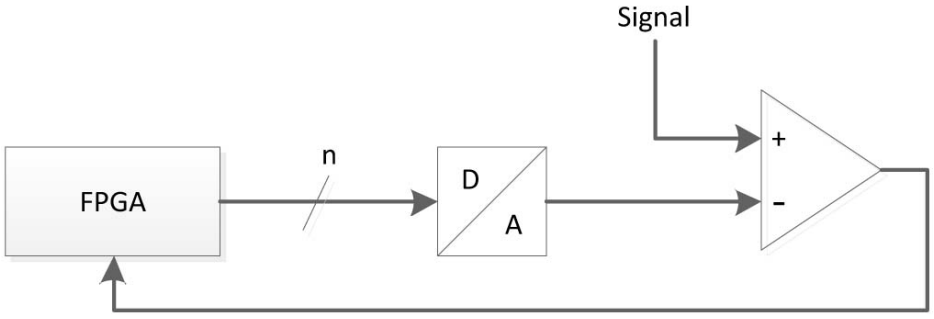


Bild 2.11: *Einsatz des Komparators in Kombination mit FPGA und DAC*

Hier ist zu erkennen, dass das Signal vom Vorverstärker als Eingangssignal des Komparators fungiert. Als Referenzspannung wird eine beliebig eingestellte Spannung vom DAC gegeben, die vom FPGA gesteuert wird. Das Ergebnis des Komparators wird dem FPGA zurückgegeben, sodass diese Daten weiter ausgewertet und bearbeitet werden können.

2.2.1.3 Digital-Analog-Converter

Aus den vier oben genannten Analogausgängen kommen noch vier weitere hinzu. Es werden insgesamt vier Ausgänge für die vier Komparatoren verwendet und einer für die Offset-Kompensation des Vorverstärker-Moduls. Die letzten drei werden benötigt, um auf dem Vorverstärker-Modul per Hand die Spannungen anzulegen. Hierzu gibt es keine Verbindungen oder Anschlussstellen. Dies sind sogenannten Brückenspannungen, die auf der ADC-Platine mit Hilfe zusätzlicher Pins erreicht werden.

Die Funktion des DACs ist in dem Bild 2.11 dargestellt. Der DAC wird vom FPGA gesteuert und gibt die analogen Werte an die Komparatoren weiter. Um bei der Präzision zu bleiben, wird hier ebenfalls ein DAC eingesetzt, der mit $12bit$ ausgestattet ist. Es bietet sich ebenfalls die parallele Ansteuerung der Datenbits an. Da sowohl für den ADC und DAC die gleiche Datenbitbreite gewählt wird, können diese zusammen über gleiche Datenleitungen zu dem FPGA genutzt werden. Dies setzt voraus, dass beide Bauelemente mit einem *ChipSelect*-Signal ausgestattet sind, damit einer zur Zeit gesteuert wird.

Das einzige passende Bauelement ist das Modell *AD5348* von der Firma Analog Devices.

Wichtige Details des DACs sind:

- 12 bit Eingang
- 8 analoge Ausgangskanäle
- paarweise eigene Referenzspannungen für die Ausgangskanäle
- synchrone bzw. asynchrone Aktualisierung der Ausgangsspannungen

Dieser DAC wird mit $3,3V$ versorgt, da keine größeren Ausgangsspannungen benötigt werden. Für die Ausgangskanäle sind Referenzspannungen notwendig, die von dem Bauelement *REF192* geliefert werden. Die Spannung beträgt hier $2,5V$. Der Verstärkungswert *GAIN* des DACs wird auf '1' gesetzt, um die doppelte Ausgangsspannung von *VREF* zu erhalten. Die maximale Ausgangsspannung ist auf die Betriebsspannung begrenzt, sodass der Spannungsbereich der Ausgangskanäle auf $0.001V$ bis $3,3V$ begrenzt ist.

Im folgenden wird gezeigt, welcher digitale Wert eingegeben werden muss, damit die gewünschte Ausgangsspannung erreicht wird (siehe Formel 2.2).

$$
\begin{aligned}
V_{REF} &= 2,5V \Rightarrow 1LSB = 0,6mV \\
&\quad für\ 0xFFF\ folgt\ 5V \\
V_{OUT} &= \frac{D}{2^n} * 2V_{REF} \qquad (2.2) \\
&\quad mit \\
D &= decimal \\
n &= 12bit \qquad (2.3) \\
V_{REF} &= 2,5V
\end{aligned}
$$

2.2.1.4 Weitere Bauelemente

Verbindungsboard

Das Verbindungsboard ist bereits in dem Abschnitt 2.1.6 beschrieben. Es besitzt zwei Steckverbinder für den Modsys-Connector. Diese sind in Bild A.1 im Anhang veranschaulicht.

In dieser Arbeit wird nur der eine Stecker *CONN3* verwendet. *CONN4* bleibt offen und ist nicht mit dem FPGA verbunden. Für Testzwecke werden vom FPGA einige Pins, die zu *CONN4* laufen (*JA* bis *JD* in Bild 2.8), genutzt.

In der Tabelle A.1 im Anhang ist die Pinbelegung der Pins *PIO* aufgezeigt und wie sie auf der ADC-Platine und auf dem FPGA bezeichnet werden. Auf der ADC-Platine gibt es zwei Versionen, da die eine sich auf den analogen Part, d.h. ADC/-DAC, und der andere auf den MSP bezieht.

Bei dem Steckverbinder sind 37 Pins frei belegbar. Die restlichen stehen entweder gar nicht zur Verfügung oder sie sind mit der Masse und den Spannungsversorgungen $3,3V$ und $5V$ vergeben.

Bei den Pins 1-12 handelt es sich um die Datenbits für ADC und DAC. Sie verwenden die gleichen Datenleitungen, da der Stecker über nicht genügend freie Pins verfügt. Es hat im Gegensatz dazu einen großen Vorteil, da weniger Leitungen verlegt werden müssen, da sie doppelt genutzt werden. Entweder wird nur der ADC angesprochen, über Pin 15 oder der DAC über Pin 24. Das Ansteuern läuft mit Hilfe der VHDL-Beschreibung (siehe Kapitel 3.1.2).

Drei Pins sind insgesamt offen und frei belegbar. Zwei weitere sind für die Komparator-Signale für die beiden Signale *HB1* und *HB2* vorgesehen. Diese werden im FPGA zur Zeit nicht ausgewertet.

Spannungsversorgung

Bei dem Modsys-Aufbau der eigenen Entwicklung des Hauses werden sowohl $5V$ als auch $3,3V$ von dem Board an die ADC-Platine gespeist. Das Verbindungsboard dagegen liefert nur die $3,3V$. Aus diesem Grund werden die $5V$ noch zusätzlich über einen externen Anschluss geliefert. Wenn es gewollt ist, können die $3,3V$ ebenfalls von den $5V$ bezogen werden. Es müssen lediglich die entsprechenden Bauelemente auf der ADC-Platine bestückt werden und der entsprechende Lötjumper überbrückt werden.

Der ADC benötigt die Spannungen von $5V$ für den analogen Bereich und $3,3V$ für den digitalen Bereich. Der DAC wird mit $3,3V$ versorgt. Die Referenzspannung erhält die externen $5V$ und liefert stabile $2,5V$ für die Referenzeingänge des DACs. Weiterhin wird das Komparator-Bauelement mit $5V$ versorgt.

Das Nexys2-Board benötigt $5V$ für den Betrieb. Diese werden von dem Verbindungsboard geliefert. Dieser bezieht seine Spannung von ebenfalls $5V$ von einer externen Versorgungsquelle und versorgt die ADC-Platine mit den $3,3V$, wenn dies gewünscht ist. Dies lässt sich durch einen Jumper ändern.

Das Vorverstärkungs-Modul wird mit $5V$ versorgt. Die digitalen Signale, die zu dem Vorverstärker geführt werden, laufen über einen Spannungsteiler, damit sie die $3V$ nicht überschreiten.

Alle Signale, die zum FPGA hin- bzw. vom FPGA wegführen, haben eine Spannung von $3,3V$ im 'High'-Zustand und $0V$ im 'Low'-Zustand.

2.2.2 Microcontroller

Bei der Auswahl des Microcontrollers ist bereits ein Modell vorgesehen. Es handelt sich dabei um das Modell *MSP430F1611*. Details sind dem Datenblatt [22] und der Diplomarbeit [11] zu entnehmen.

2.2.3 Bestückungsvarianten

Bei dieser Platine gibt es zwei Bestückungsvarianten. Die erste Variante besteht aus ADC und DAC, die zweite aus dem MSP-Microcontroller.

Zu der ersten Variante gehören die bereits beschriebenen Bauelemente. Diese werden kurz zusammengefasst.

Der ADC wird zur Konvertierung der ABS-Sensor-Signale verwendet. Die digitalen Ergebnisse werden dem FPGA übergeben, wo sie ausgewertet und weiter verarbeitet werden. Das FPGA übergibt digitale Signale an den DAC, der sie in analoge Werte konvertiert. Diese werden zum einen an das Vorverstärker-Modul übergeben und an die Komparatoren. Als Referenzspannung wird das externe Referenzbauelement hinzugezogen. Weitere digitale Signale vom FPGA werden zu dem Vorverstärker-Modul geführt, die die Verstärkung der Sensor-Signale steuert.

Zu der zweiten Variante gehört der *MSP430F169* bzw. *MSP430F1611*. Dieser muss mit einem Quarz betrieben werden. Weiterhin gibt es einen *Reset*-Eingang zum MSP. Dieses durch einen Taster ausgeführte *Reset*-Signal wird mit Hilfe der Hardware entprellt (Bauelement: *STM6717*). Der MSP hat ein eigenes *RS232*-Interface. Die Datenleitungen werden an einen Pin-Stecker weitergeführt. Die Sensorsignale des Vorverstärker-Moduls werden in dieser Variante direkt dem Microcontroller übergeben. Die Auswertung der Signale wird im MSP ausgeführt und die Verstärkung selber vom MSP mit Hilfe des Signalbusses *AMP_DB* und des Signals *AMP_CS* dem Vorverstärker-Modul mitgeteilt. Es gibt Verbindungen zwischen dem MSP und dem FPGA, sodass diese Daten austauschen und miteinander kommunizieren können. Die restlichen Kontakte des Microcontrollers werden mit einem Steckverbinder des Typs *CON3* (siehe Tabelle 2.3) verbunden.

Gemeinsames bestücken dieser beiden Varianten ist nicht vorgesehen. Da das Ziel dieses Projektes die Chipimplementierung ist, wird hier nicht weiter auf die zweite Variante mit dem Microcontroller eingegangen.

2.2.4 Platine

Zur vereinfachten Darstellung der ADC-Platine dient das Bild 2.12. Es werden die Anschlüsse der wichtigsten Bauelemente skizziert.

Ein wichtiger Punkt bei dem Platinenentwurf ist die Trennung der Masse zwischen dem analogen und dem digitalen Bereich des ADCs. Der *analoge* Bereich des ADCs umfasst die Eingangskanäle einschließlich die Verbindung zum dem Vorverstärker-Modul. Der digitale Bereich umfasst den Rest. Die beiden Massen sind unterhalb des ADCs mittels Durchkontaktierungen (engl. vias) miteinander verbunden. Der Grund für die Trennung ist die Pontentialverschiebung der Masse. Da die digitalen Signale schnelle und große Schwankungen hervorrufen, ändert sich das Massepotential sehr schnell. Diese Verschiebung darf sich nicht auf den analogen Bereich

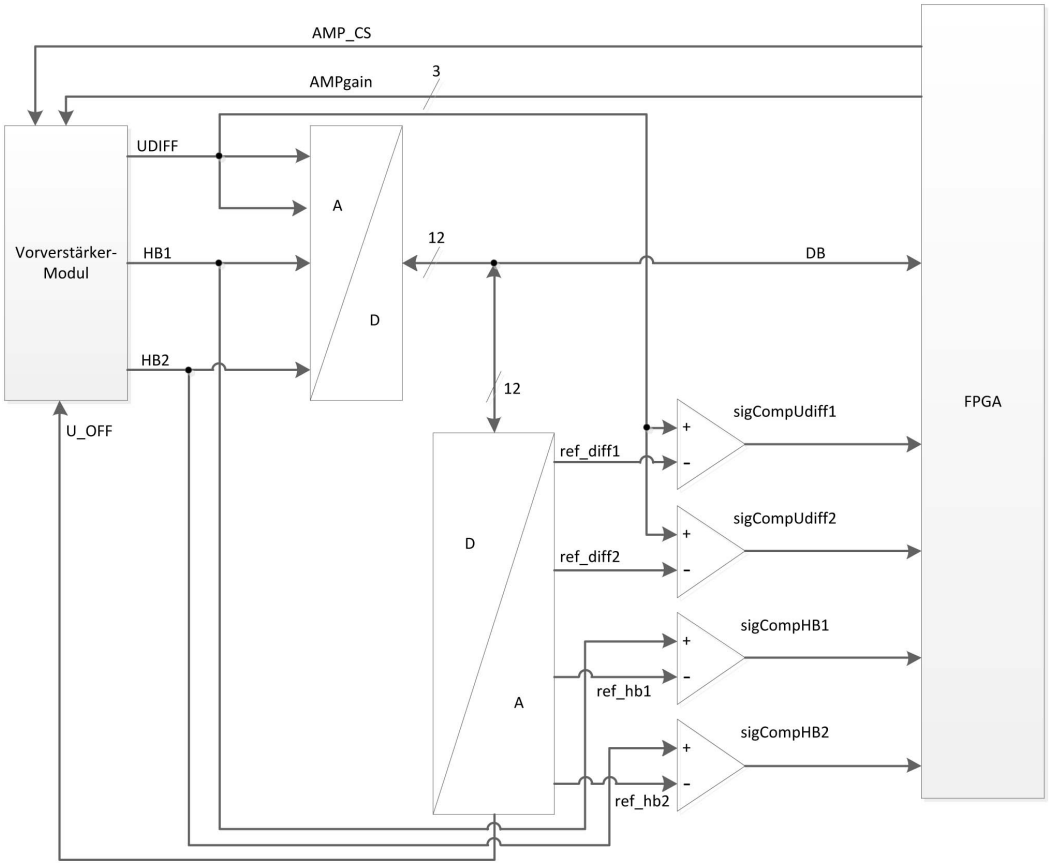


Bild 2.12: *Skizziertes Blockschaltbild der wichtigsten Bauelemente*

auswirken. Um dies zu verhindern, werden nur wenige Verbindungen der beiden Massen verwendet (siehe Datenblatt [9] Seite 26).

Die Schematics-Bilder befinden sich im Anhang A.2. Im folgenden sind die Ebenen nochmals in Bild 2.13 dargestellt sowie die entwickelte Platine 2.14. Weitere Bilder zu dem Entwicklungsdesign sind dem Anhang (siehe A.3) zu entnehmen (Toplayer siehe Bild A.2 und Bottomlayer siehe Bild A.3).

Das Vorverstärker-Modul wird, wie bereits erwähnt, auf zwei Steckverbinder gesetzt. Um für Stabilität zu sorgen, wurden noch weitere vier Löcher auf der ADC-Platine hinzugefügt, um das Vorverstärker-Modul mit Schrauben an die ADC-Platine zu befestigen.

In Tabelle 2.3 sind die Zahlen in den Bildern 2.13 und 2.14 mit den entsprechenden Bauelementen wiederzufinden.

Nr.	Bauelement
1	ADC
2	DAC
3	REF192
4	Komparator
5	Pins für Eingangssignale
6	CON3
7	MSP430F1611
8a-d	Löcher für das Festschrauben des Vorverstärker-Moduls
9	Modsys-Connector
10	externe Spannungsversorgung

Tabelle 2.3: *Zuordnung der Bauelemente zu den Nummern in den Bildern 2.13 und 2.14*

2.3 Erster Funktionstest

Bei diesem ersten Funktionstest ist es nur möglich zu kontrollieren, dass keine Kurzschlüsse, alle Lötstellen sicher und sauber sind sowie an gewissen Stellen die richtigen Spannungen anliegen. Dies wird durch ein Digitalvoltmeter durchgeführt.

Die zu messenden Spannungen sind der Tabelle 2.4 zu entnehmen und in den Bildern 2.13 und 2.14 dargestellt.

Bezeichnung	Bauelement
ma	$2,5V$
mb	$3,3V$
mc	$5V$

Tabelle 2.4: *Stellen der Spannungsmessungen zu den Buchstaben in den Bildern 2.13 und 2.14 dargestellt*

Weitere Tests sind erst mit der Implementierung des VHDL-Codes möglich.

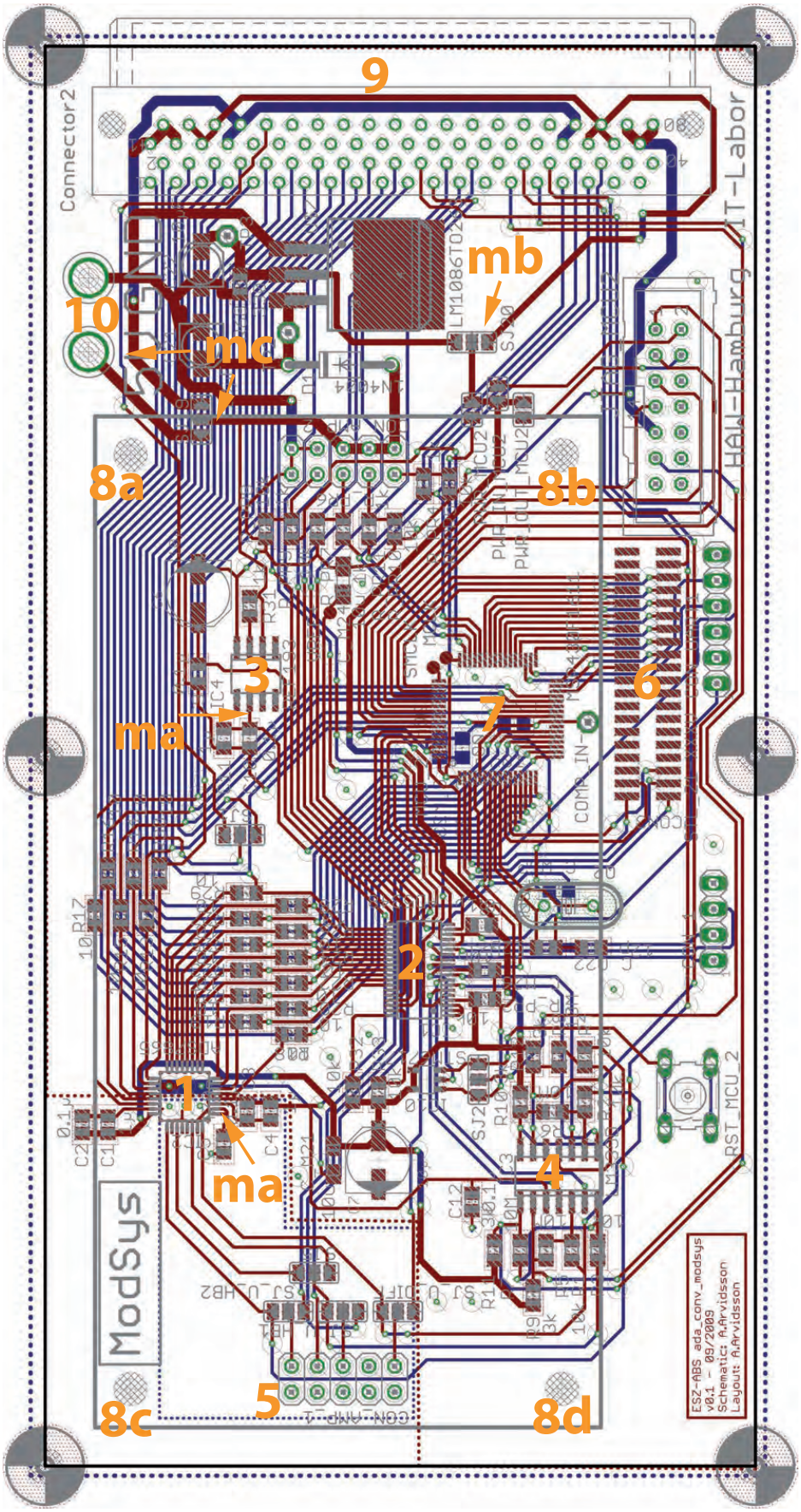


Bild 2.13: *Alle Layer des fertigen Designs*

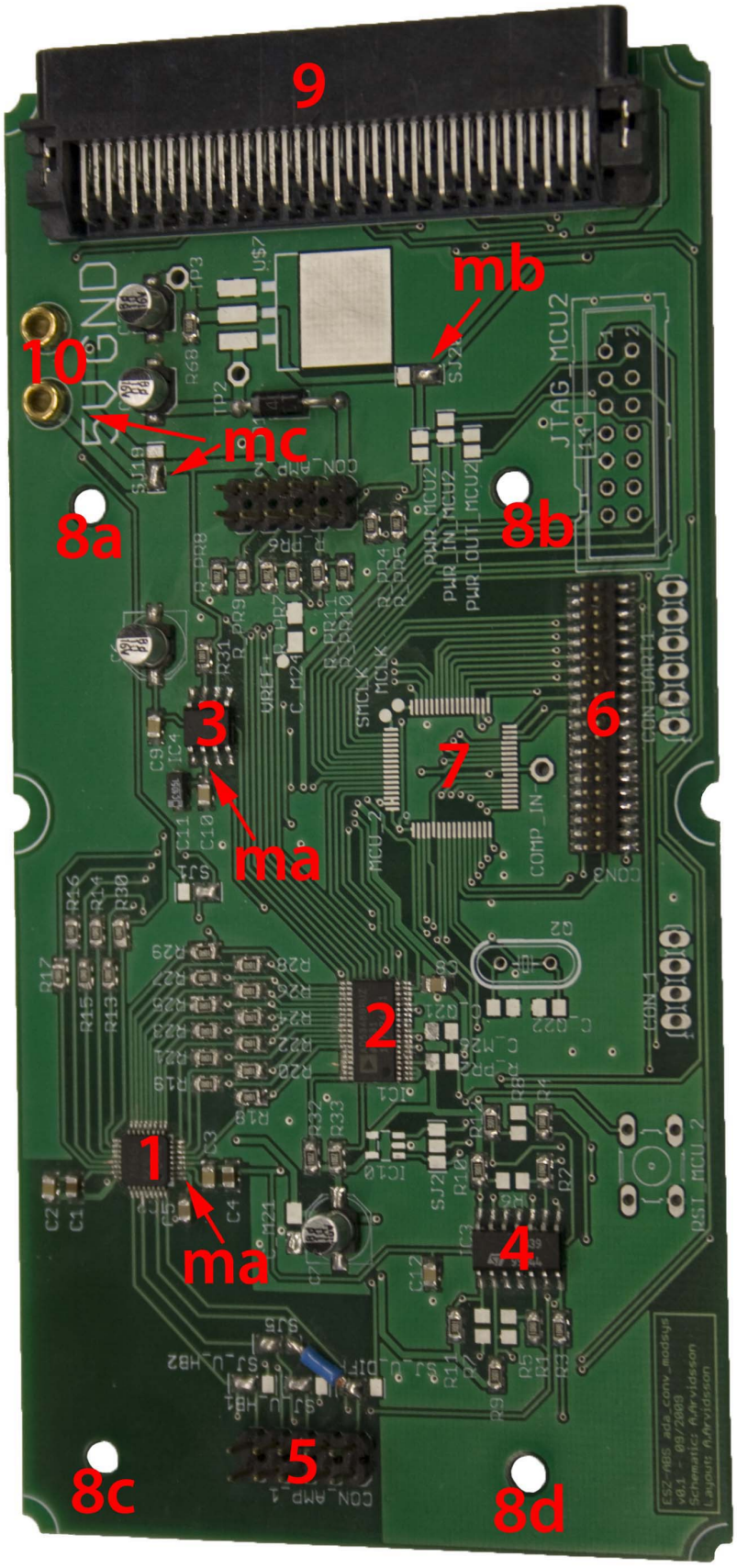


Bild 2.14: *Die entwickelte Platine*

3 Hardware - VHDL

In diesem Kapitel geht es um die Analyse und die Umsetzung der Aufgaben des FPGAs. Darunter fallen die Aufgaben Steuerung der ADC-Platine, die Kommunikation mit dem PC und weitere Funktionen, die im FPGA implementiert sind.

3.1 Analyse

In diesem Abschnitt der Analyse geht es darum, die einzelnen Aufgaben und Funktionen sowie die verschiedenen Zeitverhalten der Bauelemente zu analysieren.

3.1.1 Aufbau des VHDL-Codes

Der Aufbau des VHDL-Codes lässt sich in zwei Bereiche aufteilen (siehe Bild 3.1). Der erste Bereich *ada_conv_modsys_TL* ist für die Kommunikation mit der ADC-Platine zuständig, die die Steuerung und Datenkommunikation mit dem ADC und dem DAC sowie dem Vorverstärker-Modul mit dem FPGA-Board umfasst. Der zweite Bereich *MiniUART_TL* ist zuständig für die Kommunikation mit Hilfe des RS232-Interfaces zwischen dem FPGA und dem PC. Sie steuert das Senden, das Empfangen und die Auswertung der gesendeten und empfangenen Daten.

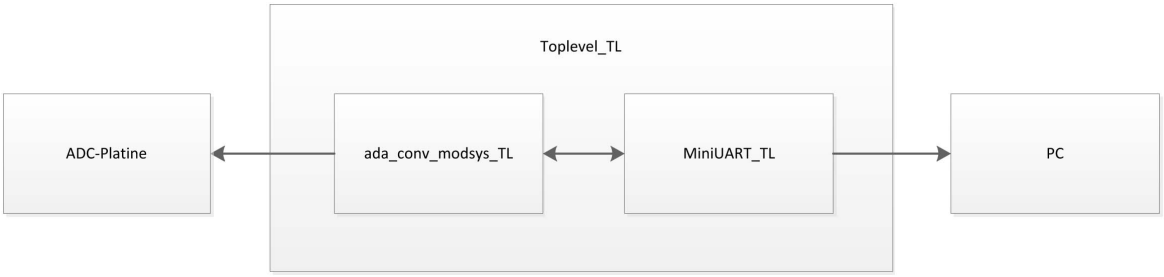


Bild 3.1: *Der Aufbau des VHDL-Codes in Toplevel_TL in Kommunikation mit ADC-Platine und PC*

3.1.2 Steuerung der ADC-Platine

Bei der Steuerung der ADC-Platine werden der ADC und der DAC getrennt betrachtet. Sie werden aber in dem gleichen Zustandsautomaten gesteuert und kontrolliert.

3.1.2.1 Analog-Digital-Converter

In der Software müssen die Signale *DB[11:0]*, $\overline{CS}$, *CLOCK*, $\overline{WR}$, $\overline{RD}$, *BUSY* und $\overline{CONVST}$ kontrolliert und gesteuert werden. Es soll hier genauer erklärt werden, wie der Zeitablauf aussieht und welche Signale wann gesetzt werden müssen. Dazu wird das Bild 3.2 zur Hilfe genommen.

Datenerfassung

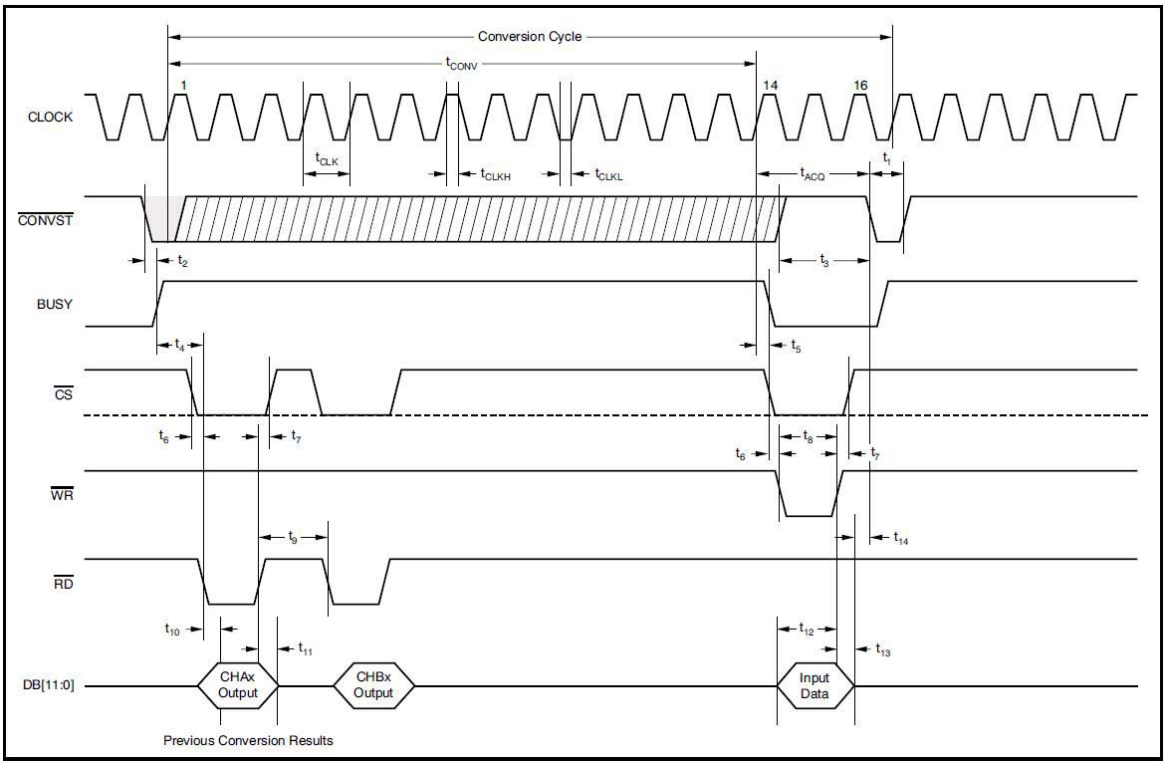


Bild 3.2: *Die allgemeine Zeitabhängigkeit der ADC-Steuerung und -Auswertung entnommen aus [9]*

Der Anfang des Bildes zeigt das Auslesen der konvertierten Daten aus dem ADC. Es müssen die Signale $\overline{CS}$ und $\overline{RD}$ gesetzt werden (sie sind 'low'-aktiv). Sind beide 'low', dann liegen die Daten auf dem Datenbus *DB* an. Dann liegt der erste

Datensatz bereit und kann abgespeichert werden. Beim nächsten 'low' der Signale liegt der zweite Datensatz an. Beim Auslesen der Daten ist das BUSY-Flag auf 'High' (das Signal ist 'high'-aktiv). Das BUSY-Signal wird vom ADC bestimmt. Es schlägt aus auf 'high', sobald das Signal $\overline{CONVST}$ ('low'-aktiv) gesetzt wird. Das BUSY-Signal ist solange auf 'high', bis die neuen anliegenden Analogsignale konvertiert wurden. Dies nimmt 13 Takte in Anspruch. Nach diesen 13 Takten ist es möglich, neue Daten in den ADC-Registern zu schreiben. Dafür müssen die Signale $\overline{CS}$ und $\overline{WR}$ ('low'-aktiv) gesetzt werden und die Daten, die vom FPGA bestimmt werden, müssen ebenfalls anliegen. Nach insgesamt 16 Takten ist ein kompletter Zyklus beendet und der nächste Zyklus beginnt. Damit alle vier Kanäle konvertiert sind, müssen zwei Zyklen abgewartet werden. Nach dem ersten Zyklus sind die ersten beiden Kanäle der jeweiligen internen ADCs und nach dem zweiten Zyklus sind die beiden anderen Kanäle fertig konvertiert und bereit für das Auslesen. Werden die Daten nicht ausgelesen, werden sie mit neuen Daten überschrieben, sobald neue vorliegen.

Konzept des Auslesens der Daten

Bei einem ABS-System werden immer neue und aktuelle Daten benötigt. Aufgrund dessen wird bei dem ADC eine bestimmte Funktion angewendet, die *Sequencer* genannt wird und dem *scan mode* eines Microcontrollers entspricht. Sobald eine Konvertierung fertig ist, wird gleich der nächste Wert konvertiert. Die konvertierten Daten müssen dann zum richtigen Zeitpunkt abgeholt werden. Dazu dient das BUSY-Signal.

Register programmieren und auslesen

Für das Programmieren eines Registers wird zuerst in das *Config*-Register geschrieben. Dort wird mitgeteilt, in welches Register als nächstes geschrieben werden soll. Soll beispielsweise in das DAC-Register des ADCs ein neuer Wert programmiert werden, dann wird in das *Config*-Register der Wert $0x101$ geschrieben. Bei dem nächsten Schreibzugriff wird dann in das DAC-Register geschrieben. Der neue Wert wird sofort an dem Analogausgang geändert. Soll das DAC-Register anschließend ausgelesen werden, wird zuerst wieder in das *Config*-Register geschrieben. Diesmal wird der Wert $0x103$ verwendet. Der nächste Zugriff ist ein Lese-Zugriff, sodass der Wert des DAC-Registers abgeholt werden kann.

Dieses Prinzip des Programmierens und Auslesens funktioniert ebenfalls mit den anderen Registern.

3.1.2.2 Digital-Analog-Converter

Register beschreiben und auslesen

Bei dem DAC gestaltet es sich einfacher als bei dem ADC. Hier werden lediglich die Daten vom FPGA auf den Datenbus *DB[11:0]* gelegt. Damit unterschieden wird, in welches Register die Daten geschrieben werden, wird ein weiterer Bus *A[2:0]* benötigt. Dadurch können alle acht Ausgangsregister separat und gezielt angesprochen werden.

Soll das erste Register beschrieben werden, wird der Bus A auf "000" eingestellt und der gewünschte zu schreibende Wert auf den Datenbus DB gelegt. Mit den Signalen $\overline{CS}$ und $\overline{WR}$ (hier ebenfalls 'low'-aktiv) wird der Wert in das erste Register, das hier die Bezeichnung $V_{OUT}A$ trägt, geschrieben.

Zum Auslesen werden die gleichen Einstellungen genutzt, wie beim Beschreiben. Der Unterschied dazu ist das zu setzende Signal $\overline{RD}$ und das Speichern des Datenbusses DB. D.h. damit keine Konflikte auf dem Datenbus entstehen, muss dieser in den hochohmigen Zustand versetzt werden, sobald dieser nicht benötigt wird. Dadurch ist es möglich, dass mehrere Komponenten auf den gleichen Bus zugreifen können.

Soll auf das vorletzte Register $V_{OUT}G$ zugegriffen werden, wird die Register-Codierung "110" für den Bus A verwendet.

Eine Besonderheit, die hier beachtet werden muss, ist dass die Daten erst geschrieben werden, sobald das Signal $\overline{WR}$ auf 'high' und beim Lesen, sobald das Signal $\overline{RD}$ auf 'high' zurückgesetzt wird. Einzelheiten dazu sind dem Datenblatt [4] auf den Seiten 5 und 18 und in Bild 3.3 zu entnehmen.

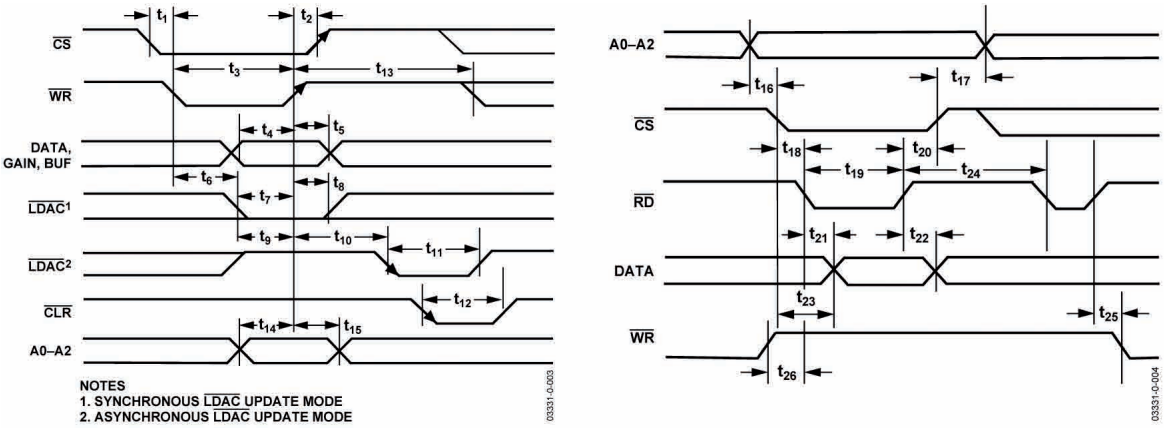


Bild 3.3: *Die allgemeine Zeitabhängigkeit der DAC-Steuerung und -Auswertung entnommen aus [4]: links wird in das Register geschrieben und rechts wird aus dem Register gelesen*

3.1.3 RS232-Interface

Für die Kommunikation zwischen dem FPGA und dem PC wird das RS232-Interface verwendet. Auf der PC-Seite gibt es in Matlab bereits fertige Funktionen für diese Kommunikation *binblockread()* und *binblockwrite()*. Für das FPGA gibt es entweder die Möglichkeit, das Protokoll selber zu schreiben, oder ein Fremdmodul in das FPGA einzubinden. Dazu wird das Fremdmodul *MiniUART* [3] benutzt.

Zur Ansteuerung des Moduls dient die Kommunikation mittels *Wishbone* (siehe Abschnitt 3.2.2.1).

Wie auch bei dem ADC und dem DAC gibt es hier bestimmte Zeiten, die bei der Ansteuerung des *Cores* einzuhalten sind (siehe Bild 3.4).

3.2 Umsetzung

In diesem Abschnitt wird darauf eingegangen wie die Funktionen umgesetzt sind.

In Bild 3.5 sind die Verbindungen der beiden Module *ada_conv_modsys_TL* und *MiniUART_TL* innerhalb des FPGAs gezeigt sowie die Verbindungen mit weiteren Modulen wie dem *Taktteiler* und dem Tastenentpreller wie auch den Verbindungen mit dem ADC und DAC der ADC-Platine und der RS232-Schnittstelle mit dem PC.

Die Module *ada_conv_modsys_TL* und *MiniUART_TL* sind im Anhang dargestellt (siehe Bilder B.1 und B.2).

Im folgenden wird auf die einzelnen Module näher eingegangen, wie sie funktionieren und wie sie voneinander abhängig sind.

3.2.1 Steuerung der ADC-Platine

In Bild 3.6 ist der Aufbau zur Ansteuerung von ADC und DAC schematisch dargestellt. Nach einem *RESET*-Befehl werden der ADC und DAC mit Standardwerten initialisiert. Diese werden anschließend wieder ausgelesen, welches in der Initialisierungsphase eingestellt ist. Damit befinden sich alle Einstellungen und Werte im Auslieferungszustand dieses Designs.

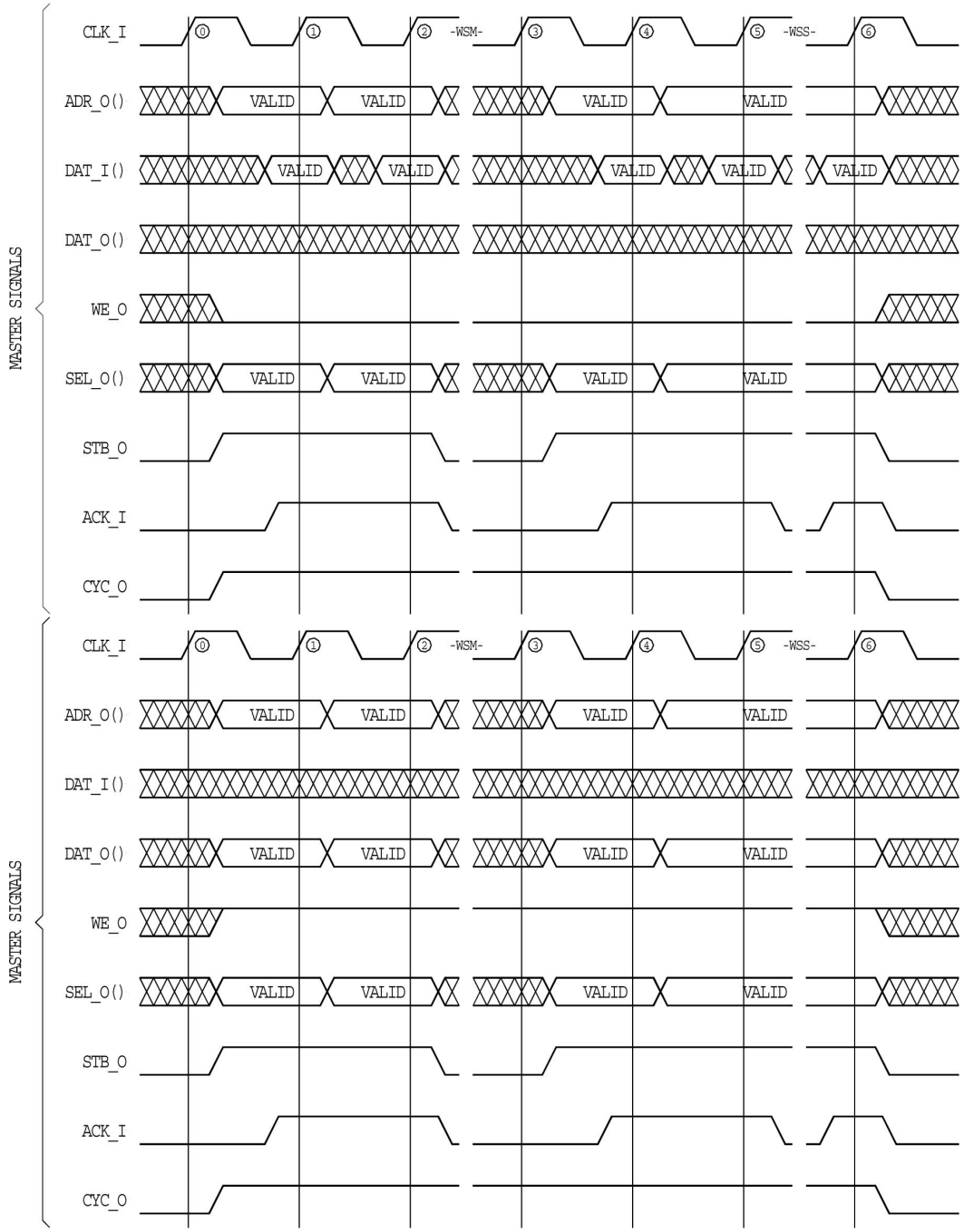


Bild 3.4: *Die Zeitabhängigkeit der Steuerung fürs Lesen (oben) und Schreiben (unten) für Wishbone [8] Seite 54 und 57*

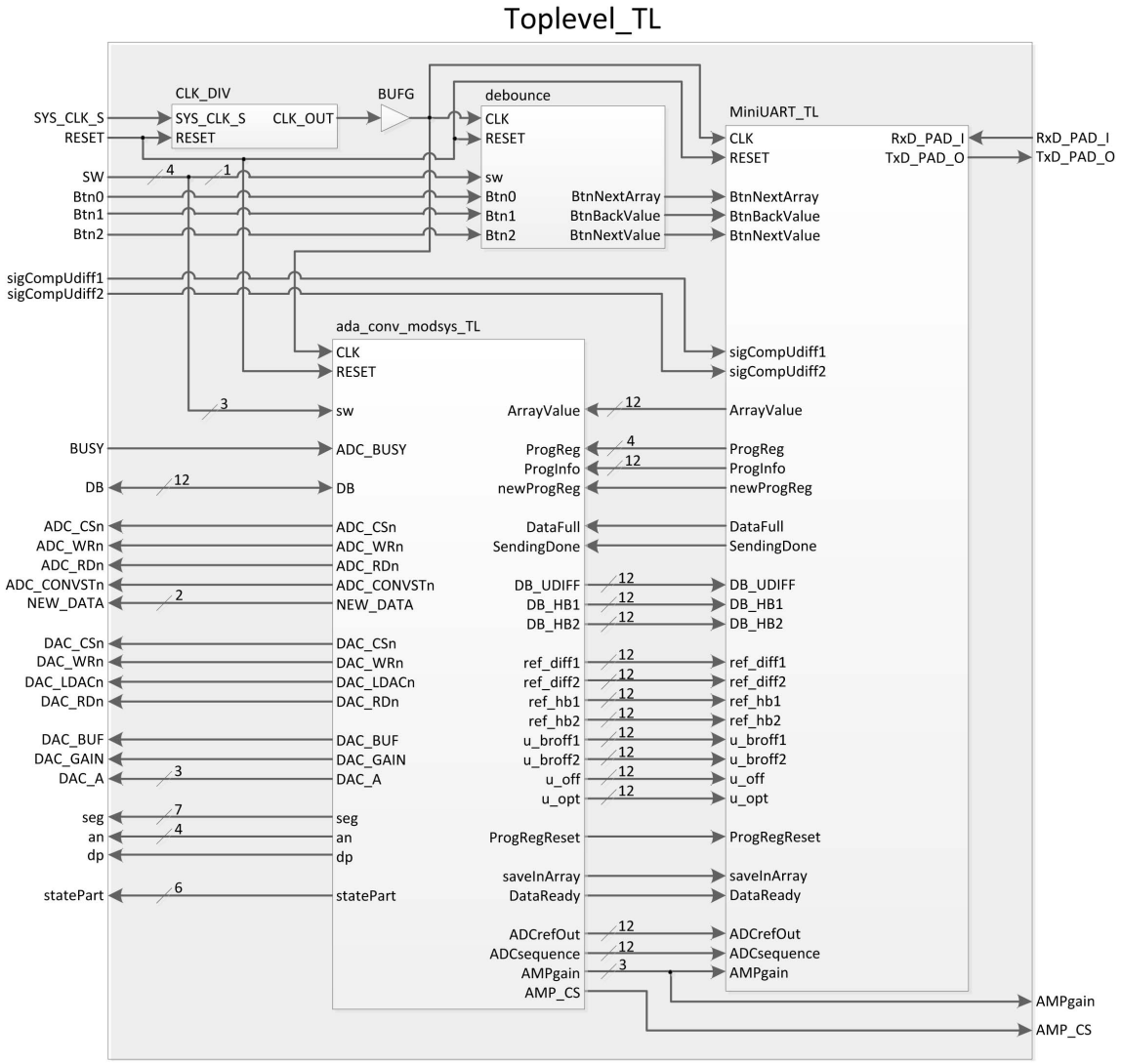


Bild 3.5: *Toplevel mit der Verbindung zwischen den Modulen ada_conv_modsys_TL und MiniUART_TL und den FPGA-Ein- und -Ausgängen*

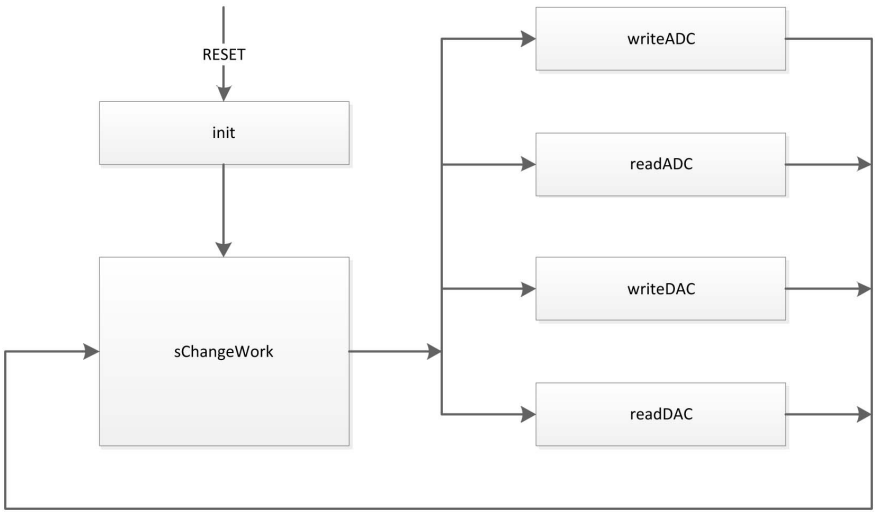


Bild 3.6: *Aufbau zur Ansteuerung von ADC und DAC*

Wenn von Matlab ein Befehl kommt, wird dieser ausgewertet und vom Steuerpfad (siehe Abschnitt 3.2.1.1) bearbeitet. Parallel dazu läuft der Datenpfad (siehe Abschnitt 3.2.1.2), der dafür sorgt, dass die richtigen Daten auf dem Datenbus anliegen.

3.2.1.1 Steuerpfad

Der ADC wird mittels des Signals ChipSelect (*ADC_CSn*) angesprochen. Zum Beschreiben wird das Signal WR (*ADC_WRn*) gesetzt und zum Lesen das Signal RD (*ADC_RDn*). Alle Steuersignale außer das Signal *ADC_BUSY* sind low-aktiv, wie bereits oben erwähnt.

Bei einem Neustart des ADCs werden die Standardeinstellungen des ADCs verwendet. Da dies nicht immer erwünscht oder sinnvoll ist, werden die einzelnen Register für die verschiedenen Einstellungen umprogrammiert bzw. neu beschrieben. Hierzu wird ein Zustandsautomat verwendet, über den die Steuerung der Einstellungen des ADCs laufen (*CONTR_FSM*, siehe Bild 3.7). Sie wird auch Steuerpfad genannt. Dieser Automat setzt die verschiedenen Signale des ADCs und setzt ebenfalls weitere Steuersignale, die an den Datenpfad (*Datapath*) weitergegeben werden, der wiederum für die Daten zuständig ist. Durch die Trennung von Steuer- und Datenpfad bleibt die Übersicht der Funktionen und der Codes erhalten.

Für den DAC gilt die gleiche Funktionsweise. Mit dem Signal ChipSelect *DAC_CSn* wird der DAC angesprochen. Mit dem Signal *DAC_WRn* in die Register geschrieben und mit *DAC_RDn* die Register ausgelesen. Mit dem drei Bit breiten Bus *DAC_A* wird das entsprechende Register ausgewählt.

Der Steuerpfad (siehe Bild 3.7) übernimmt folgende Aufgaben (die in eckigen Klammern genannten Farben entsprechen den Farben in Bild 3.7, die durchgezogenen Linien bedeuten Schreiben und die unterbrochenen Linien Lesen):

- ADC:
  - PowerOnReset bzw. Initialisierung durchführen [schwarz]
  - DAC-Register beschreiben [rot]
  - DAC-Register auslesen [rot gestrichelt]
  - Sequence-Register beschreiben [hellgrün]
  - Sequence-Register auslesen [hellgrün gestrichelt]
  - Kontrolle, welche Kanäle als nächstes konvertiert werden [hellblau gestrichelt]
  - BUSY-Flag abfragen [lila]
  - Werte aus den Registern holen [hellblau]
- DAC:
  - DAC-Register beschreiben [dunkelgrün]
  - DAC-Register auslesen [dunkelgrün gestrichelt]

Der Steuerpfad besteht aus dem Kopf (sChangeWork), der die Steuerung der Aufgaben übernimmt, und aus mehreren unterteilten Aufgaben. Diese einzelnen Aufgaben umfassen die oben genannten Punkte.

PowerOnReset bzw. Initialisierung [schwarz]:

Im PowerOnReset-Modus werden alle Register des ADCs komplett gelöscht und in den Ursprungszustand zurückgesetzt. Um dies durchzuführen, muss der Wert $0x005$ (siehe Tabelle 3.1) in das *Config*-Register geschrieben werden. Nach etwa $20ns$ (siehe Datenblatt [9]) ist der Reset-Vorgang abgeschlossen und es kann wieder auf den ADC zugegriffen werden. Wird dieser Zustand ausgeführt, so wird die komplette Initialisierungsphase erneut durchlaufen. Nach einem ADC-Reset ist es notwendig, die entsprechenden Einstellungen dieses Projektes für die Ausgänge erneut zu setzen.

Für das Durchlaufen des PowerOnResets (POR) werden vier Zustände benötigt. Dabei wird die Initialisierungsphase gestartet und der Reset durchgeführt. Das Signal *init* wird gesetzt, damit alle Aufgaben des Zustandsautomaten durchlaufen werden. Alle Register von ADC und DAC werden mit den Standardeinstellungen

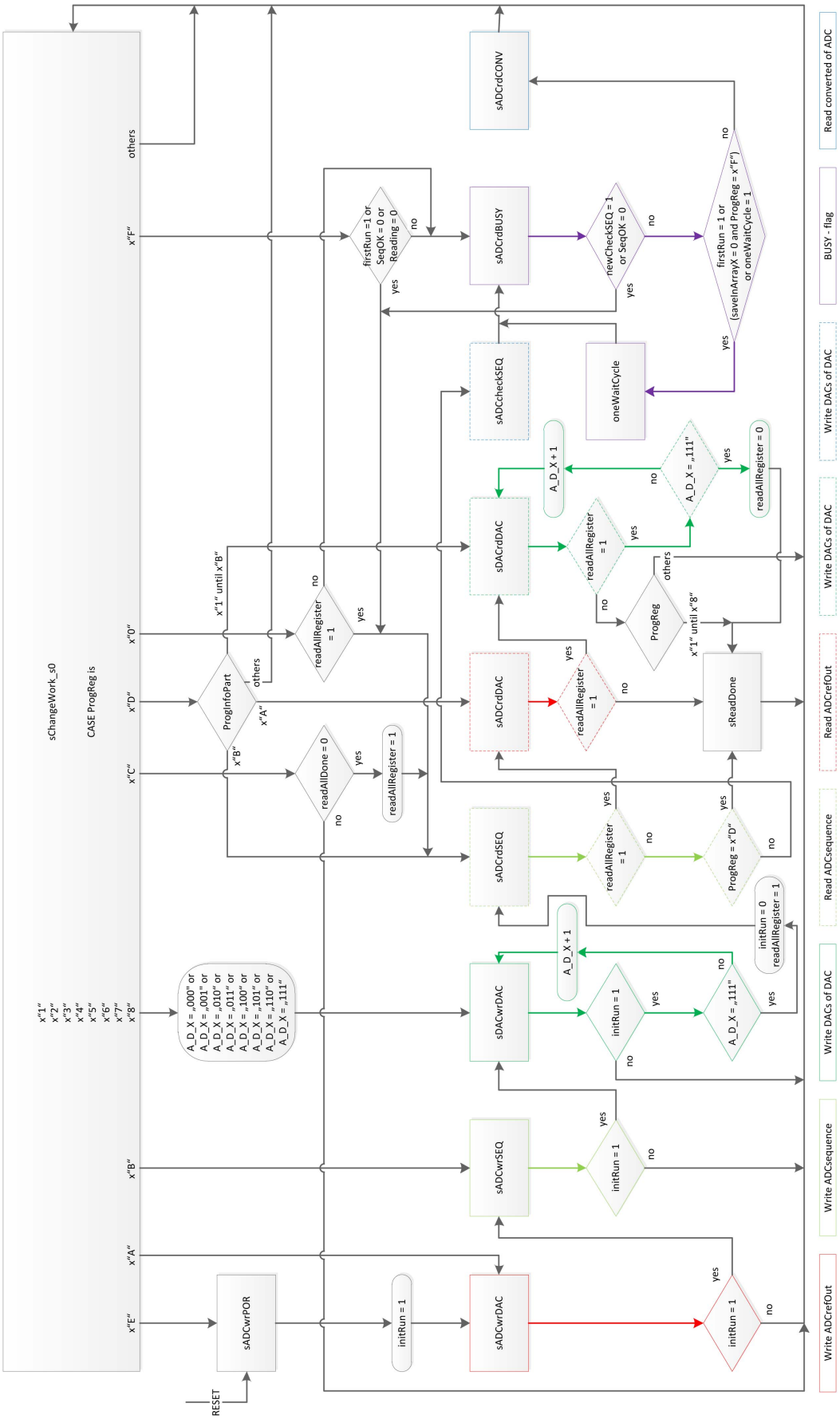


Bild 3.7: *Ablaufdiagramm des Zustandsautomaten Steuerpfad*

neu beschrieben und am Ende erneut ausgelesen. Am Ende der Initialisierungsphase wird das Signal *init* wieder zurückgesetzt. Dies ermöglicht den Wechsel in den Zustand *sChangeWork*.

DAC-Register des ADCs [rot, rot gestrichelt]:

Für das Beschreiben des DAC-Registers werden neun Zustände verwendet. Ist das Signal *ADC_BUSY* auf 'low', wird in das Config-Register der Wert für den DAC geschrieben. Im nächsten Schreibzugriff wird der gewünschte Wert für den DAC geschrieben. Damit die Daten sicher anliegen, werden die Daten ein Takt vor dem Schreiben angelegt.

Um dieses Register wieder auszulesen, werden sieben Zustände benötigt. Es wird in das Config-Register geschrieben, dass das DAC-Register ausgelesen werden soll. Beim nächsten Lesezugriff liegt der Wert des DAC-Registers zum Lesen bereit.

Sequence-Register [hellgrün, hellgrün gestrichelt]:

Der Ablauf ist wie beim DAC-Register des ADCs.

Beim Auslesen dieses Registers ist zu beachten, dass die letzten beiden Bits Auskunft darüber geben, welche beiden Kanäle als nächstes konvertiert werden. Diese Information wird für die Kanalzuordnung benötigt, damit die Daten in die richtigen Arrays (Speicherfelder) gespeichert werden. Durch dieses Auslesen werden Kanalwechsel vermieden.

Kontrolle [hellblau gestrichelt]:

Bei der Kontrolle wird das Sequence-Register kontrolliert. Die beiden letzten Bits werden hier ausgewertet. Dadurch ist es möglich zu erkennen, welche Kanäle als nächstes konvertiert werden. Dies ist nötig, um bei der Speicherung der Werte Kanalwechsel zu vermeiden.

Diese Funktion wird nur einmal pro Abspeicherungsfolge ausgeführt. Sollen neue Werte in die Arrays abgelegt werden, wird diese Funktion ausgeführt. Dann wird entschieden, ob ein zusätzlicher Zyklus des Signals *ADC_BUSY* gewartet werden muss. Dies wird solange durchgeführt, bis der erste Kanal dran ist.

Abfrage des Signals ADC_BUSY [lila]:

Bei der Abfrage des Signals *ADC_BUSY* geht es darum zu erfahren, in welchem Zustand sich der ADC befindet, d.h. ob dieser neue Werte konvertiert oder nicht. Sobald das Signal auf 'low' zurückgesetzt wird, sind neue Werte abholbereit.

Ist das Signal *ADC_BUSY* gesetzt, werden die neuen Daten, die an den ADC-Eingängen liegen, konvertiert.

ADC-Werte holen [hellblau]:

In diesem Bereich werden die konvertierten Daten aus dem ADC gelesen und den Variablen UDIFF, HB1 und HB2 zugewiesen. Damit die Zuweisung einwandfrei läuft, ist die Signal-Abfrage von *ADC_BUSY* in Kombination mit dem Sequence-Register notwendig.

Im Sequence-Modus wird darauf gewartet, dass das Signal *ADC_BUSY* von 'high' auf 'low' geht und wieder auf 'high' gesetzt wird. Sobald *ADC_BUSY* auf 'high' ist, werden die Daten aus den Registern gelesen. Beim nächsten 'high' des Signals werden die nächsten Daten ausgelesen.

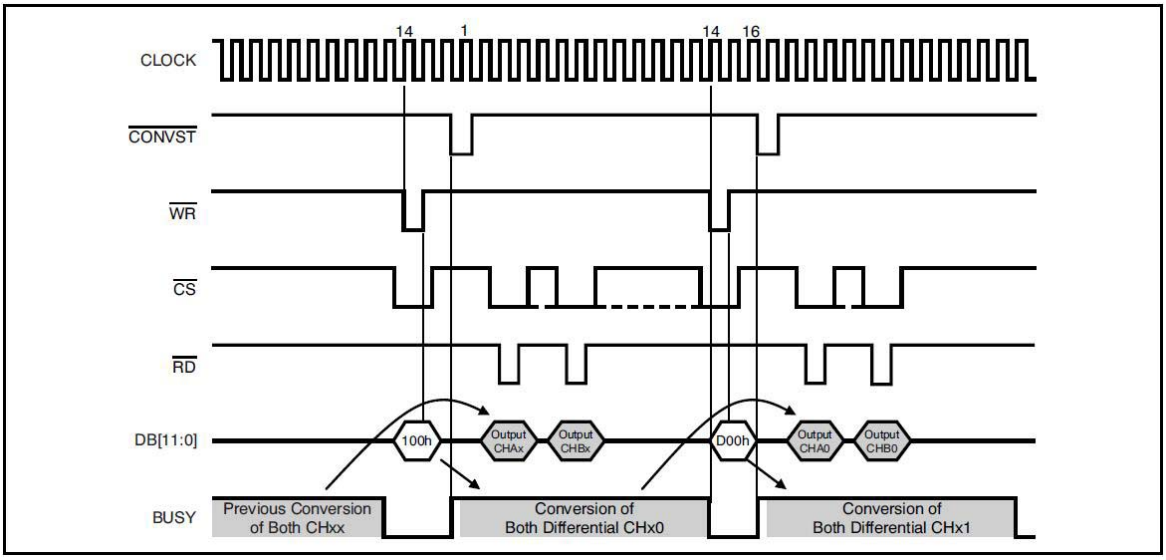


Bild 3.8: *Einstellung der als nächstes zu konvertierenden Kanäle; entnommen aus [9]*

In dem Bild 3.2 ist zu erkennen, wann welche Signale gesetzt und welche Daten angelegt werden müssen. Eine Konvertierung benötigt insgesamt 13 Takte. Mit Programmierung der Register dauert ein Zyklus somit insgesamt 16 Takte. Das Signal *ChipSelect* ist wichtig, um die beiden analogen Devices ADC und DAC voneinander zu trennen.

Das Besondere ist, dass das Sequence-Register mit verschiedenen Kanälen und Zyklen programmiert werden kann. Es muss anschließend dafür gesorgt werden, dass die Daten zum richtigen Zeitpunkt abgerufen und den entsprechenden Variablen zugeordnet werden. Dadurch wird das Umprogrammieren der Kanäle dem ADC überlassen. Das *Sequence*-Register muss anfangs mit den entsprechenden Kanälen eingestellt werden. Es werden immer zwei Kanäle gleichzeitig konvertiert, da dieser ADC zwei interne ADCs besitzt. Dies hat den Vorteil, dass genau nach zwei

Zyklen, d.h. nach 32 Takten, alle vier Kanäle konvertiert und an das FPGA übertragen worden sind. Vier Kanäle werden konvertiert, nur drei davon werden weiterverarbeitet. So wie es zur Zeit mit den Kanälen angeschlossen ist, ist es möglich, nach jedem Zyklus einen neuen Wert von Udiff zu bekommen, während die beiden anderen Kanäle erst nach zwei Zyklen. Dies ist aber nicht notwendig, da bei der maximalen Frequenz des ABS-Sensors von $2.5kHz$ nach 78 Takten (siehe Formel 3.4) ein neuer Wert eines Signals entgegengenommen wird.

Anzahl der Werte für eine komplette Schwingung:

$$\frac{1}{2.5kHz \cdot 32\ Takte} = 156.25\ . \tag{3.1}$$

Dies entspricht

$$\frac{156.25}{64\ Werte} = 2.44141 \tag{3.2}$$

Zyklen. Dies entspricht wiederum

$$2.44141 \cdot 32\ Takte \cdot 80ns = 6\mu s. \tag{3.3}$$

Ein Zyklus besteht aus 32 Takten, sodass für 2.44141 Zyklen

$$2.44141 \cdot 32 = 78.125 \tag{3.4}$$

Takte für einen neuen Wert ergeben. Nach 78 Takten wird ein neuer Wert in die Arrays gespeichert, wenn 64 Werte für eine Schwingung mit der Frequenz $2.5kHz$ genommen werden soll.

DAC beschreiben und auslesen [dunkelgrün, dunkelgrün gestrichelt]:

Bei der Ansteuerung des DACs sieht es ein wenig anders aus. Wenn das DAC-Register des Kanals A eingestellt werden soll, wird zum einen der gewünschte Wert auf den Datenbus *DB* gelegt. Da es nun mehrere Kanäle (A-H = ”000” bis ”111”) gibt, muss noch eine weitere Einstellung getätigt werden. Der 3Bit-breite Bus *DAC_A* muss entsprechend dem gewünschten Kanal angepasst werden, damit das richtige Register beschrieben wird. Da beispielsweise das Register des Kanals C beschrieben werden soll, wird der Bus *DAC_A* auf ”010” eingestellt. Um den Wert in das Register zu schreiben, muss noch das Signal *DAC_WRn* gesetzt werden.

Das Auslesen funktioniert genauso, nur wird das Signal *DAC_RDn* gesetzt anstatt *DAC_WRn*.

3.2.1.2 Datenpfad

Der Datenpfad dagegen legt die richtigen Daten an den Datenbus (*DB*). Die zusätzlichen Steuersignale des Steuerpfades geben dem Datenpfad den Zeitpunkt an, wann welche Daten angelegt werden sollen. Beispielsweise muss zum Auslesen des DAC-Registers des ADCs zuerst der Wert $0x103$ angelegt werden. Dieser Wert wird in das *Config*-Register geschrieben. Beim nächsten Lesezugriff liegt der Wert aus dem DAC-Register an.

Aus der Tabelle 3.1 lässt sich erkennen, welche Daten angelegt werden müssen, um in die verschiedenen Register zu gelangen:

Daten	Funktion
$0x000$	nur einmal neue Werte
$0x005$	PowerOnReset
$0x101$	um ins DAC-Register des ADCs zu gelangen
$0x103$	DAC-Register des ADCs auslesen
$0x104$	um ins Sequence-Register des ADCs zu gelangen
$0x106$	Sequence-Register des ADCs auslesen

Tabelle 3.1: *Steuerungswerte für den ADC*

Für den DAC wird die Registerwahl im Steuerpfad über den Bus *DAC_A* eingestellt. Dieser Bus wird dem Datenpfad übermittelt, damit dieser die entsprechenden Daten korrekt auf den Datenbus *DB* legen bzw. den Datenbus korrekt auslesen kann.

3.2.1.3 Vorverstärker-Modul

Das Vorverstärker-Modul erhält insgesamt vier Signale, die vom FPGA über die ADC-Platine gesteuert werden. Hierbei geht es um die Verstärkung *AMP_DB*, das 3Bit breit ist, und um das Speichern der neuen Verstärkung in den Microcontroller, der sich auf dem Vorverstärker-Modul befindet.

Zur Bestimmung der Verstärkung wird das Bild 3.9 zu Hilfe gezogen. Das Eingangssignal *UDIFF* wird vom Vorverstärker-Modul geliefert. Die beiden grünen Linien *DIFF1* und *DIFF2* sind die analogen Schwellenspannungen der beiden Komparatoren. Die roten Linien *barrier1* bis *barrier4* sind die Schwellen, die maßgebend für die Bestimmung der Verstärkung sind. Diese Verstärkungsbestimmung wird im FPGA realisiert. Zur weiteren Veranschaulichung des Ablaufs dient der Programmausdruck 3.1.

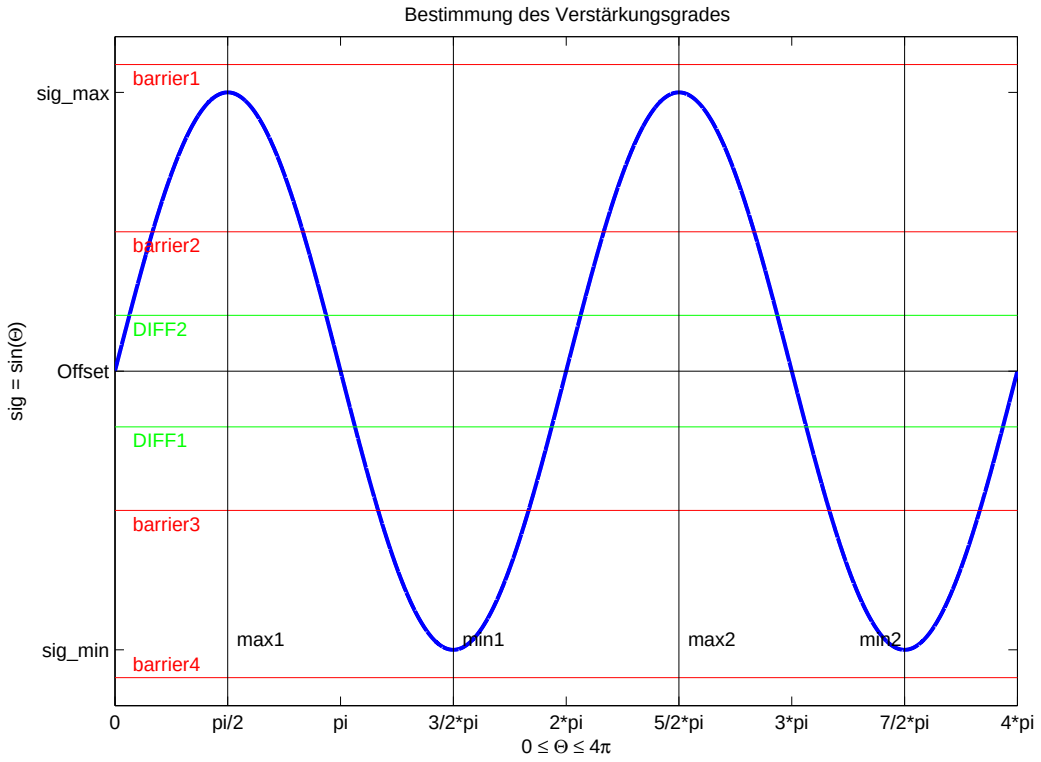


Bild 3.9: *Bestimmung der Verstärkung für das Vorverstärker-Modul*

```
1  if (UdiffMin < barrier4) then
2    if GainInt > "000" then
3      AMP_CS <= '1' after 9 ns;
4      GainInt <= GainInt - 1 after 9 ns;
5    end if;
6  end if;
7  if (UdiffMin > barrier3) then
8    if GainInt < "111" then
9      AMP_CS <= '1' after 9 ns;
10     GainInt <= GainInt + 1 after 9 ns;
11   end if;
12 end if;
```

Programmausdruck 3.1: *Bestimmung der Verstärkung*

Damit die Funktion funktioniert, wird nur etwas ausgeführt, wenn sich das Signal *Udiff* im oberen Bereich befindet, d.h. die beiden Komparatoren geben beide eine '1' heraus, oder wenn sich das gleiche Signal im unteren Bereich befindet und zwar

wenn beide Komparatoren eine '0' ausgeben. Aus diesem Grund wird dieser Code in den Zustandsautomaten zur Bestimmung der Periodendauer eingesetzt.

Im ersten Fall wird das Signal der Variablen UdiffMax solange zugewiesen, bis das Maximum erreicht wurde. Weiterhin wird das Minimum aus der vorherigen Halbwelle genommen und mit den Schwellen (rot dargestellt) verglichen. Befindet sich das Minimum unterhalb von *barrier4*, d.h. $UdiffMin < barrier4$, dann wird die Verstärkungsstufe *Gain* um '1' verringert werden. Dies ist aber nur möglich, wenn die Verstärkungsstufe $Gain > 0$ ist. Die Signalamplitude ist somit zu groß. Ist die Signalamplitude zu klein, d.h. UdiffMin befindet sich oberhalb der Schwelle *barrier3* ($UdiffMin > barrier3$), dann wird die Verstärkungsstufe *Gain* um '1' erhöht, sofern der sie nicht bereits sein Maximum mit der Stufe 7 erreicht hat.

Befindet sich das *Udiff*-Signal im zweiten Fall, d.h. beide Komparatoren geben den Wert '0' an, dann wird das Maximum des Signals mit den Schwellen *barrier1* und *barrier2* verglichen. Ist $UdiffMax > barrier1$, dann wird die Verstärkungsstufe *Gain* um '1' verringert, sofern sie nicht bereits auf '0' ist. Ist $UdiffMax < barrier2$, dann ist die Signalamplitude zu klein und die Verstärkungsstufe *Gain* wird um '1' erhöht, wenn die Stufe < 7 ist.

Die Änderung der Verstärkungsstufe *Gain* wird nur einmal pro Halbwelle geändert. Die Bestimmung des Maximums bzw. Minimums wird ununterbrochen bestimmt. Ist die neue Verstärkungsstufe bestimmt, wird das entsprechend gespeicherte und gerade verwendete Maximum bzw. Minimum gelöscht. Das Maximum wird dann auf den minimalen Wert ($(others =>' 0')$) gesetzt und das Minimum auf den maximalen Wert seines 12bit breiten Speichers ($(others =>' 1')$).

Diese Funktion befindet sich in dem gleichen Modul wie die Bestimmung der Periodendauer (siehe Quellcode C.1.19 im Anhang), da beide Module mit Hilfe eines Zustandsautomaten realisiert sind und mit den gleichen Komparatorensignalen arbeiten. Somit kann ein kompletter Zustandsautomat und weitere Leitungen eingespart werden.

3.2.2 Kommunikation mit PC

Dieser Abschnitt umfasst die Kommunikation zwischen dem FPGA und dem PC, auf dem Matlab läuft.

Für die Kommunikation zwischen FPGA und PC wird die serielle Schnittstelle (RS232) des Nexys2-Boards verwendet. Um nicht die komplette Vorgehensweise

selber zu programmieren, konnte bei der Firma OpenCores ein Fremdmodul heruntergeladen werden, das die Funktion des MiniUARTs übernimmt. Nötige Einstellungen sind Baudrate, die mittels eines Baudraten-Teilers *BRDIVISOR* eingerichtet wird, Anzahl der StopBits, Timeout usw.

Für die Einstellung der Baudrate muss diese erstmal festgelegt werden. Anschließend wird der Wert BRDIVISOR bestimmt und in der Datei *miniuart.vhdl* angepasst. Es wird die Baudrate 19200 verwendet. Somit ergibt sich für die Variable BRDIVISOR der Wert:

$$BRDIVISOR = \frac{f}{Baudrate \cdot 4} \tag{3.5}$$

$$= \frac{12.5MHz}{19200 \cdot 4} \tag{3.6}$$

$$= \frac{12.5MHz}{76800} = 162.76\ . \tag{3.7}$$

Dieser Wert wird auf 163 aufgerundet, da es im Format *Integer* vorliegen muss.

Der als Fremdmodul eingesetzte MiniUART-Core muss in das vorhandene Projekt eingebunden werden, welches mittels der Wishbone-Komponente gelöst wird. Dazu wird eine zusätzliche Komponente benötigt, die die Steuerung über Wishbone übernimmt (siehe [2]). Die folgende Darstellung veranschaulicht dies.

3.2.2.1 Wishbone

Wishbone ist ein Steuerungsverfahren, mit dem andere Fremdmodule (Cores) in ein bereits vorhandenes VHDL-Projekt eingebunden und gesteuert werden. Die VHDL-Dateien werden wie ein neues Modul eingebunden und dann mittels Signale von Wishbone angesprochen und angesteuert. Es muss darauf geachtet werden, dass die entsprechenden Wishbone-Signale zur richtigen Zeit angesprochen und gesetzt werden, damit die Kommunikation überhaupt funktionieren kann. Ist dies gegeben, so kann der Core mit seiner eigentlichen Funktion angewendet werden (siehe Bild 3.10).

Das Bild 3.4 aus der Analyse weist deutlich mehr Signale auf, als hier verwendet wird. Für dieses Modul werden nur die benötigt, die auch in diesem Bild 3.10 gezeigt werden. Hierauf ist zu achten, dass die Signale *WB_CLK_I* und *CLK_I* dem Takt *CLK* entsprechen.

In diesem Bild ist dargestellt, dass die Komponente *Tracer* die komplette Steuerung des Interfaces übernimmt. Sie erhält nur zwei Signale (*IntTx_O* und *IntRx_O*), anhand denen sie weiß, ob neue Daten verschickt und ob neue Daten entgegengenom-

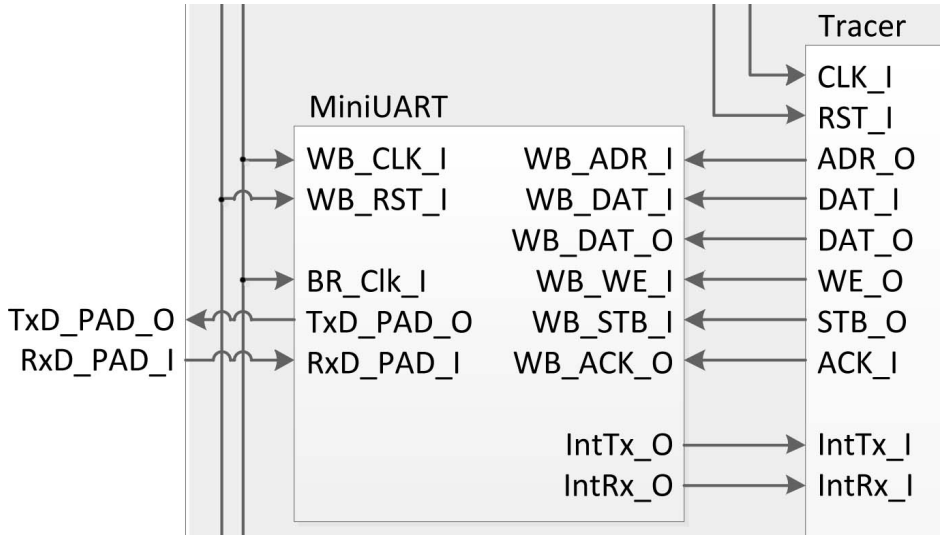


Bild 3.10: *Benötigte Signale für die Ansteuerung mittels Wishbone; Auszug aus B.2*

men werden können. Die Komponente *MiniUART*, die für die RS232-Schnittstelle zuständig ist, erhält den Takt, mit der sie arbeitet und die beiden Signale für die Kommunikation (*TxD_PAD_O* und *RxD_PAD_I*). Diese beiden Signale sind mit dem PC verbunden.

Damit die RS232-Schnittstelle stabil läuft, ist eine Veränderung des originalen Cores nötig. Bei einem StopBit gab es Probleme, dass Daten nicht übertragen wurden. Dies konnte durch hinzufügen eines zweiten StopBits gelöst werden. Dazu musste im „RxUnit“ und „TxUnit“ die Datenlängen verändert werden, sodass zwei Stopbits statt nur einem übertragen und empfangen werden und richtig ausgewertet werden können.

3.2.2.2 Steuerpfad

Zur Steuerung dieses Cores für das RS232-Interface wurde eine eigene Komponente entwickelt, die für das Senden, Empfangen und das entsprechende Auswerten der Daten zuständig ist. Dies wird über zwei Zustandsautomaten realisiert, die voneinander abhängig sind. Damit lässt sich die Steuerung auf eine sichere Art und Weise herrichten. Zur Verständlichkeit werden die beiden Zustandsautomaten grafisch dargestellt (Bild 3.11 und Bild 3.12) und näher erläutert.

Der zweite Zustandsautomat entscheidet, ob gelesen oder geschrieben wird und wertet gewisse empfangene Daten dafür aus. Der erste Zustandsautomat steuert anhand des zweiten Automaten das Lesen und das Schreiben auf die Leitungen für die RS232-Verbindung.

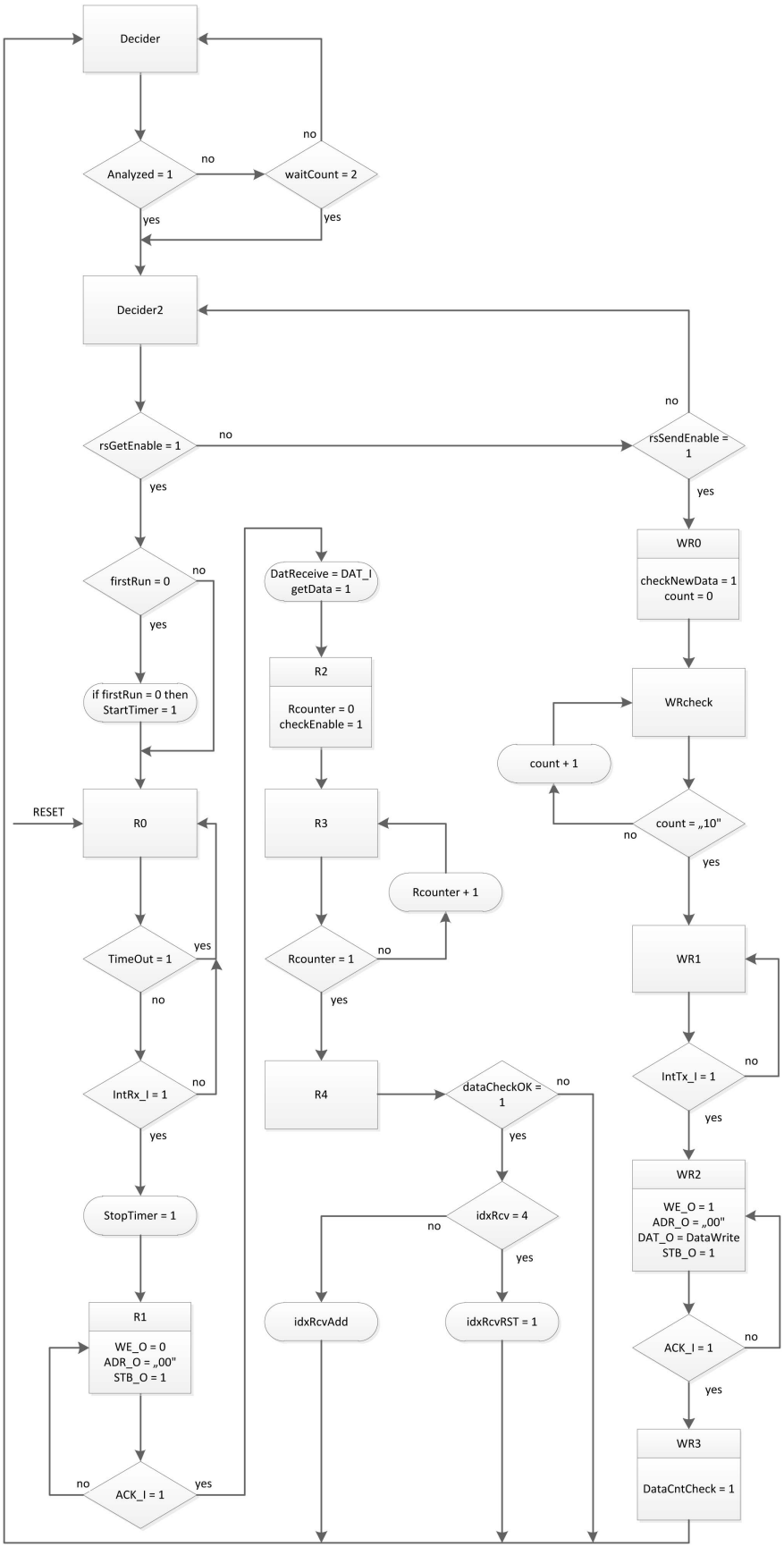


Bild 3.11: *Ablaufdiagramm des Zustandsautomaten für das Senden und Empfangen via der RS232-Schnittstelle*

- **Decider**
  Im Decider wird entweder zwei Takte oder auf das Signal *analyzed* gewartet. Das Signal *analyzed* wird gesetzt, sobald die angekommenen Daten von Matlab ausgewertet wurden (siehe Bild 3.12).
- **Decider2**
  Hier werden zwei Signale überprüft. Ist das Signal *rsGetEnable* gesetzt, so wird in den Zustand R0 übergegangen, sodass neue Daten von Matlab empfangen werden können. Handelt es sich um den ersten Durchlauf, wird der Timer gestartet. Ist stattdessen das Signal *rsSendEnable* gesetzt, wird in den Zustand WR0 übergegangen, sodass die aufgenommenen Daten vom ADC an Matlab übertragen werden. Ist keines der Signale gesetzt, wird in diesem Zustand gewartet, bis eines der beiden wieder gesetzt ist.
- **WR0**
  Mit dem Signal *checkNewData* werden die zu sendenden Daten bereitgestellt. Weiterhin wird der Datenzähler *count* zurückgestellt.
- **WRcheck**
  Dies ist ein Wartezustand, der zwei zusätzliche Takte wartet, bis die zu sendenden Daten vorliegen.
- **WR1**
  Ist der Wishbone-Puffer leer, wird in den nächsten Zustand gewechselt.
- **WR2**
  Hier werden die entsprechenden Signale für Wishbone gesetzt, sodass die Daten übertragen werden können. Wurden die Daten an Wishbone richtig übergeben, kommt eine Bestätigung (*ACK_I*) vom MiniUART und es geht mit Zustand WR3 weiter.
- **WR3**
  Es wird das Signal *DataCntCheck* gesetzt. Dadurch wird kontrolliert, welche Daten als nächstes bereitliegen sollen. Fortgesetzt wird im Zustand Decider.
- **R0**
  Ist das Signal *TimeOut* gesetzt, wird in diesem Zustand geblieben. Wurde von Matlab ein Byte an das FPGA gesendet, wird das Signal *IntRx_I* gesetzt. Dadurch wird in den nächsten Zustand R1 übergegangen.
- **R1**
  Es werden die Signale für Wishbone zur Steuerung des MiniUARTs gesetzt. mit dem Signal *Acknowlegde* wird das empfangene Byte in der Variablen *DatReceive* gespeichert und das Signal *getData* gesetzta. Forgesetzt wird im Zustand R2.

- **R2**
  In diesem Zustand wird ein interner Zähler zurückgesetzt, der für den nächsten Zustand benötigt wird. Das Signal *checkEnable* wird gesetzt, sodass das Byte überprüft werden kann, um was für ein Byte es sich handelt. Dieses Signal ist nur beim ersten empfangenen Byte interessant. Es wird überprüft, ob es sich um das Zeichen $0x23$ handelt, welches das erste Zeichen der Datenfolge entspricht.
- **R3**
  Hier werden zwei Takte gewartet, bevor in den Folgezustand R4 übergegangen wird.
- **R4**
  In R4 wird anfangs überprüft, ob das allererste Byte korrekt ist. Ist dies der Fall, wird der Index *idxRcv* zurückgesetzt, wenn dieser auf 4 steht, ansonsten wird er um 1 erhöht. Ist dies getan, wird der Anfangszustand *Decider* wieder aufgerufen.
- **WaitForData**
  In diesem Zustand wird auf das erste Byte gewartet. Der Folgezustand ist *DataCheck*.
- **DataCheck**
  Im Zustand *DataCheck* wird kontrolliert, ob das erste empfangene Byte dem Wert $0x23$ entspricht, welches das Startbyte für die Kommunikation der seriellen Schnittstelle ist. Ist das Datenbyte richtig, dann kann in den nächsten Zustand *receiveAll* übergegangen werden.
- **receiveAll**
  Hier wird gewartet, bis insgesamt 5 Byte empfangen wurden. Ist dies getan, wird in den Zustand *waitPart* gewechselt.
- **waitPart**
  Dies ist eine Wartefunktion, die auf den Index $= 0$ wartet. Somit ist dafür gesorgt, dass alle Daten richtig angekommen sind und ausgewertet werden können. Der Folgezustand ist *allReceived*.
- **allReceived**
  Hier wird das Signal *rsGetDone* gesetzt, welches besagt, dass das Auswerten nun möglich ist. Der nächste Zustand ist *waitDone*.
- **waitDone**
  In diesem Zustand wird darauf gewartet, dass die Daten ausgewertet wurden (Signal *analyzed*$= 1$), sodass in den nächsten Zustand *nextWork* übergegangen werden kann.

Bild 3.12: *Zustandsautomat für das Auswerten der empfangenen Daten*

- **nextWork**
  In *nextWork* werden die Daten *ProgReg* abgefragt. Haben die eines der Werte $0xC$, $0xD$ oder $0xF$, dann geht es in den Zustand *waitDataFull*. Wenn es andere Daten sind, dann geht es in den Zustand *enableReceive*.

- **waitDataFull**
  Hier wird abgewartet, bis das Array komplett mit neuen Daten vom ADC gefüllt ist oder die angeforderten Daten vorliegen. Folgezustand ist *waitForSent*.

- **waitForSent**
  In diesem Zustand wird das Signal *rsSendEnable* gesetzt, sodass die Daten nun an Matlab gesendet werden können. Ist dies getan (*MatlabDone* ist gesetzt), dann wird in den Zustand *enableReceive* gewechselt.

- **enableReceive**
  Dieser Zustand setzt das Signal *rsSendEnable* zurück, während das Signal *rsGetEnable* gesetzt wird, sodass wieder neue Daten empfangen werden können. Der Folgezustand ist wieder der vom Anfang (*WaitForData*).

Diese Kommunikation setzt voraus, dass von Matlab aus fünf Bytes gesendet werden müssen, wovon zwei Byte die entsprechenden Daten sind. Genaueres zu dem Aufbau siehe Abschnitt 4.1.1. Dies liegt daran, dass nur die Daten ausgewertet werden.

Beim Senden sieht es anders aus. Da werden so viele Daten gesendet, wie die Aufgabe benötigt. Sollen neue aufgenommene Werte der Signale gesendet werden, betragen die zu übertragenden Daten insgesamt 393 Bytes inklusive den Steuerbytes und dem *LF*. Werden dagegen nur die Registerinhalte übertragen, so sind es 26 Bytes. Wird nur ein Register übertragen, so sind es sechs Bytes, die an Matlab gesendet werden.

3.2.3 Zusatzfunktionen

3.2.3.1 Timeout

Werden im FPGA Daten von MATLAB entgegengenommen und während des Prozesses die Verbindung aus irgendwelchen Gründen unterbrochen, so greift der Timeout ein. Nach jedem empfangenen Byte wird der Timer erneut gestartet. Passiert auf der Verbindung nichts mehr, d.h. es kommen keine Daten an, so läuft der Timer über und das Timeout-Bit wird gesetzt. Dieses Bit sorgt dafür, dass dieser Empfangsprozess unterbrochen wird und zurück in den Anfangszustand gewechselt wird, sodass Daten wieder empfangen werden können. Dieser Zustand ist *R0* im Zustandsautomaten in Bild 3.11. Weiterhin wird der zweite Zustandsautomat (siehe 3.12) ebenfalls zurückgesetzt und in den Anfangszustand versetzt. Die Folge davon ist das Verwerfen der bereits empfangenen Daten.

3.2.3.2 Periodendauer

In diesem Abschnitt wird gezeigt, wie die Periodendauer bestimmt wird. Hierzu gibt es zwei Varianten, die näher beschrieben werden. Die erste Variante bestimmt die Periodendauer schneller und weniger genau.

Die erste Variante benötigt einen Komparator, der genau in der Mitte des Signals eingestellt ist. Nach einer halben Periode, schlägt der Komparator um. Die Zeit

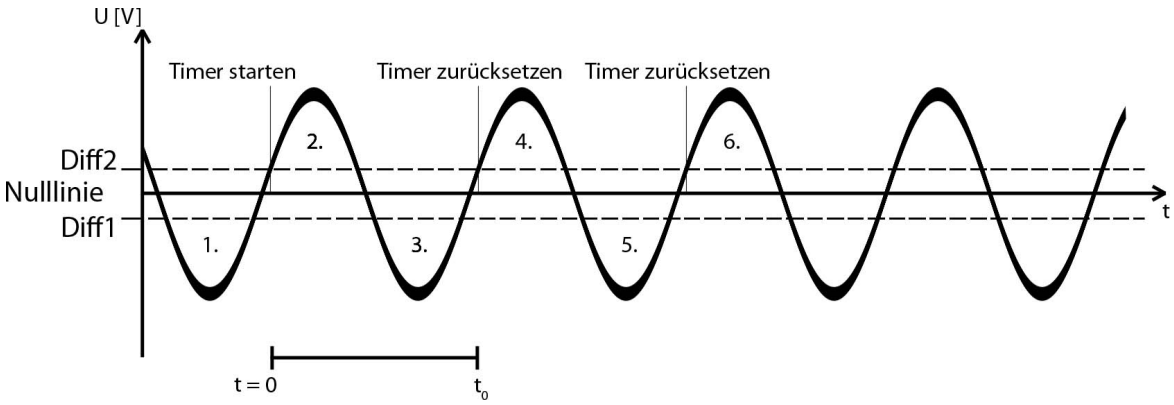


Bild 3.13: *Bestimmung einer Periode*

wird gemessen und hochgerechnet, wie lang genau eine Periode ist. Davon ausgehend wird ein 1/64-stel der Periode als zeitlicher Abstand berechnet.

Stattdessen wird die zweite Variante eingesetzt. Es werden zwei Komparatoren benötigt, der eine Komparator ist unterhalb der Offsetspannung eingestellt und der andere oberhalb des Offsets. Sobald beide Komparatoren gesetzt sind, d.h. das Eingangssignal ist größer als beide Spannungsschwellen *DIFF1* und *DIFF2*, wird der Timer gestartet. Dann muss das Eingangssignal die beiden Komparatorschwellen unterbieten. Sobald es wieder oberhalb der beiden Schwellen kommt, wird der Timer zurückgesetzt, d.h. die gestoppte Zeit gespeichert und für die Berechnung verwendet und den Timer erneut von 0 gestartet. Das Eingangssignal muss wieder beide Komparatoren unterbieten und anschließend erneut überbieten, damit der Timer erneut gestoppt, die neue Zeit gespeichert und erneut zurückgesetzt wird (siehe Bild 3.13). Dies wiederholt sich für jede weitere Periode, sodass immer eine aktuelle Zeit für die Frequenz vorhanden ist. Dies ist notwendig, da sich die Frequenz ständig ändert aufgrund der unterschiedlichen Geschwindigkeiten des Autos bzw. der Räder.

3.2.3.3 Speicherung neuer Daten

Nachdem die Periodendauer bestimmt worden ist (siehe Abschnitt 2.1.5), können die Daten anhand der Periodendauer aufgenommen werden. Es soll eine komplette Schwingung mit insgesamt 64 Werten aufgenommen werden. Dafür wird die bestimmte Zeit durch 64 geteilt. Dadurch ergibt sich die Zeit zwischen den aufzunehmenden Werten. Diese Zeit wird durch den Takt (80ns) geteilt. Das Ergebnis ist die Anzahl der benötigten Takte zwischen den einzelnen Werten. Nach Ablauf der Anzahl der Takte wird ein neuer Wert gespeichert. Der Zähler fängt erneut von vorne an, bis die Anzahl der Takte wieder erreicht ist. Dies wiederholt sich so lange,

bis der Speicher mit ihren 64 Werten für jedes der drei Signale, d.h. also 192 Werte, belegt ist. Anschließend wird das Signal *DataFull* gesetzt, welches das Zeichen für das Senden der Daten mittels des RS232-Interfaces gibt. Befindet sich das Interface nicht im Empfangsmodus, kann mit dem Senden begonnen werden (siehe Abschnitt 3.2.2.2).

3.2.3.4 Ausgabe der Daten auf der FPGA-Umgebung

Die Ausgabe der Daten erfolgt über der 7-Segment-Anzeige. Hier können die wichtigsten Register angezeigt werden sowohl vom ADC und DAC als auch von drei aufgenommenen Eingangssignale (Differenzsignal und die beiden Halbbrückensignale). Mittels der Switches (Umschalter) ist es möglich, zwischen den einzelnen Registern zu wechseln (siehe Tabelle 3.2). Für das Darstellen der Arrays wird eine weitere Funktion (*ArrayTest*) benötigt. Sie wird mittels der ersten drei der vier letzten Taster (siehe Bild 2.8) des Nexys2-Boards gesteuert. Die erste Taste sorgt dafür, dass zwischen den Signalen gewechselt wird, die nachfolgenden beiden sorgen dafür, dass in den Arrays Zeile für Zeile vor- und zurückgesprungen werden kann. Die drei Taster sind erst aktiviert, wenn der letzte Schalter auf „ON“ steht. Die Reihenfolge für das Wechseln der Arrays ist *U_DIFF*, *U_HB1* und *U_HB2*. Nach einem weiteren Druck auf die erste Taste springt sie wieder zu dem ersten Signal *U_DIFF*.

Switch-Code	Ausgabe	Kommentar
000	DB_UDIFF	aktueller Eingangswert
001	DB_HB1	aktueller Eingangswert
010	DB_HB2	aktueller Eingangswert
011	ADCsequence	gespeicherter ADC-Wert
100	ADCrefOut	gespeicherter ADC-Wert
101	ref_diff1	gespeicherter DAC-Wert
110	ref_diff2	gespeicherter DAC-Wert
111	ArrayValue	Ausgabe der Arrays

Tabelle 3.2: *Switch-Codierung für die Ausgabe der Daten für die 7-Segment-Anzeige*

Diese Möglichkeit der Ausgabe dient in erster Linie zu Testzwecken und zu einem schnellen Überblick ausgewählter Daten.

Über die Pins JB0 bis JB5 kann auf einem Oszilloskop dargestellt werden, in welchem Zustand sich die Steuer-FSM befindet. Sie ist auf 6bit codiert. Durch diese Zustandsausgabe lässt sich der Zustandsautomat kontrollieren (siehe Tabelle B.1 im Anhang).

4 Software - Matlab

In diesem Kapitel geht es um den Software-Bereich mit Matlab. Es beginnt der Kommunikation mit dem FPGA und die Datenauswertung der empfangenen Daten vom FPGA und sonstigen umgesetzten Funktionen in Matlab.

4.1 Kommunikation mit dem FPGA

Für die Kommunikation mit dem FPGA wird eine Kombination aus m-File und Treiber (siehe Abschnitt 4.1.2) verwendet. Mit der m-File (ADC_Board_Control.m) wird die komplette Steuerung und Auswertung durchgeführt. Bei der Kommunikation mit dem FPGA wird der Treiber in der m-File aufgerufen. Es werden Daten übergeben und es kommen Daten wieder zurück, die mit dem FPGA ausgetauscht werden.

Für die Kommunikation muss zuerst die Schnittstelle eingerichtet werden. Es wird ein Objekt erstellt und anschließend mit dem Befehl *connect()* die Verbindung aufgebaut. Dies wird im Programmausdruck 4.1 gezeigt.

```
1  interfaceObj_ADC_Board_Control = instrfind('Name', 'Serial-COM1');
2
3  if isempty(interfaceObj_ADC_Board_Control)
4      interfaceObj_ADC_Board_Control = serial('COM1');
5  else
6      fclose(interfaceObj_ADC_Board_Control);
7      interfaceObj_ADC_Board_Control = interfaceObj_ADC_Board_Control(1);
8  end
9
10 deviceObj_ADC_Board_Control = icdevice('ADC_Board_Control.mdd', ...
11     interfaceObj_ADC_Board_Control);
12 connect(deviceObj_ADC_Board_Control);
```

Programmausdruck 4.1: *Aufbau der RS232-Verbindung*

Die Ansteuerung des Treibers funktioniert mit Hilfe der Funktionen *set()* und *get()*. Die Codezeile 1 des Programmausdruckes 4.2 zeigt den Aufruf des Treibers mit

der Aufgabe, das DAC-Register des ADCs auszulesen und diese an Matlab zu senden.

```
1 ADCdone   = get(deviceObj_ADC_Board_Control.Control,'ADCrefOut');
2 ADCrefOut = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');
```

Programmausdruck 4.2: *Aufruf zum Auslesen des ADCrefOut-Registers*

Die empfangenen Daten werden mittels der zweiten Codezeile (siehe Programmausdruck 4.2) vom Treiber zur m-File übertragen. Die Funktion *invoke()* ruft hier die selbsterstellte Funktion *read_channels()* auf und wertet die empfangenen Daten vom FPGA aus. Die Daten werden von den acht Byte großen Datenstücken zu den entsprechenden Größen wieder zusammengefügt. Diese fertigen Daten werden anschließend an das m-File zurückgegeben, die anschließend in der Variablen *ADCrefOut* gespeichert werden. Hierbei handelt es sich um einen zwei Byte großen Wert.

Die von *invoke()* kommenden Daten können weiter ausgewertet, bearbeitet und dargestellt werden.

Wird statt des Befehls *get() set()* angewendet, wird die *invoke()*-Funktion im Prinzip nicht benötigt. Allerdings muss sie eingesetzt werden, da der Treiber automatisch dem Befehl *set() get()* voranstellt. Das Problem, das sonst entstehen kann, ist dass die gelieferten Daten vom Treiber nicht mit den tatsächlichen angeforderten Daten entsprechen bzw. übereinstimmen.

4.1.1 Datenabfolge

Für die Kommunikation mit Hilfe der RS232-Schnittstelle gilt es eine bestimmte Zeichenfolge zu beachten, damit der jeweiligen Seite des Senders bzw. Empfängers bekannt ist, wie viele Daten (hier Bytes) übertragen werden. Zusätzlich wird ein Timeout bei solchen Verbindungen bzw. Kommunikationen benötigt, die in VHDL wiederum selber umgesetzt werden müssen. In Matlab ist dies schnell mittels der Eigenschaft *Timeout* eingestellt (siehe Programmausdruck 4.6 in Abschnitt 4.1.2.4).

Die Darstellung "*#ZCD*" steht für:

#	=	Einleitendes Zeichen für die Funktionen *binblockread()* und *binblockwrite()*
Z	=	Anzahl der folgenden Ziffern für die Anzahl der Daten in Bytes
C	=	Anzahl der Daten in Bytes
D	=	Daten in Bytes

Wenn beispielsweise das DAC-Register des ADCs beschrieben werden, so sieht die Folge wie folgt aus:

$$\#ZCD = \#12A3FF \tag{4.1}$$
$$\# = \# \tag{4.2}$$
$$Z = 1 \tag{4.3}$$
$$C = 2 \tag{4.4}$$
$$D = 1.Byte: A3,\ 2.Byte: FF \tag{4.5}$$

Es muss Matlab in der Funktion *binblockwrite()* nur die beiden Bytes 'D' übergeben werden. Alle weiteren Einstellungen werden von Matlab automatisch erzeugt und ausgeführt.

Alle Zeichen werden in Bytes im ASCII-Format übertragen. Somit ergibt sich für die einzelnen Zeichen folgende Übertragungsdarstellung (siehe Tabelle 4.1):

Zeichen	ASCII(hex)
#	0x23
1	0x31
2	0x32
A3	0xA3
FF	0xFF

Tabelle 4.1: *Übertragung der Zeichen im ASCII-Format*

4.1.2 Treiber

Zur Vereinfachung der Kommunikation zwischen Matlab und dem FPGA wird ein Treiber von Matlab eingesetzt, das mit dem *MATLAB Instrument Driver Tool* eingerichtet wird. Dieser wird mit Funktionen ausgestattet, um die verschiedenen Register des ADCs und DACs zu beschreiben und auszulesen. Mit diesem Treiber können Mess- und Steuerinstrumente wie Oszilloskope oder wie in dieser Arbeit die RS232-Schnittstelle gesteuert und kontrolliert werden.

In dem Treiber werden die einzelnen Befehle und Funktionen eingerichtet und eingestellt, die für die Kommunikation, d.h. Senden und Empfangen, notwendig sind. In einer zusätzlichen m-File werden die in dem Treiber eingestellten Befehle nur durch Aufruf der Funktion verwendet. Dadurch bleibt das Programm bzw. die Steuerung von Matlab aus übersichtlich.

Die Befehle für die Steuerung des Steuerpfades des FPGAs sind in der Tabelle 4.2 aufgelistet. Der Befehl nimmt die ersten vier Bits der zwei Bytes in Anspruch. Wird einer der Befehle $0x1$ bis $0x8$, $0xA$ oder $0xB$ verwendet, so sind die restlichen zwölf Bits Daten, die in das entsprechend gewählte Register geschrieben werden sollen. $0xC$ ist der Befehl für das komplette Auslesen aller Register. Hier werden keine weiteren Daten an das FPGA gesendet bzw. sie werden gar nicht ausgewertet. Bei dem Befehl $0x9$ wird direkt ins *Config*-Register des ADCs geschrieben. Wird der Befehl $0xD$ übertragen, müssen die vier letzten Bits mit dem Befehl des entsprechend auszulesenden Registers belegt werden. Sie entsprechen den wie beim Schreiben ($0x1$ bis $0x8$ sowie $0xA$ und $0xB$). Wird bei dem Befehl $0xD$ die letzten vier Bits mit dem Befehl $0x9$ belegt, wird die Verstärkungsstufe *AMPgain* an Matlab übertragen.

ProgReg	Funktion
$0x0$	/
$0x1$	ref_diff1
$0x2$	ref_diff2
$0x3$	ref_hb1
$0x4$	ref_hb2
$0x5$	u_broff1
$0x6$	u_broff2
$0x7$	u_off
$0x8$	u_opt
$0x9$	ins *Config*-Register schreiben
$0xA$	ADCrefOut
$0xB$	ADCsequence
$0xC$	alle Register von ADC und DAC auslesen
$0xD$	ein spezielles Regsiter auslesen
$0xE$	Reset durchführen
$0xF$	neue Werte für eine Schwingung holen

Tabelle 4.2: *Programmierbefehle von Matlab und dem FPGA*

Ein Beispiel zur Veranschaulichung:

Es soll die Referenzspannung des ADCs programmiert und direkt danach wieder ausgelesen werden. Dafür wird in dem Treiber unter „Groups/Control/Properties“ der Eintrag *ADCrefOut* erstellt. Nach der Erstellung der neuen Eigenschaft kann sie mit einer Funktion versehen werden. In diesem Fall wird die M-Code-Methode eingesetzt, da es sich um einen FPGA ohne einer eigenen Steuerung handelt. Die Steuerung wurde für diese Arbeit selber erstellt, sodass hier nur der Matlab-Code zum Einsatz kommen kann.

Als erstes muss das Interface, d.h. die Schnittstelle mit dieser Funktion in Verbindung gebracht werden. Anschließend werden die Daten bereitgestellt, die in das FPGA geschickt werden sollen. In diesem Beispiel zum Programmieren der Referenzspannung wird die Datenfolge $'A3FF'$ mit der Matlabfunktion *binblockwrite()* geschrieben. Das $'A'$ steht in diesem Zusammenhang dafür, dass das Referenz-Register des ADCs programmiert werden soll. Die drei nachfolgenden Zeichen $'3FF'$ stehen für die zu schreibenden Daten.

Nachdem die Daten geschrieben worden sind, wird ein *binblockread()* durchgeführt, um die neuen Daten aus dem Register auszulesen. Dies wird dadurch erreicht, indem wieder ein *binblockwrite()* mit dem Inhalt $'D00A'$ an das FPGA geschickt wird. Das $'D'$ steht für ein einzelnes Register auslesen und das $'A'$ steht hier für das Referenz-Register des ADCs. Das FPGA liest das entsprechende Register aus und schickt diese Daten dann zurück an Matlab. Hier werden die Daten mit der *binblockread()*-Funktion entgegengenommen. Es handelt sich bei den Daten um zwei Bytes, die im BigEndian-Format übermittelt werden. Dadurch müssen diese mittels *bitshift* in die richtige Reihenfolge gebracht werden. Die Daten sind nun vollständig und können in Matlab weiterverarbeitet werden. In diesem Fall werden diese im Matlab-Fenster ausgegeben.

4.1.2.1 Funktion *set()*

Die Funktion *set()* (siehe Programmausdruck 4.3) wird mit dem Aufrufen des Objekts begonnen. Anschließend werden die beiden Bytes zusammengestellt. Dies wird mittels von Bitschieben erreicht. Das erste Byte setzt sich zusammen aus dem Befehl (hier 'A') (siehe Zeile 2) und den ersten vier Bits von dem zu schreibenden Daten (siehe Zeile 3). Zeile 4 setzt die beiden Einzelelemente als Byte zusammen. Die restlichen acht Bits werden für das zweite Byte erstellt (siehe Zeile 5). Mittels der *binblockwrite()*-Funktion werden die Daten *Byte1* und *Byte2* an das FPGA gesendet (siehe Zeile 6). Diese Funktion besitzt keinen Rückgabewert, da hier nur an das FPGA gesendet wird.

```
1 interface = get(get(obj,'parent'),'interface');
2 byte1a = bitand(bitshift(hex2dec('A'),4),hex2dec('F0'));
3 byte1b = bitand(bitshift(propertyValue,-8),hex2dec('0F'));
4 byte1 = bitor(byte1a,byte1b);
5 byte2 = bitand(propertyValue,hex2dec('FF'));
6 binblockwrite(interface, [byte1 byte2]);
```

Programmausdruck 4.3: *Funktion set()*

4.1.2.2 Funktion *get()*

Wie in der Funktion *set()* wird hier in der Funktion *get()* in der ersten Zeile die Verbindung zum dem Objekt hergestellt (siehe Programmausdruck 4.4). In Zeile 2 wird das erste Byte festgelegt mit dem Befehl $'D'$. Die Bedeutung der ersten vier Bits sind der Tabelle 4.2 zu entnehmen. Das zweite Byte setzt sich hier aus dem gleichen Befehl wie bei der Funktion *set()*- zusammen, nur befindet sich dieser Befehl in den letzten vier Bits des zweiten Bytes (siehe Zeile 3). Die Daten werden wie in der Funktion *set()* gesendet. Als Rückgabewert wird hier eine '1' eingesetzt. Sie sagt nur aus, dass die Daten gesendet wurden. Da nach diesem Schreibprozess Daten erwartet werden, kommt die Funktion *read_channels()* zum Einsatz (siehe Abschnitt 4.1.2.3).

```
1 interface = get(get(obj,'parent'),'interface');
2 byte1 = bitand(bitshift(hex2dec('D'),4),hex2dec('F0'));
3 byte2 = hex2dec('0A');
4 binblockwrite(interface, [byte1 byte2]);
5 propertyValue = 1;
```

Programmausdruck 4.4: *Funktion get()*

4.1.2.3 Funktion *read_channels()*

Diese Funktion *read_channels* (siehe Programmausdruck 4.5) wird im m-File durch den Befehl *invoke()* aufgerufen. Wie in den anderen Funktionen auch, wird die Verbindung zum Objekt aufgebaut. Anschließend werden die Daten mittels der *binblockread()*-Funktion entgegengenommen (siehe Zeile 3). Es wird in den nächsten Zeilen überprüft, ob es Fehler bei dem Empfangen aufgetreten sind. War der Empfang erfolgreich, werden die empfangenen Daten (*values*) wieder durch Bitschieben richtig zusammengesetzt (siehe Zeile 11 bis 19). Die Daten werden in der Variablen *out* zwischengespeichert und der Funktion *invoke()* im m-File zurückgegeben. Empfangen werden insgesamt bei neuen Werten der drei Signale *U_DIFF*, *U_HB1* und *U_HB2* 393 Bytes, von denen 387 verwendbare Daten sind, die an das m-File zurückgegeben werden. Die restlichen 6 Bytes sind für die Steuerung zuständig, wie oben bereits beschrieben. Hier sieht die Steuerfolge so aus: #3387387$Bytes$. Das 6.Byte ist die Endung $0x0A$ für den Abschluss des Sendens.

```
1  interface = get(get(obj,'parent'),'interface');
2
3  [values, bytesReceived, errorMessage] = binblockread(interface, 'uint8');
4
5  if (~strcmp(errorMessage,''))
6      error('error by using binblockread()-function');
7      disp(['error: ',errorMessage]);
8  end
9  fprintf('\nReceived Bytes: %d\n',bytesReceived);
10
11 for i=1:1:length(values)/2
12     if i<194
13         temp(2*i-1) = bitshift(cast(values(2*i-1),'uint16'),8);
14         temp(2*i)   = cast(values(2*i),'uint16');
15         out(i)      = bitor(temp(2*i-1),temp(2*i));
16     end
17     if i==193
18         out(i+1)    = cast(values(387),'uint16');
19     end
20 end
21 propertyValue = out;
```

Programmausdruck 4.5: *Funktion read_channels()*

Die maximale Anzahl an Daten beträgt 387 Bytes. Da sie ungerade ist, wird eine extra Abfrage in dieser Funktion eingebaut. Sie sorgt dafür, dass die Funktion keinen Array-Abfrage-Überlauf erzeugt, d.h. dass sie nicht auf ein Feld zugreift, das nicht existiert. Die Hälfte von 387 ist bei Integer-Werten 193, sodass hier die zusätzliche Abfrage in Zeile 17 hinzugefügt werden muss.

Diese Funktion ist sehr sinnvoll, da wenn sie nicht genutzt wird und die Zusammensetzung der Daten nicht bereits hier erfolgt, das m-File extrem lang und unübersichtlich wird.

4.1.2.4 Zusätzliche Einstellungen

Damit der Treiber mit dem RS232-Interface richtig kommunizieren kann, muss dieser erstmals eingerichtet werden. Dies wird in dem Treiber in der Funktion „Initialization and Cleanup“ durchgeführt. Sie stellt zuerst die Verbindung zwischen der Schnittstelle und dem Objekt her (siehe Zeile 1 in Programmausdruck 4.6). Anschließend wird das Interface geschlossen, um die Einstellungen setzen zu können. Bei den Einstellungen handelt es sich um die Baudrate, Anzahl von Datenbits, Anzahl von StopBits, wie lange der Timeout laufen soll und wie eine Verbindung abgeschlossen wird (*Terminator*) (siehe Zeile 5 bis 10). Sind diese Einstellungen getätigt,

kann das Interface zusammen mit dem Objekt geöffnet werden, sodass die Verbindung bzw. die Kommunikation zwischen PC und FPGA genutzt werden kann.

```
interface = get(obj, 'Interface');
fclose(interface);

% Configure interface
set(interface,'BaudRate',19200);
set(interface,'DataBits',8);
set(interface,'Parity','none');
set(interface,'StopBits',2);
set(interface,'Terminator','LF');
set(interface,'Timeout',3);
```

Programmausdruck 4.6: *Initialisierung*

Wie weiterhin aus dem Programmausdruck 4.6 zu erkennen ist, werden die Verbindungen mit einem „linefeed“ (LF) abgeschlossen, das dem heximalen Zeichen $0x0A$ entspricht.

4.2 Datenauswertung

Die empfangenen Daten werden ausgewertet und in Arrays gespeichert. Hierbei handelt es sich um die drei Werte-Reihen *Udiff*, *HB1* und *HB2*. Wurden die Daten komplett empfangen, werden diese grafisch in einem Plot mit drei Darstellungsfenstern dargestellt (siehe Bild 4.1). Bevor sie aber dargestellt werden können, werden diese bearbeitet, sodass die HEX-Zahlen in die entsprechende Spannung umgewandelt wird, die auch vorne an den Eingangskanälen des ADCs anliegen (siehe Programmausdruck 4.7 und Formel 4.6).

```
udiff_vec(index,1) = udiff .* (Vmax./power(2,12));
```

Programmausdruck 4.7: *Umwandlung von HEX in Volt*

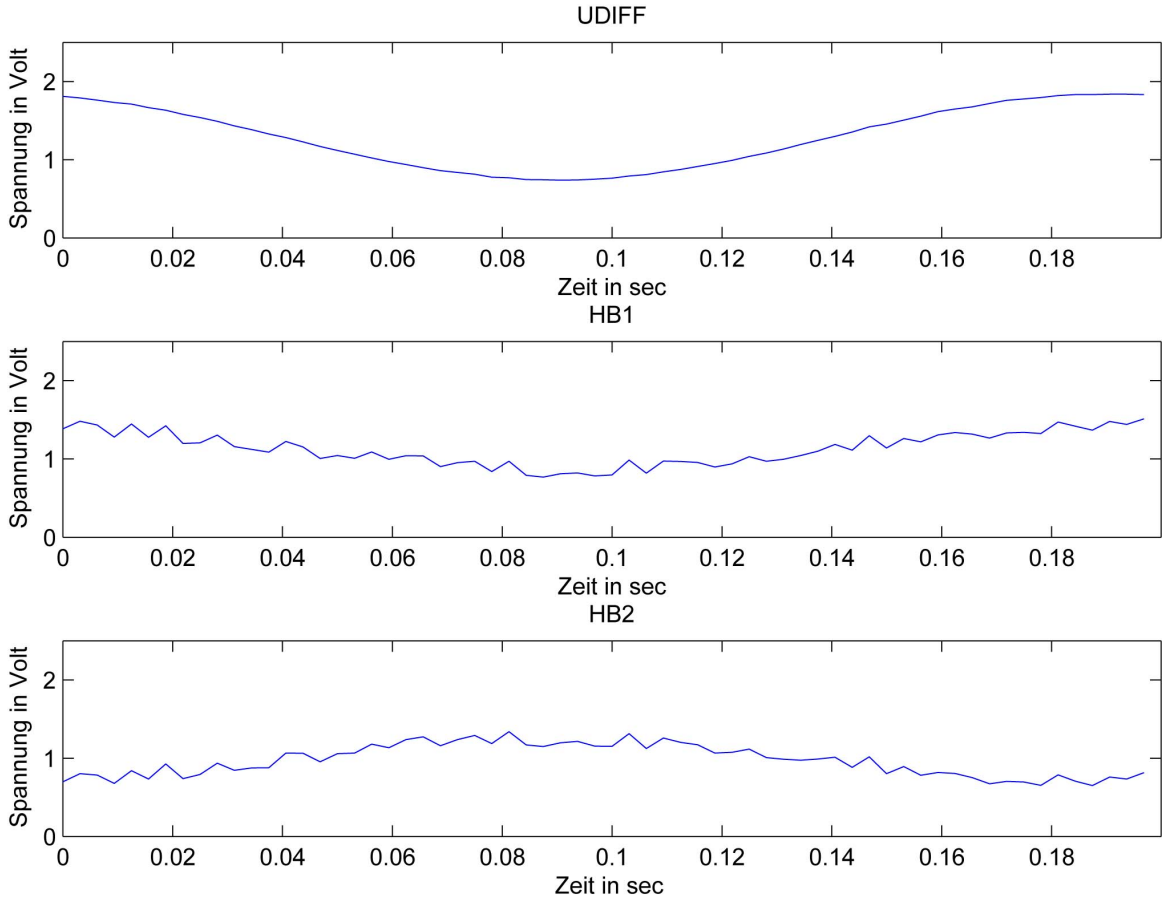


Bild 4.1: *Darstellung in Matlab*

$$\Rightarrow udiff\_vec(index,1) = udiff \cdot \frac{Vmax}{2^{12}} \tag{4.6}$$

mit

$$index = 1..64\ (integer) \tag{4.7}$$
$$1 \mathrel{\hat{=}} Spalte\ des\ Arrays \tag{4.8}$$
$$udiff = übertragener\ Wert \tag{4.9}$$
$$Vmax = 5V \tag{4.10}$$
$$power(2,12) \mathrel{\hat{=}} 2^{12} \tag{4.11}$$
$$\tag{4.12}$$

Diese Umrechnung bezieht sich ebenfalls auf die beiden anderen Signale.

Die Ausgabe der einzelnen Register wird in dem Hauptmenü von Matlab ausgegeben (siehe Tabelle 4.3).

Received Bytes:	26
ADCrefOut:	3FF
ADCsequence:	231
ref_diff1:	380
ref_diff2:	450
ref_hb1:	0
ref_hb2:	0
u_broff1:	0
u_broff2:	0
u_off:	0
u_opt:	0
gain:	7
frequence:	65535 Hz

Tabelle 4.3: *Registerausgabe von ADC und DAC durch das Matlab-Skript*

Zum Auslesen der Daten wird die Funktion *invoke()* verwendet. Mit dieser Funktion wird die Funktion *readchannels()* aus dem Treiber gestartet und ausgeführt. Als Rückgabewert liegen die Daten vom FPGA vor.

Die Steuerung durch Matlab hat den besonderen Vorteil, dass die Einstellungen der einzelnen Bauelementen schnell und einfach verändert werden können, welches die Ausführung bzw. Nutzung des Programms komfortabel gestaltet.

4.3 Umgesetzte Funktionen

Das m-File (ADC_Board_Control.m) erstellt alle benötigten Variablen und löscht alle Zwischenspeicher. Im nächsten Schritt wird das Objekt geöffnet, sodass die Verbindung aufgebaut werden kann. Nach der Verbindung kann mit dem Beschreiben und Auslesen des FPGAs begonnen werden.

Nachdem alle gewünschten Einstellungen gesetzt und Informationen geholt wurden, können diese grafisch dargestellt werden. Dazu dient die *plot()*-Funktion. Sie enthält insgesamt drei Darstellungen untereinander. Die x-Achse ist mit der Zeit angegeben. Diese ändert sich abhängig von der Frequenz des angeschlossenen Signals am ADC-Eingang auf der ADC-Platine. Die y-Achse ist in Volt aufgetragen.

Sie reicht von $0V$ bis $2,5V$. Mehr kann der ADC in diesem Aufbau nicht herbringen. Somit wird die Darstellung in diesem Bereich bleiben.

Die angezeigte Zeit auf der x-Achse wird jedes Mal neu berechnet, sobald neue Daten grafisch dargestellt werden sollen. Dazu sendet das FPGA hinter den drei Arrays noch zusätzlich die benötigte Zeit für eine ganze Periode (siehe Abschnitt 4.2).

Sollen die übertragenen Werte der Arrays also der drei Schwingungen mit den Hardware-Werten verglichen werden, gibt es hier eine weitere Funktion, die die übertragenen Werte in die HEX-Darstellung umwandelt. Dadurch lässt sich der Vergleich beschleunigen und vereinfachen.

Am Ende des Scripts wird die Verbindung zu dem Objekt und dem Interface wieder getrennt und die Objekte gelöscht.

5 Inbetriebnahme

In diesem Kapitel wird zum einen auf die Simulation eingegangen und zum anderen auf die Hardware-Funktionen. Am Ende gibt es einen Funktions- und Datenvergleich.

5.1 Simulation

Für die Simulation kommt das Programm ModelSim XE III 6.4b zum Einsatz. Dieses Programm überprüft die Syntax des VHDL-Codes. Ist sie fehlerfrei, kann sie compiliert werden, sodass die Simulation durchgeführt werden kann. Die Simulationen werden entweder durch ein .do-File oder mittels der Kommandozeile gestartet. Damit die Simulation funktioniert, müssen die zu testenden VHDL-Module mit Signalen versorgt werden. Dies wird entweder durch das Hinzufügen spezieller Signaldaten in der .do-File realisiert oder durch eine dazu entsprechend entworfene Testbench. Bei komplexeren Projekten ist der Einsatz von Testbenches empfohlen, da sich die VHDL-Module durch einen geringen Aufwand testen lassen, dessen Umsetzung in dieser Form in einer .do-File nur mit großem Aufwand möglich wäre. Mit Testbenches lassen sich ebenfalls komplette Module beschreiben, die nur in der Hardware zur Verfügung stehen, wie in dieser Arbeit der ADC und der DAC. Es wird weiterhin eine Testbench eingesetzt, die ein Sinussignal erzeugt ([10]). Die Nachbildung der Hardware-Komponenten in VHDL-Module (Testbenches) ermöglicht den kompletten Funktionstest der einzelnen Module, die später beim Hardware-Test in das FPGA eingesetzt werden. Somit können Eingangs- und Ausgangssignale angelegt bzw. geholt werden. In Bild 5.1 ist der Simulationsaufbau mit dem Toplevel, der im FPGA eingesetzt wird, und den Nachbildungen der Hardware-Komponenten (ADC, DAC, Eingangssignal und RS232-Interface) dargestellt.

In Bild 5.1 sind insgesamt fünf Elemente dargestellt. Es handelt sich zum einen um das FPGA *TB_Toplevel_TL* und zum anderen um Nachbildungen der Hardwareelemente, die sich auf der ADC-Platine befinden (ADC *TBadc_ads7865*, DAC *TBdac_ad5348*, Eingangssignal *tb_source* und Vorverstärker *SinGain*). Das Modul *tb_source* ist von [10] übernommen.

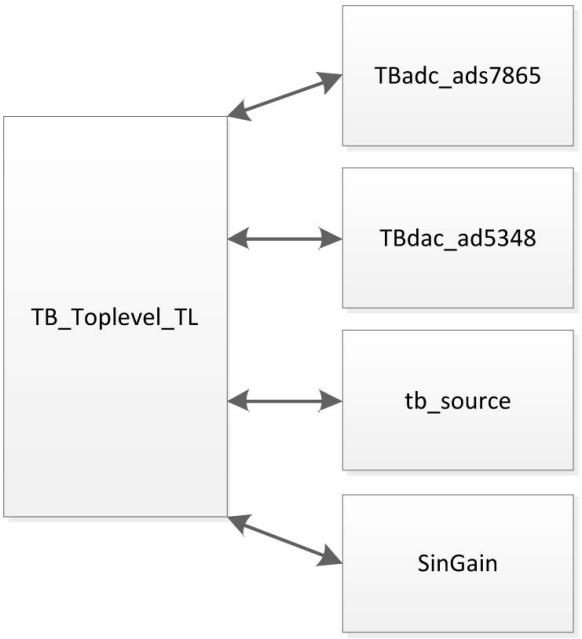


Bild 5.1: *Der Simulationsaufbau mit den nachgebildeten Hardware-Komponenten in VHDL-Module/Testbenches*

Es gibt zwei verschiedene Simulationsmethoden. Zum einen gibt es die funktionale Simulation, in der der VHDL-Code in seiner logischen Funktion ohne das physikalische Zeitverhalten überprüft wird, zum anderen gibt es die Simulation, die vom Timing abhängig ist, in der das physikalisch-simulierte Zeitverhalten der synthetisierten VHDL-Beschreibung betrachtet wird.

Diese beiden Simulationsmethoden sollen in den beiden Abschnitten 5.1.1 und 5.1.2 näher beschrieben werden. Anschließend gibt es einen Vergleich der beiden Methoden.

5.1.1 Funktionale Simulation

Bei der funktionalen Simulation handelt es sich um eine Simulation, anhand der die erforderlichen Funktionen der VHDL-Module überprüft werden.

5.1.1.1 Analog-Digital- und Digital-Analog-Converter

- Anhand des Bildes 5.2 ist der Ablauf des Schreibzyklusses des DAC-Registers des ADCs zu erkennen. Es werden die Signale *wr_n_en*, *adc_csn*, *adcgotodacreg* und *adcwrdacreg* so gesetzt, dass die Daten *writeto* sicher auf dem Datenbus *db*

anliegen, bevor das Signal *adc_wrn* zum Schreiben gesetzt wird. Die Signale *adcgotodacreg* und *adcwrdacreg* werden vom Steuerpfad dem Datenpfad übergeben, damit die richtigen Daten auf den Datenbus angelegt werden. Nach dem Schreibzyklus der beiden Werte wird das Signal *progregreset* gesetzt, das diese Aufgabe für erledigt erklärt. Es wird in den Zustand fürs Auslesen neuer ADC-Werte gewechselt und das Signal *adc_convstn* kann wieder gesetzt werden, sodass der Leseprozess fortgesetzt wird. Wichtig an dieser Stelle ist die Ausführung dieses Schreibzyklus bei dem 'low'-Signal von *adc_busy*. Der Bus *statepart* gibt Auskunft über den Zustand des Automaten.

- In Bild 5.3 wird die Initialisierungsphase gezeigt. Sie lässt sich in zwei Aufgabenbereich aufteilen. Die erste Phase ist die Schreibphase (siehe Bild A.4) und die zweite die Lesephase (siehe Bild A.5). Die beiden Phasen werden direkt hintereinander ausgeführt. Der Anfang der Initialisierungsphase besteht darin, alle Register des ADCs zu löschen. Dies wird durch das Schreiben des Wertes $0x005$ ausgeführt. Anschließend werden drei Takte gewartet, bis die Schreibphase beginnt. Das Beschreiben der ADC-Register ist oben bereits beschrieben.

- In der Lesephase (siehe Bild A.5) wird bei dem ADC zuerst das gewünschte auszulesende Register in das Config-Register geschrieben. Anschließend werden die Lesesignale gesetzt. Liegen die Daten sicher an, werden sie in die entsprechenden Variablen gespeichert. Nachdem die beiden Register des ADCs ausgelesen sind, werden die Register des DACs ausgelesen. Dazu werden die Daten zum Lesen bereitgestellt und ebenfalls mit dem Signal *readnow* in die Variablen abgelegt. Zu der Auswahl der DAC-Register wird das Bussignal *dac_a* entsprechend der 3Bit-Codierung eingestellt. Sind alle Register ausgelesen, wird das Signal *readallregister* zurückgesetzt.

- In Bild 5.4 ist der Prozess abgebildet, wie die ADC-Werte nach den Zyklen in die entsprechenden Arrays gespeichert werden. Es werden pro gesetztes *adc_busy*-Signal zwei neu konvertierte Werte bezogen, die ihren entsprechenden Variablen zugewiesen werden. Ist der erste Zyklus an der Reihe, d.h. die beiden ADC-Eingangssignale *UDIFF* und *HB1*, ist das Signal *conv_ch* auf 'low'. Der erste ausgelesene Wert soll in die Variable *db_udiff* gespeichert werden, sodass das Signal *part* ebenfalls auf 'low' liegt. Der zweite Wert gehört zu der Variablen *db_hb1*, sodass das Signal *part* auf 'high' gesetzt ist. Der zweite Zyklus enthält jeweils die beiden zweiten Kanäle der Eingangssignale, sodass hier erneut das angeschlossene Signal *UDIFF* und das Signal *HB2* ausgelesen werden. In diesem zweiten Zyklus wird das Signal *UDIFF* zwar ausgelesen, aber nicht gespeichert, da es nicht nach jedem Zyklus gebraucht wird. Das zweite übermittelte Signal wird in die Variable *db_hb2* gespeichert. Da jetzt

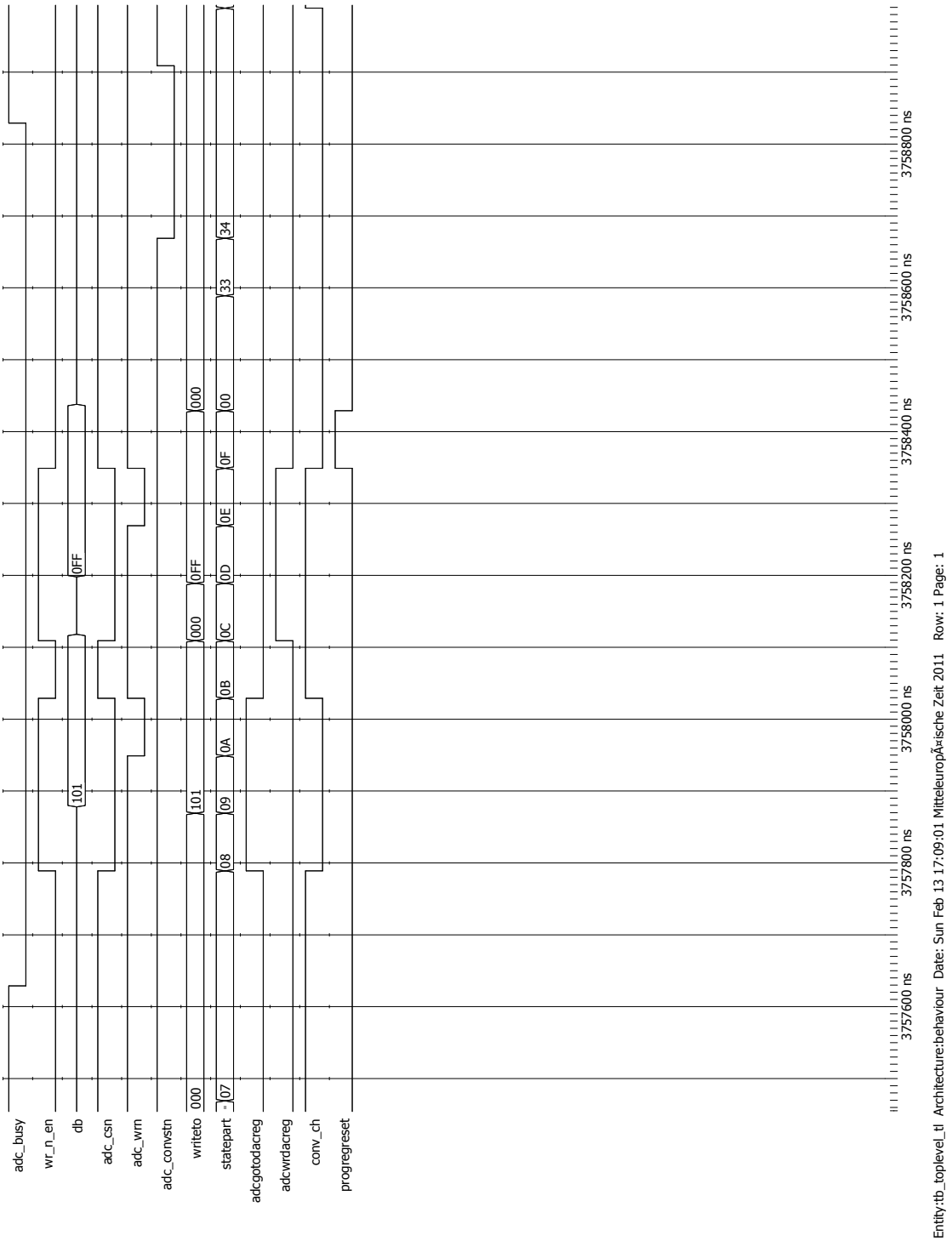


Bild 5.2: *Programmieren des DAC-Registers des ADCs*

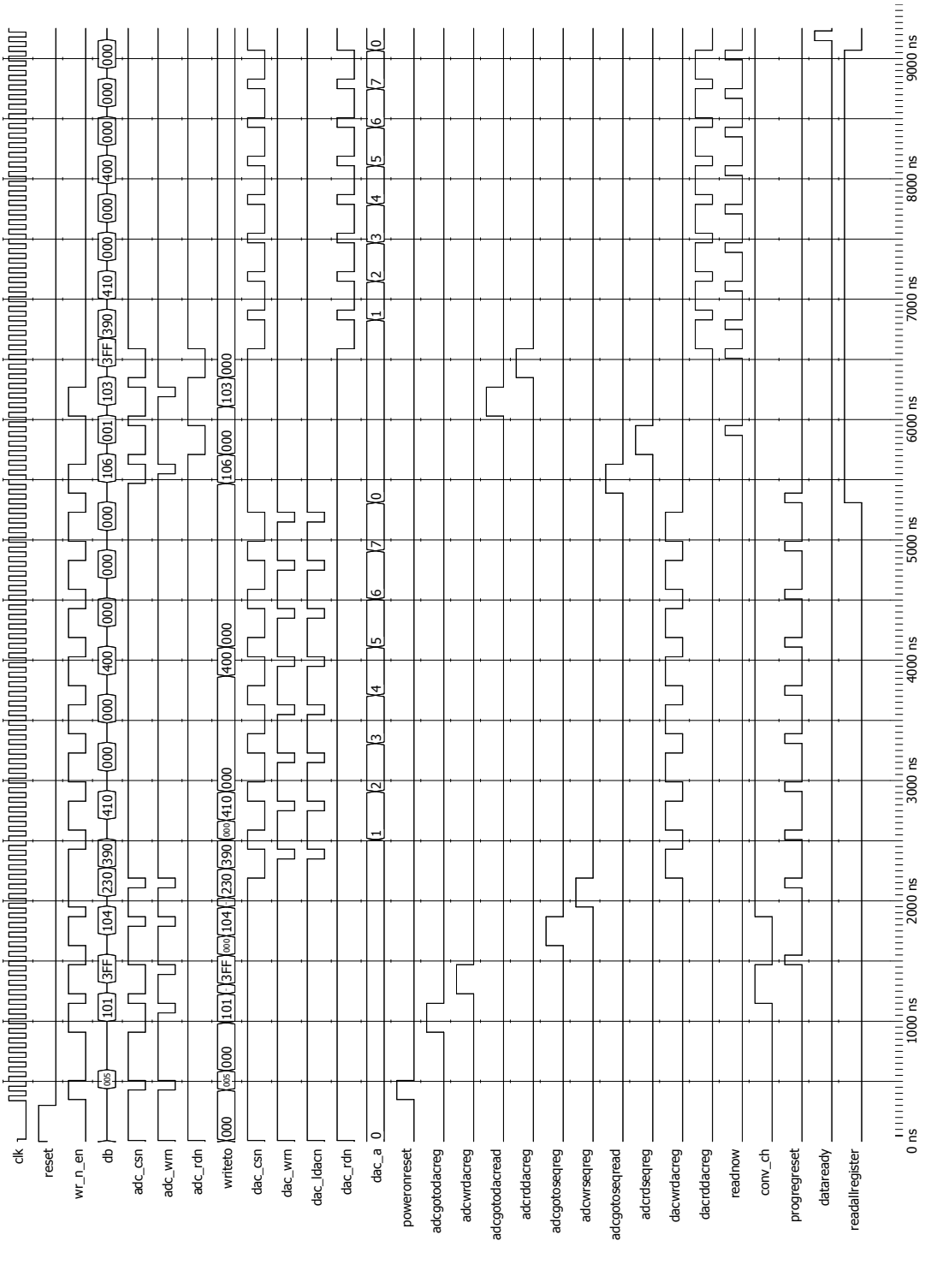


Bild 5.3: *Initialisierungsphase zum Programmieren und Auslesen der Register für ADC und DAC*

zwei Zyklen durchlaufen sind, wird erneut der erste Zyklus durchlaufen, sodass hier die beiden Werte für die Signale *db_udiff* und *db_hb1* gespeichert werden. Hiernach folgt erneut der zweite Zyklus. Diese Zyklen wechseln sich ununterbrochen ab.

- Wird der Befehl $0xF$ gegeben, bedeutet dies, dass die drei Eingangssignale des ADCs in den Arrays aufgenommen werden sollen. Es werden pro Array 64 Werte gespeichert. Nachdem der Befehl $0xF$ gesendet und diese von der Funktion *Tracer_Data* ausgewertet wurde, wird das Signal *analyzed* gesetzt (siehe Bild 5.5). Dies führt zum Setzen des Signals *reading*. Anschließend wird das *Sequence*-Register ausgelesen, um die letzten beiden Bits zu betrachten. Befinden sich die beiden Bits mit dem Muster „01“, wird als nächstes der erste Zyklus konvertiert. Dies leitet den Schritt zum Speichern ein (siehe Bilder A.6, A.7 und A.8 im Anhang). Ist ein anderes Muster ausgewertet, wird ein Zyklus abgewartet und erneut überprüft, bis das Muster „01“ auftritt. Sind alle drei Arrays mit neuen Werten gefüllt, wird das Signal *datafull* gesetzt. Dies sorgt dafür, dass keine weiteren Werte in den Arrays gesichert werden und dass das Übertragen der Werte an Matlab gestartet werden kann.

5.1.1.2 Periodendauer

- Für die Periodendauer werden die beiden Komparatorsignale *sigcompudiff1* und *sigcompudiff2* benötigt (siehe Bild A.9 im Anhang). Sobald diese beiden Signale auf 'high' gesetzt sind, welches durch Komparatorschwellen gelöst ist, wird der Zähler *timer* durch das Signal *timestart* gestartet. Beide Komparatorsignale müssen anschließend auf 'low' zurückgehen. Mit ihrem erneuten 'high' wird der Zähler mit dem Signal *timestop* gestoppt. Der Zähler wird gespeichert, durch 64 geteilt (die letzten sechs Bits werden abgeschnitten) und in die Variable *sampleintervall* gespeichert. Zum Aufnehmen einer neuen Schwingung mit genau 64 Werten wird die Variable *sampleintervall* eingesetzt (siehe Abschnitt 3.2.3.3).

5.1.1.3 Bestimmung der Verstärkung

- Werden die beiden Komparatorschwellen erreicht, wird die Variable *udiffmax* solange erhöht, wie das Signal größer wird. Umgekehrt gilt dies genauso für die Variable *udiffmin*, dass sie immer kleiner wird, wie das Signal verringert wird. Das Maximum wird verglichen, wenn beide Komparatorsignale 'low' ausgeben. Ist das Maximum zu groß, wird die Verstärkung verringert. Ist sie dagegen zu klein, wird sie erhöht (siehe Bild A.10 im Anhang). Die gleiche

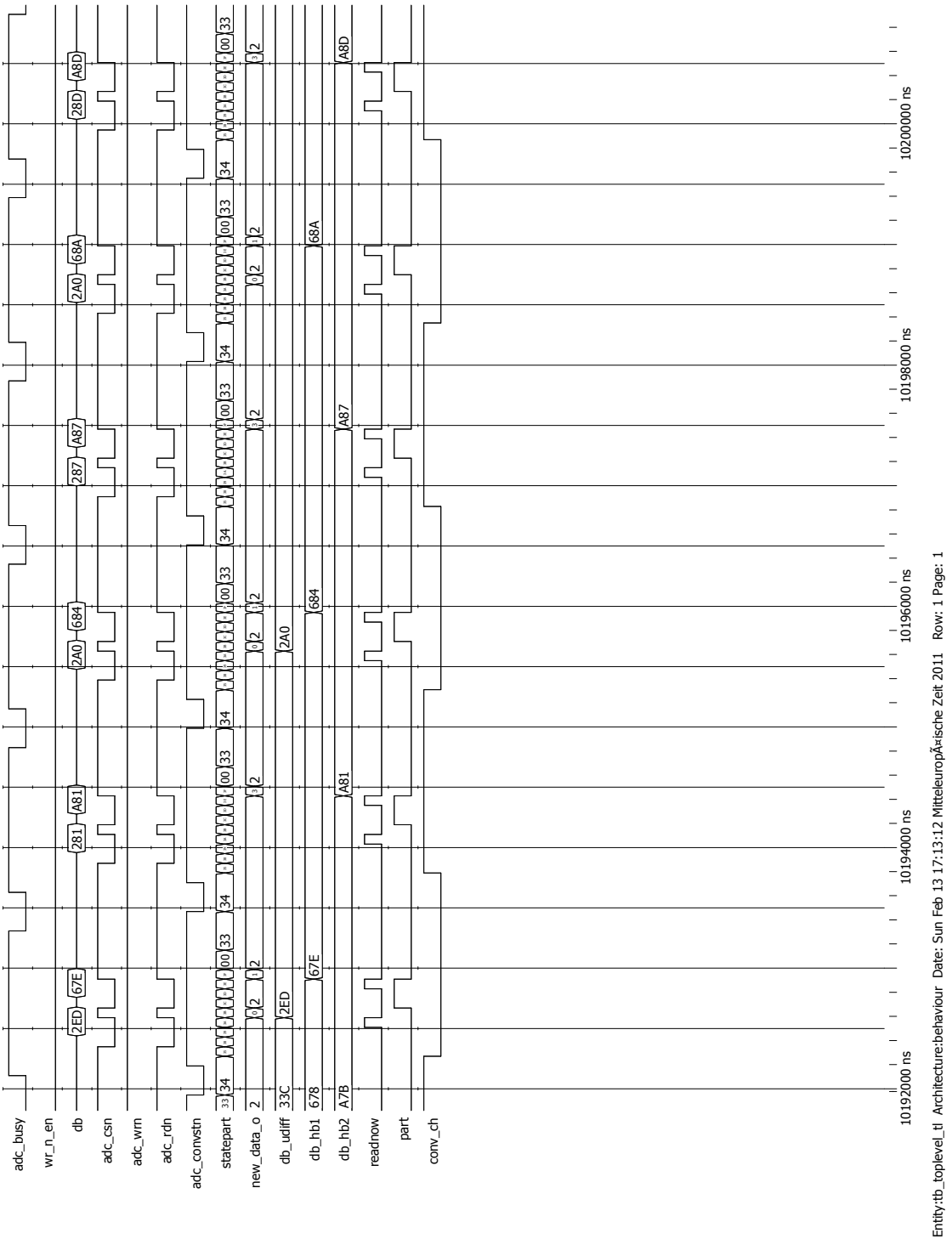


Bild 5.4: *Speicherung der ADC-Werte in Zyklen in die entsprechenden Arrays*

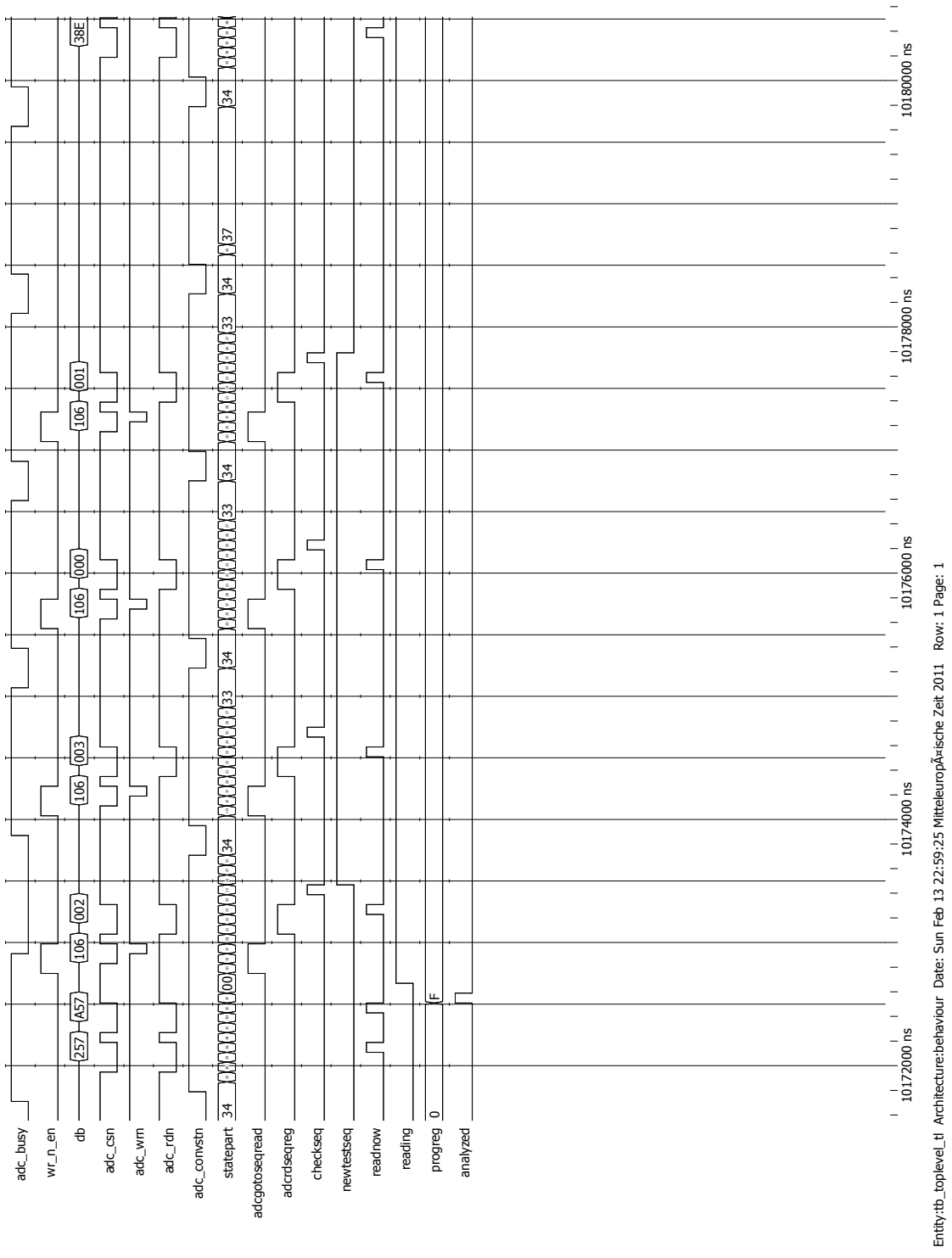


Bild 5.5: *Abfrage des Sequence-Registers*

Abfrage wird ebenfalls mit dem Minimum getätigt, sobald die beiden Komparatorsignale 'high' sind.

5.1.1.4 RS232-Interface

- Am Anfang des Bildes 5.6 werden Daten empfangen. Sobald sie ausgewertet wurden, wird das Signal *rsgetenable* zurückgesetzt, sodass keine weiteren Daten empfangen werden. Die empfangene Aufgabe wird ausgeführt. Am Ende, nachdem die zu sendenden Daten komplett sind, wird das Signal *dataready* gesetzt. Dadurch wird das Signal *rssendenable* gesetzt und die Daten werden gesendet.

- Die zu sendenden Daten müssen zusammengestellt werden, welches in Bild 5.7 dargestellt ist. Die Daten werden bei dem Setzen des Signals *p12_en* erneuert. Dadurch ändern sich in dieser Ausführung die Variablen *byte1* bis *byte6*. Dabei handelt es sich um den Auftrag $0xC$, der für das Auslesen aller Register und das anschließende Senden dieser Daten an Matlab sorgt. Wurden alle Daten gesendet, wird das Signal *sendingdone* gesetzt.

5.1.2 Timing Simulation

Bei der Timing Simulation geht es um eine Simulation, in der das physikalisch-simulierte Zeitverhalten der synthetisierten VHDL-Beschreibung betrachtet wird. Zur Vorbereitung wird das Programm *ISE Project Navigator* von der Firma *Xilinx* benötigt, das die zusätzliche Datei erstellt, die das zeitliche Verhalten des ganzen Ablaufs enthält (ada_conv_timingsim.sdf). Dies wird durch den Befehl *Generate Post-Place & Route Static Timing* in ISE ausgeführt. Anhand dieser erstellten Datei wird die Timing Simulation mit ModelSim durchgeführt. Ein großer Vorteil dieser Simulation ist der funktionale und tatsächliche Nachweis dafür, welches Verhalten die Hardware mit großer Wahrscheinlichkeit haben wird. Dadurch lassen sich schnell Fehler finden und verbessern.

Ein kritisches Problem besteht in der benötigten Zeit der Timing-Simulation. Die Simulationszeit für $1ms$ beträgt um die $6,5$ Stunden (siehe Abschnitt 5.1.3).

Ein weiterer Nachteil ist, dass alle Signale, die in der Toplevel mit der Hardware kommunizieren, in dem *std_logic*- bzw. *std_logic_vector*-Format sein müssen. Um nicht jedes Signal im ganzen Projekt ändern zu müssen, wird stattdessen eine weitere Funktion nur für diese Simulation eingesetzt, die die notwendigen Signale konvertiert.

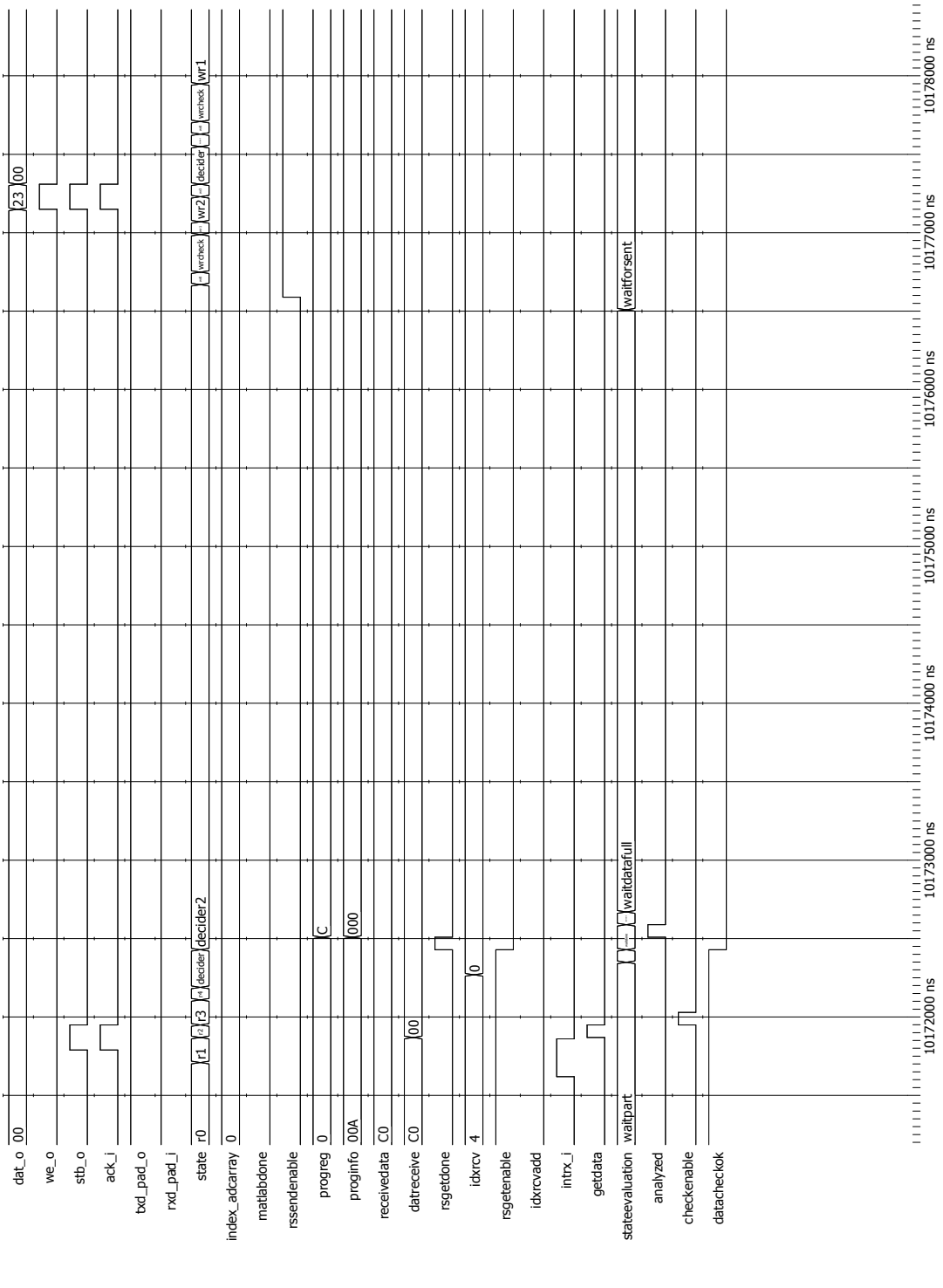


Bild 5.6: *Funktion der RS232-Schnittstelle*

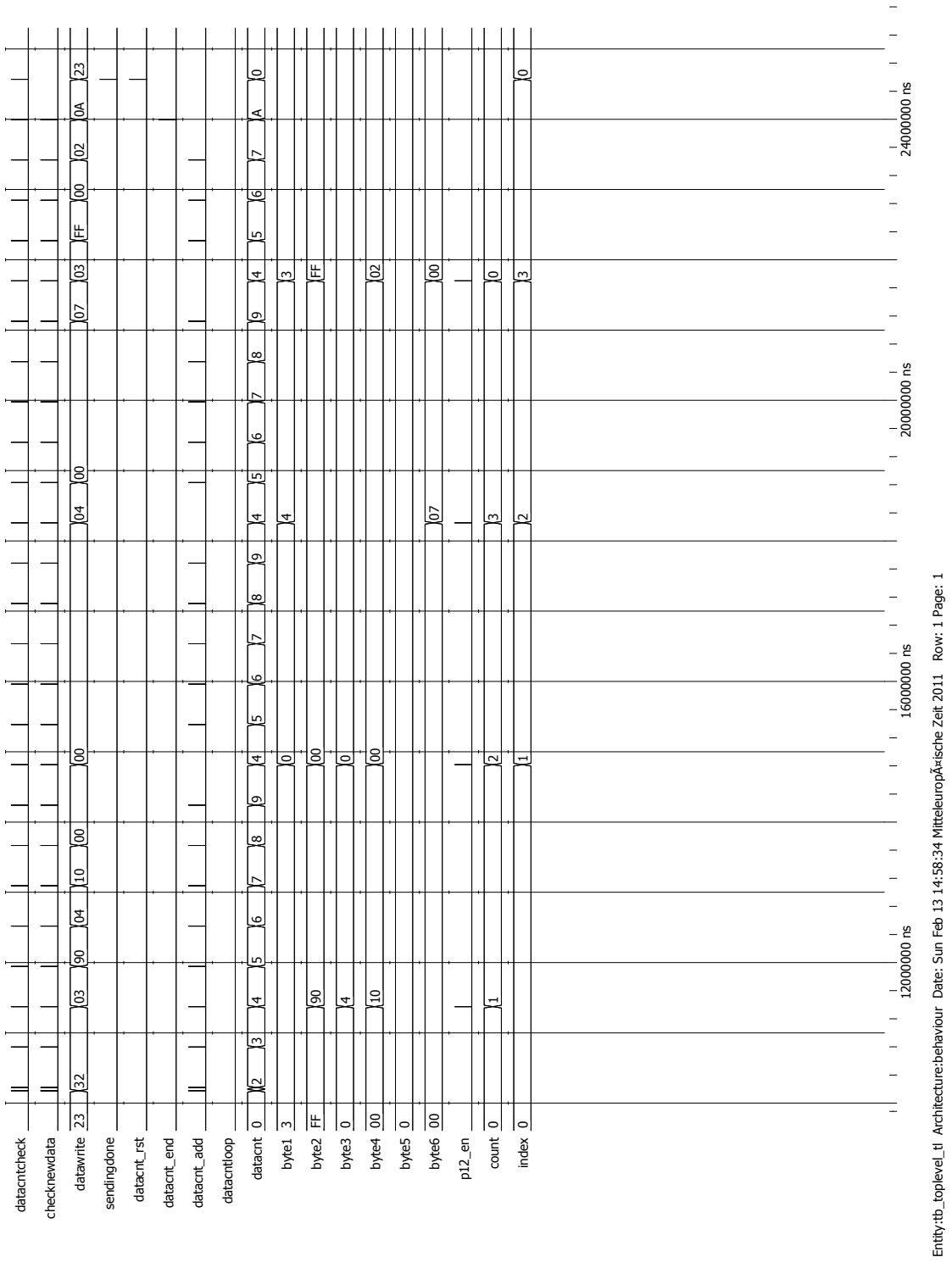


Bild 5.7: *Vorbereitung der zu sendenden Daten an Matlab*

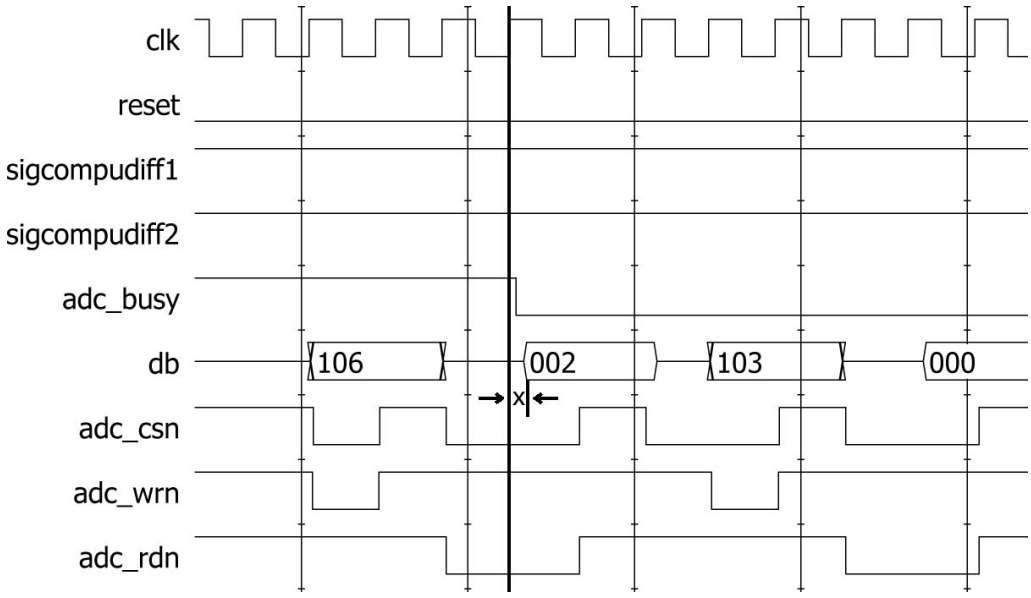


Bild 5.8: *Timing Simulation mit dem Problem der Datenlatenz*

Werden die Ergebnisse der funktionalen mit den zeitkritischen Simulationen miteinander verglichen, wird die Funktionalität gegenseitig bekräftigt. Es zeigt sich bei der Timing-Simulation, dass beim Daten-Schreiben und -Lesen ein zeitkritisches Problem entstehen kann (siehe Bild 5.8). Um dies zu vermeiden, wird bei jedem Schreib- und Lesezugriff ein weiterer Zustand hinzugefügt, der die Daten einen Takt vorher anlegt, damit die Daten sicher an ihr Ziel gelangen.

5.1.3 Vergleich

Es gibt minimale Abweichungen, wann die Daten anliegen, wie in dem Bild 5.8 zu erkennen ist. Ein weiterer großer Unterschied ist die benötigte Zeit für eine Simulation (siehe Tabelle 5.1). Aufgrund der deutlich längeren Zeit für die Timing Simulation wird sie nicht weiterverfolgt. Die vollbrachten Simulationen haben die funktionalen Aufgaben bekräftigt.

Methode	Zeit
Standard	ca. 1 sec pro 1ms
Timing	6,5h pro 1ms

Tabelle 5.1: *Benötigte Simulationszeit in Abhängigkeit von der Methode*

Der zusätzliche Zeitaufwand für die Timing-Simulation ist über das 23000-fache größer gegenüber der funktionalen Simulation, wie aus der Tabelle 5.1 erkennbar

ist. Für eine Simulation mit 45*ms* werden somit rund 12 Tage in Anspruch genommen. Für die große und auch interessante Timing-Simulation mit der Übertragung aller drei Eingangssignale, die über 220*ms* beträgt, würde rund 60 Tage dauern.

Aufgrund dieser langen Simulationszeiten würde ein schnellerer und leistungsstärkerer Rechner benötigt werden. Da dies nicht möglich ist, wurde entschieden, diese Timing-Simulationen nicht weiter zu verfolgen.

5.2 Hardware-Funktionen

In diesem Abschnitt wird näher auf das Senden und Empfangen zwischen dem FPGA und Matlab eingegangen sowie auf den Datenvergleich zwischen der Hardware, der empfangenen Daten in Matlab und dem Abhorchprogramm PortMon.

5.2.1 Senden und Empfangen zwischen FPGA und Matlab

- Das Bild 5.9 stellt das Senden von Befehlen von Matlab an das FPGA und das Zurücksenden der geforderten Daten vom FPGA zu Matlab dar. Das blaue Signal *RxD* ist das Senden von Matlab aus und das gelbe *TxD* das Senden vom FPGA aus. Hierbei handelt es sich um den Befehl $0xF$, d.h. es sollen neue Werte der drei Eingangssignale des ADCs aufgenommen und an Matlab gesendet werden.

- In Bild 5.10 wird der Befehl $0xD00A$ gesendet, d.h. es soll die Referenzspannung *ADCrefOut* vom ADC ausgelesen und zurückgesendet werden.

5.2.2 Komparatorsignal

- Bei nichtidealem Eingangssignal, wie bereits im Abschnitt 2.1.5 beschrieben ist, kommt es zu zahlreichen Pulsen sowohl bei fallender als auch steigender Flanke des Komparatorsignals. Dies ist bereits im Abschnitt 2.1.5 in Bild 2.7 dargestellt, wie das Komparatorsignal *sigCompUdiff1* im ungünstigen Fall aussieht. Oben ist das Komparatorsignal als ganzes zu sehen, unten ist das Komparatorsignal bei fallender Flanke herangezoomt. Dieser Zoom verdeutlicht das angesprochene Problem.

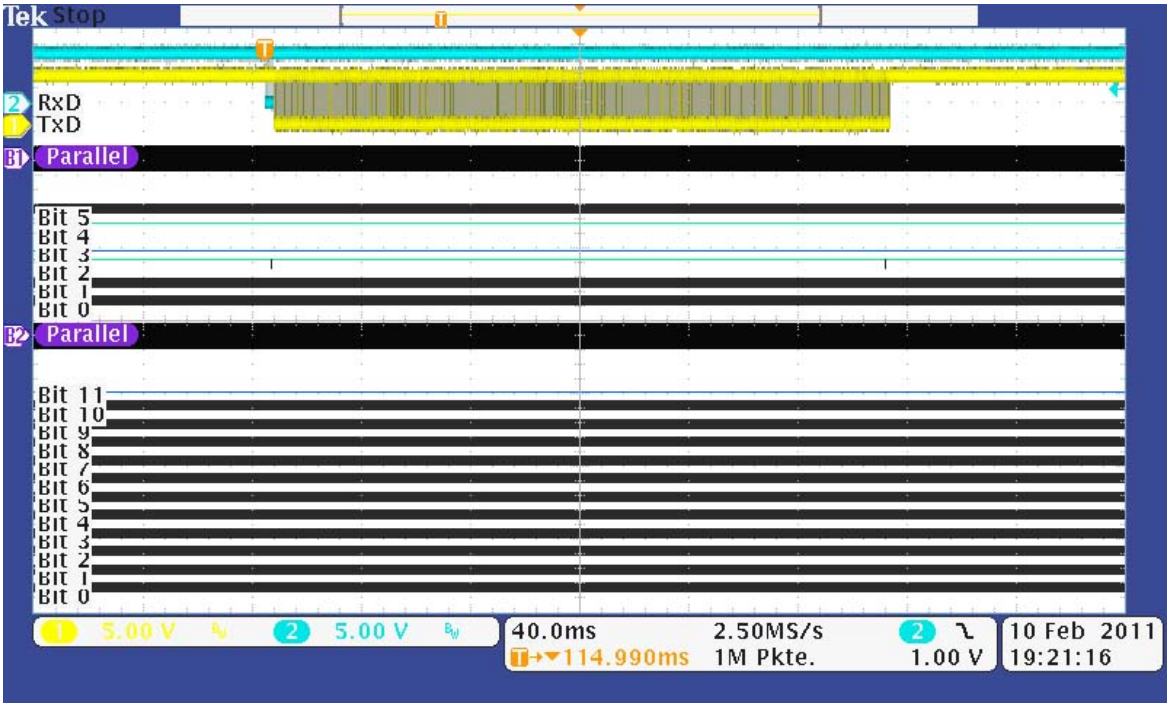


Bild 5.9: *Senden von Matlab zu FPGA (blau) und Senden von FPGA zu Matlab (gelb)*

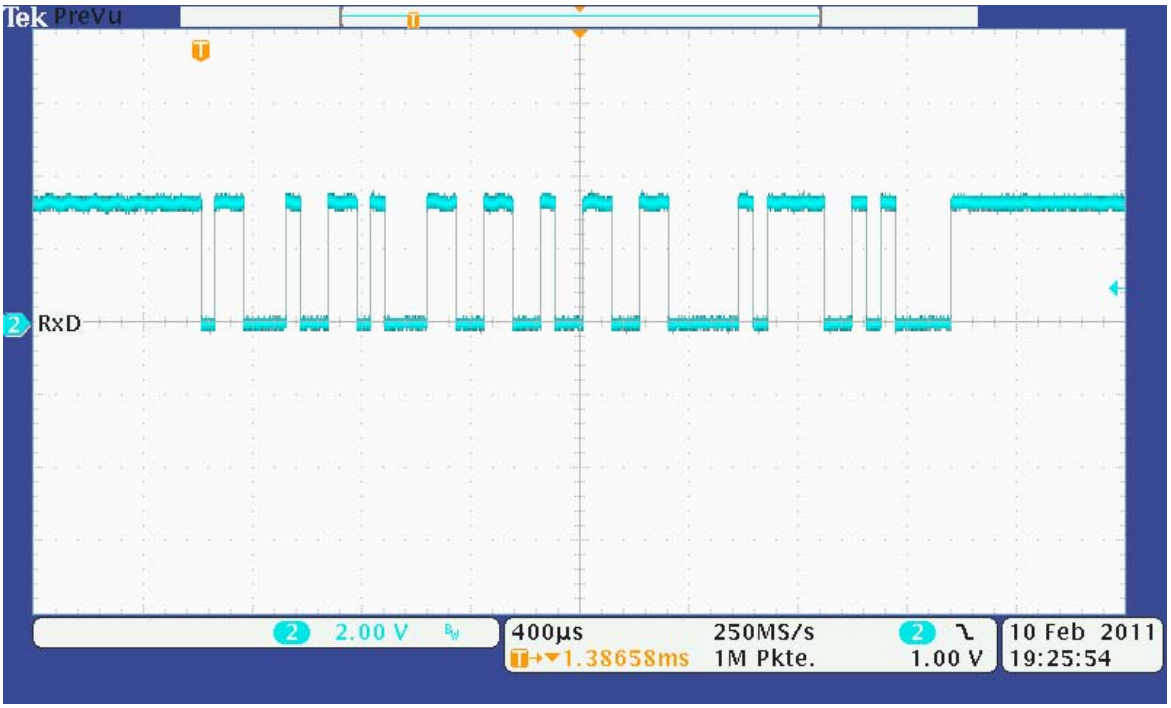


Bild 5.10: *Senden des Befehles „D00A“ von Matlab an das FPGA*

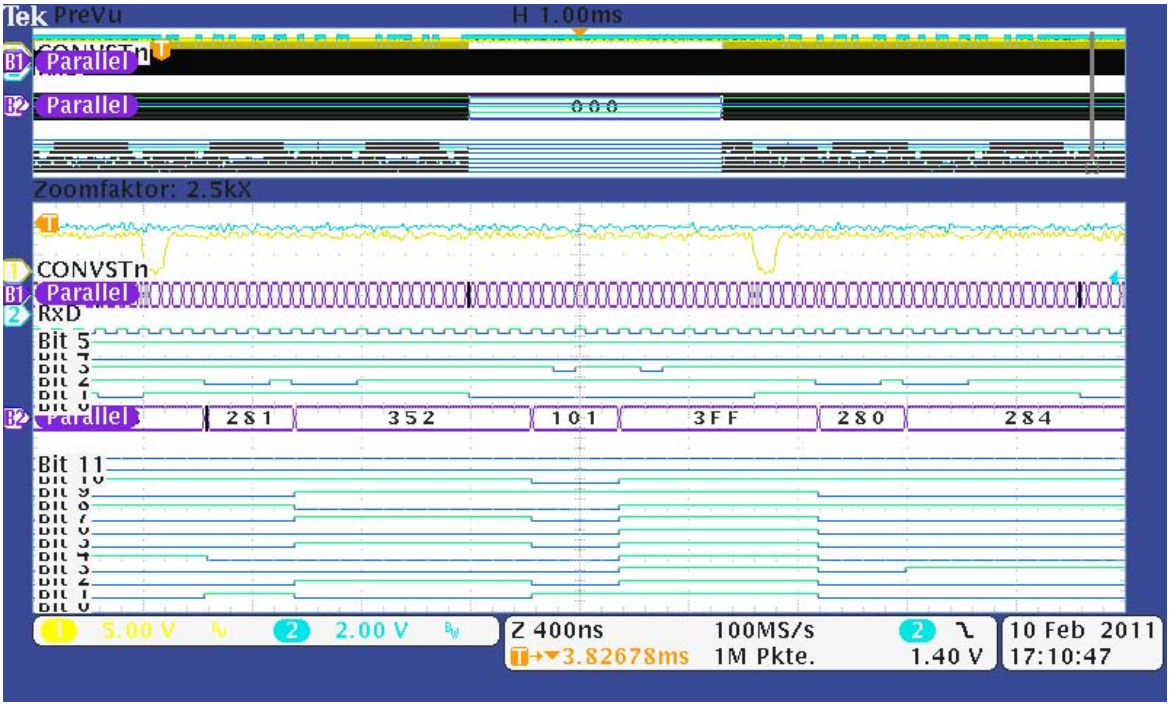


Bild 5.11: *Programmieren des DAC-Registers des ADCs*

5.2.3 Referenzspannung des ADCs programmieren

- In Bild 5.11 wird gezeigt, wie die Referenzspannung des ADCs programmiert wird. Das Schreiben des Wertes $0x101$ in das Config-Register bedeutet, dass der nächste Wert in das DAC-Register des ADCs geschrieben wird. Es wird hier der Wert $0x3FF$ geschrieben, welches einen Spannungswert von $2,5V$ entspricht. Der Bus „B1“ gibt den Datenbus an und der Bus „B2“ vom MSB nach LSB die Signale *CLK*, *ADC_RDn*, *ADC_WRn* und *BUSY*. Das gelb dargestellte Signal entspricht dem digitalen Signal *CONVSTn*, der die Konvertierung neuer Werte startet. Hier ist weiterhin zu erkennen, dass die zu schreibenden Daten einen Takt vor dem Schreiben anliegen.

5.2.4 Auswertung des Sequence-Registers

- In Bild 5.12 wird die Abfrage des Sequence-Registers abgefragt. Diese Abfrage erfolgt solange, bis die letzten beiden Bits die Folge „01“ aufweisen. Trifft die Bitfolge von „01“ zu, wird noch ein ganzer Zyklus gewartet, bevor neue Werte aus dem ADC-Register ausgelesen werden (siehe Bild 5.13).

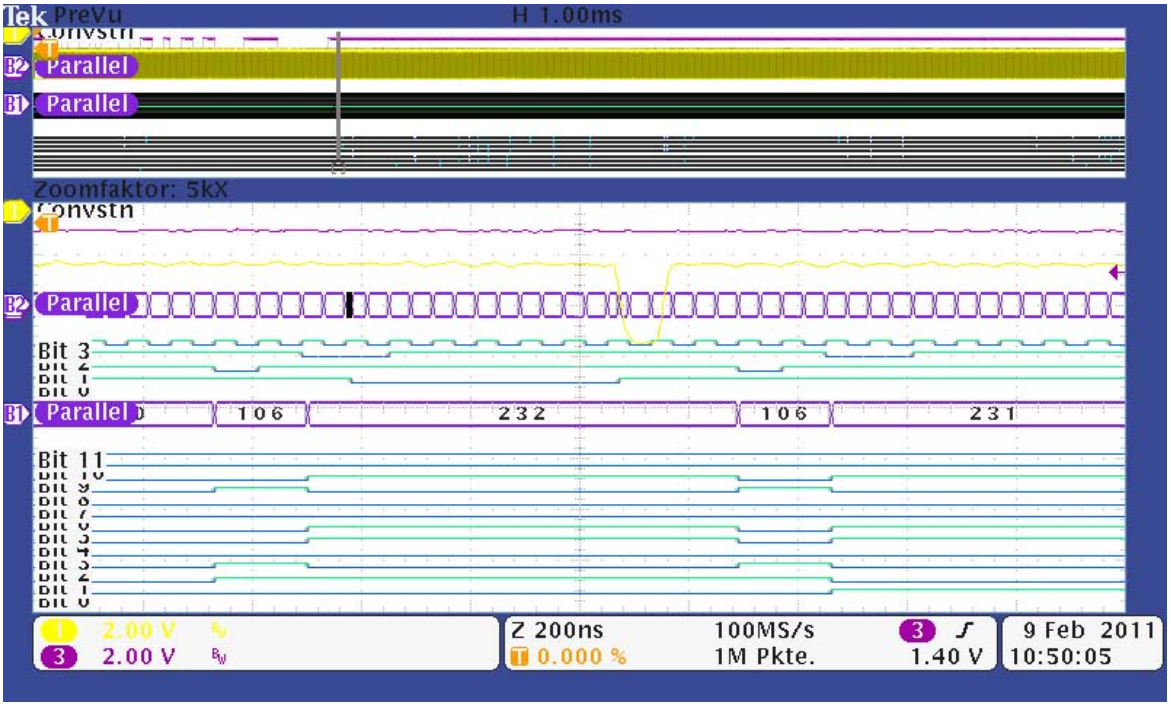


Bild 5.12: *Überprüfung der beiden letzten Bits des Sequence-Registers*

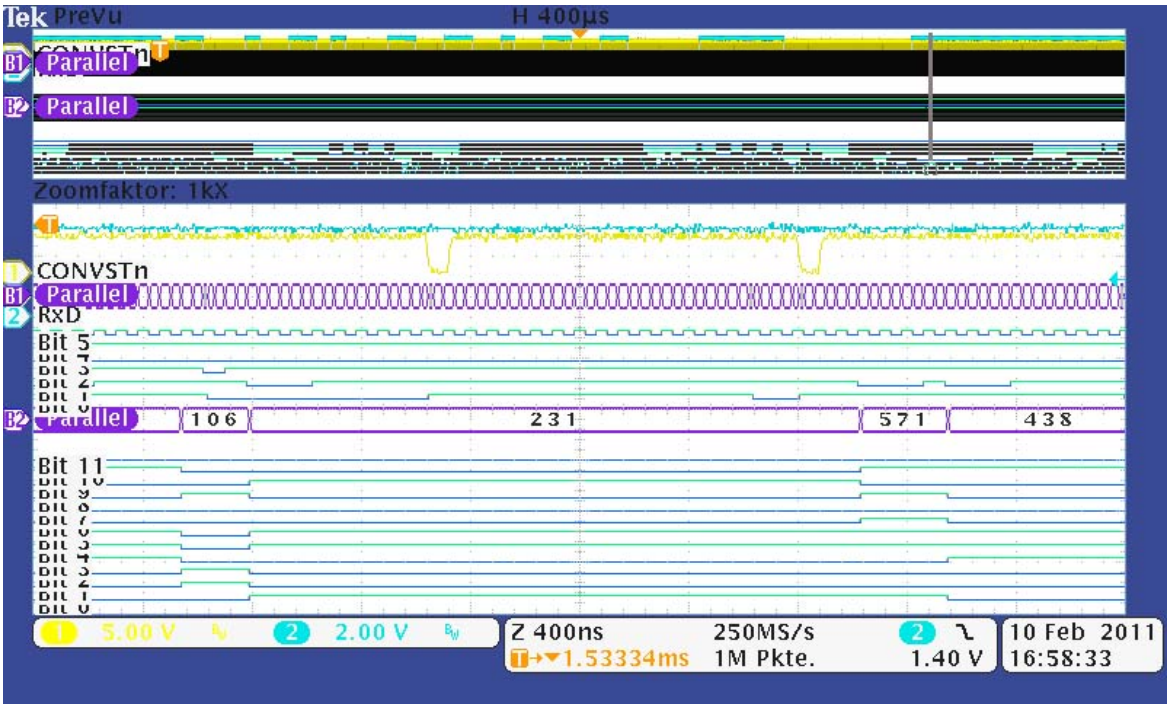


Bild 5.13: *Warten eines ganzen Zyklus*

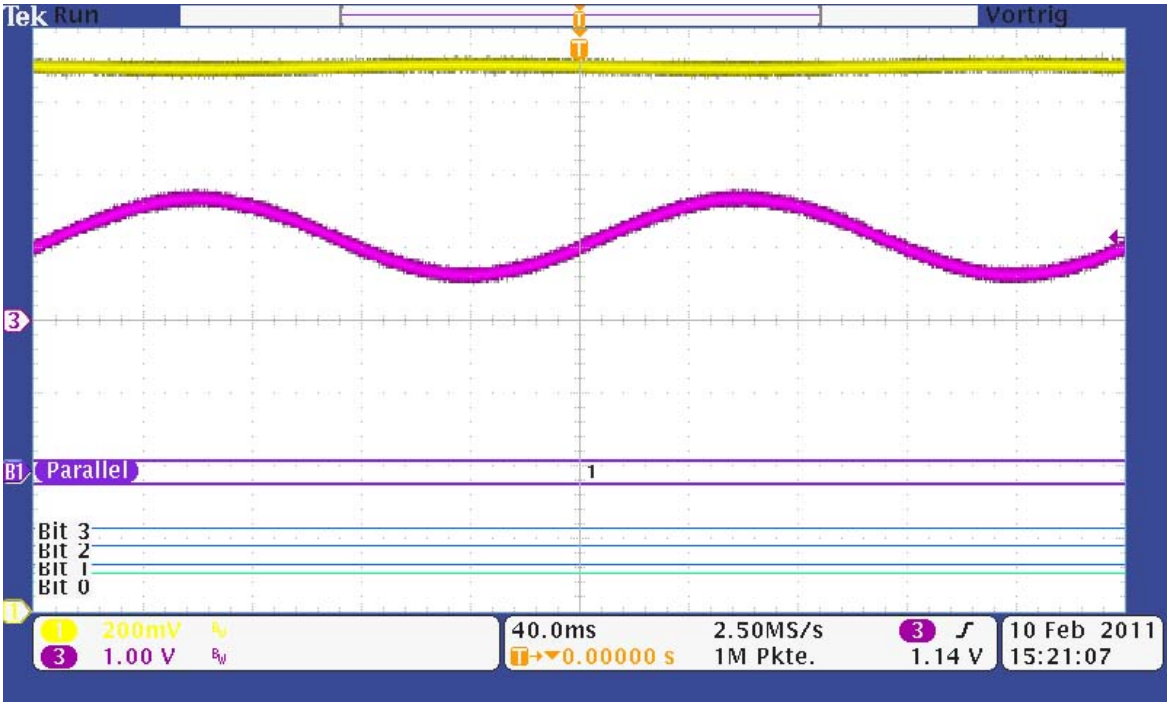


Bild 5.14: *Verstärkerstufe 1 bei einer Eingangsspannung von 10mV und einer Frequenz von 5Hz*

5.2.5 Verstärkungsregelung

- Als nächstes wird die Verstärkungsregelung betrachtet. Dazu ist das Verstärkermodul angeschlossen. Die Verstärkungsbits werden dem Verstärkermodul direkt übergeben. In Bild 5.14 ist das Eingangssignal (gelb) mit $5Hz$ und einer Eingangsspannung von $10mV$ versehen. Anhand des Busses ist zu sehen, dass die Verstärkung auf „001“ gesetzt wird. Dies entspricht der Verstärkungsstufe 1, also eine Verstärkung von 25. Wird die Eingangsspannung des Signals auf $20mV$ erhöht, wird die Verstärkungsstufe auf 0 zurückgesetzt, welches eine Verstärkung von 1 entspricht (siehe Bild 5.15. Das Differenzsignal ist lila dargestellt. Der Bus *B1* gibt die Verstärkungsstufe an.

- Ab einer Eingangsspannung von $50mV$ wird das Signal bereits übersteuert (siehe Bild 5.16. Diese Übersteuerung wird ebenfalls an Matlab gesendet und ist dort ebenfalls zu erkennen (siehe dazu Abschnitt 5.3.2).

5.2.6 Register auslesen

- In Bild 5.17 werden alle Register ausgelesen. Mit dem Wert $0x106$ wird das Sequence-Register hier mit dem Rückgabewert $0x231$ ausgelesen und mit

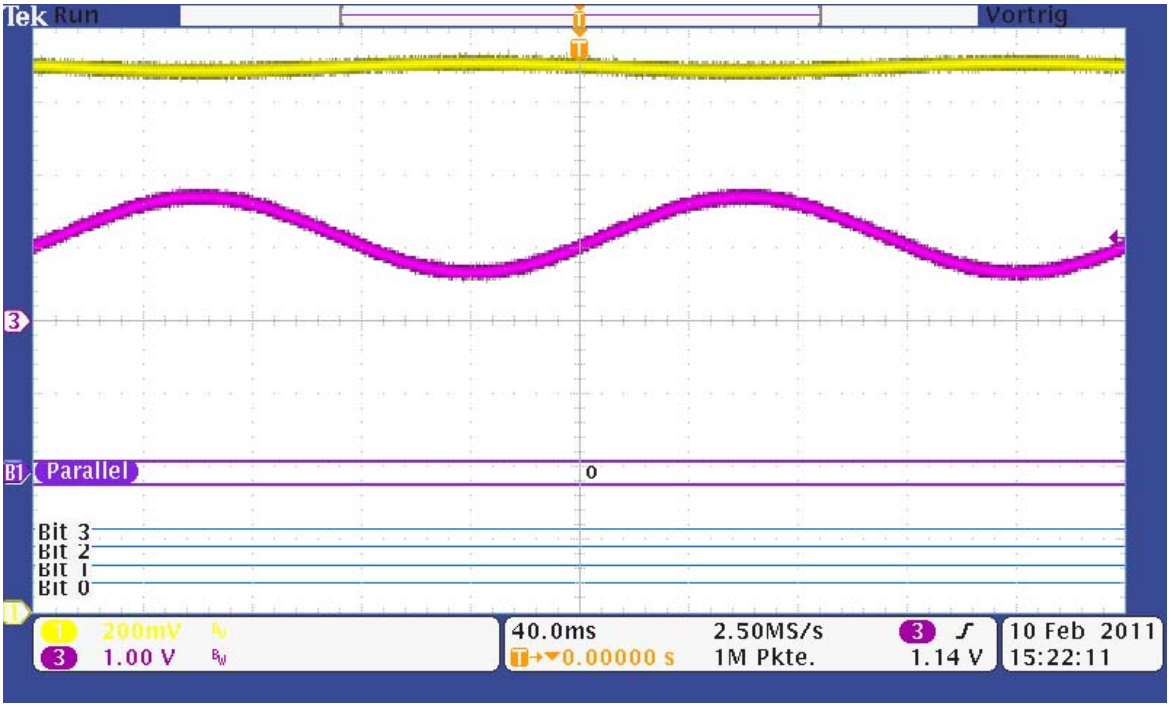


Bild 5.15: *Verstärkerstufe 0 bei einer Eingangsspannung von 20mV und einer Frequenz von 5Hz*

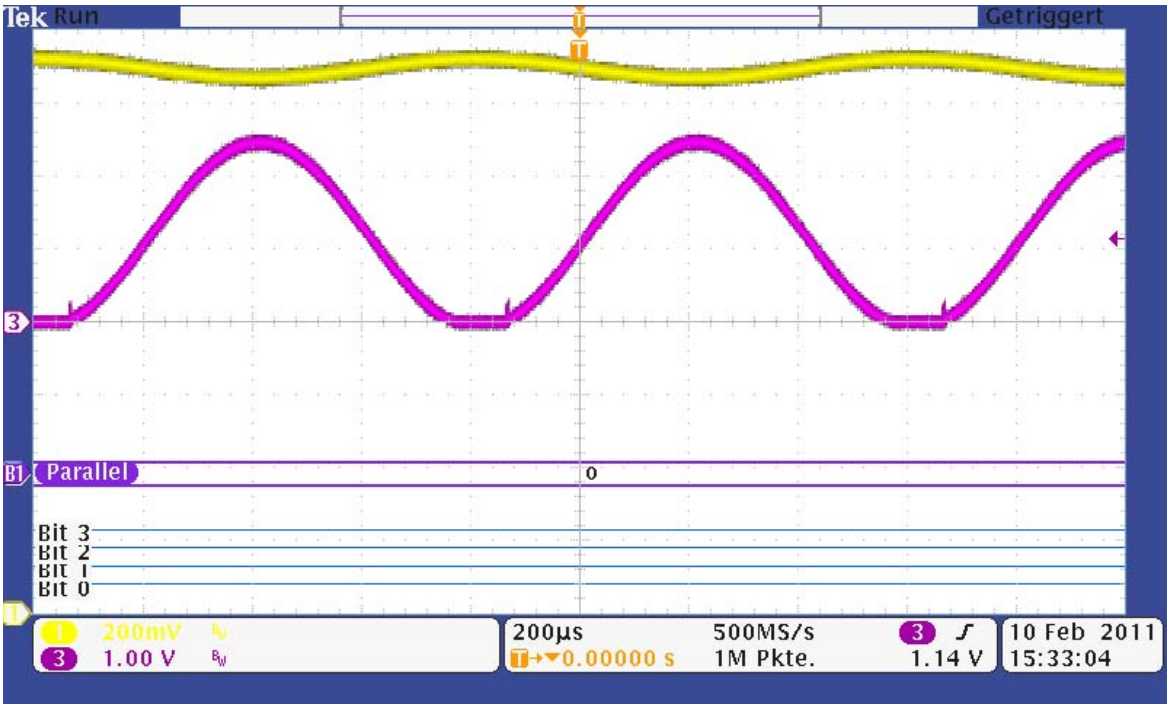


Bild 5.16: *Verstärkerstufe 0 bei einer Eingangsspannung von 50mV und einer Frequenz von* $1,25kHz$

Bild 5.17: *Es werden alle Register ausgelesen sowohl vom ADC als auch vom DAC*

$0x103$ die Referenzspannung mit dem Rückgabewert $0x3FF$. Anschließend werden die acht DAC Ausgangsregister ausgelesen ($0x390$, $0x410$, 3x $0x000$, $0x400$ und $0x000$). Als letztes wird die Verstärkung übertragen, die hier ebenfalls den Wert $0x000$ beträgt.

5.3 Funktions- und Datenvergleich

In diesem Abschnitt werden die Funktionen von den Simulationen und der Hardware sowie die Daten miteinander verglichen.

5.3.1 Funktionsvergleich Simulation - Hardware

Werden die Simulationen mit den Hardware-Funktionen verglichen (siehe Abschnitte 5.1 und 5.2), ist die Übereinstimmung festzuhalten. Die Aufgaben in der Simulation sowohl in der funktionalen als auch in der Timingsimulation und der anschließenden Aufnahmen mit dem Logicanalyzer (Speicheroszilloskop) bestätigen die Funktionalität sowohl in Ablauf als auch in der Datensicherheit. Unter der Datensicherheit ist das sichere Anliegen der Daten zu verstehen.

5.3.2 Datenvergleich Hardware - Software

Über die 7-Segment-Anzeige sind die aktuellen Werte der Eingangskanäle des ADCs, das Differenzsignal *UDIFF* und die Halbbrückensignale *HB1* und *HB2*, sowie die ADC-Registerwerte Sequence *ADCsequence* und die Referenzspannung *ADCrefOut*, die Komparatorschwellen *DIFF1* und *DIFF2* sowie der mit den Tasten einstellbare Anzeigewert der drei aufgenommenen Eingangssignale *ArrayValue* darstellbar.

Es werden zwei Tabellen aufgezeigt, die den Datenvergleich zwischen der Hardware, Matlab und PortMon liefern (siehe Tabellen 5.2 und 5.3).

	PortMon			Matlab			7-Segment-Anzeige		
	UDIFF	HB1	HB2	UDIFF	HB1	HB2	UDIFF	HB1	HB2
1	04B1	03A1	0366	4B1	3A1	366	4B1	3A1	366
2	04D7	03A8	034D	4D7	3A8	34D	4D7	3A8	34D
3	04E9	0383	0317	4E9	383	317	4E9	383	317
4	050A	0397	0326	50A	397	326	50A	397	326
5	051C	03C7	0343	51C	3C7	343	51C	3C7	343
6	053E	041B	0396	53E	41B	396	53E	41B	396
7	054C	0420	0394	54C	420	394	54C	420	394
8	054E	03EF	0353	54E	3EF	353	54E	3EF	353
9	0563	03AA	0315	563	3AA	315	563	3AA	315
10	056E	03A7	02FA	56E	3A7	2FA	56E	3A7	2FA
11	056F	0422	0380	56F	422	380	56F	422	380
12	056E	03D8	0339	56E	3D8	339	56E	3D8	339
13	055C	03A1	02FA	55C	3A1	2FA	55C	3A1	2FA
14	0561	03D1	0341	561	3D1	341	561	3D1	341
15	0548	0419	038B	548	419	38B	548	419	38B
16	0540	0401	037F	540	401	37F	540	401	37F
17	0526	03C4	0342	526	3C4	342	526	3C4	342
18	0516	03A5	0332	516	3A5	332	516	3A5	332
19	04F9	038F	0323	4F9	38F	323	4F9	38F	323
20	04D6	03C2	035C	4D6	3C2	35C	4D6	3C2	35C
21	04BC	03AB	0369	4BC	3AB	369	4BC	3AB	369
22	0498	03C4	0388	498	3C4	388	498	3C4	388
23	0466	0357	0332	466	357	332	466	357	332
24	0440	0367	034E	440	367	34E	440	367	34E
25	0415	03AD	03B7	415	3AD	3B7	415	3AD	3B7
26	03FF	034D	0365	3FF	34D	365	3FF	34D	365
27	03C9	034C	0374	3C9	34C	374	3C9	34C	374
28	039A	0330	036E	39A	330	36E	39A	330	36E

29	0371	0324	0371	371	324	371	371	324	371
30	0356	0375	03D0	356	375	3D0	356	375	3D0
31	0324	0341	03B7	324	341	3B7	324	341	3B7
32	02FD	033A	03C9	2FD	33A	3C9	2FD	33A	3C9
33	02DA	02FC	0395	2DA	2FC	395	2DA	2FC	395
34	02BC	02E2	0398	2BC	2E2	398	2BC	2E2	398
35	0291	033E	040A	291	33E	40A	291	33E	40A
36	027C	0340	040D	27C	340	40D	27C	340	40D
37	0261	035D	0442	261	35D	442	261	35D	442
38	0255	02CF	03AE	255	2CF	3AE	255	2CF	3AE
39	0244	02E8	03CF	244	2E8	3CF	244	2E8	3CF
40	0232	0326	0414	232	326	414	232	326	414
41	0221	02F7	03F2	221	2F7	3F2	221	2F7	3F2
42	0222	031C	0428	222	31C	428	222	31C	428
43	0220	02BA	03B9	220	2BA	3B9	220	2BA	3B9
44	0223	02DF	03D9	223	2DF	3D9	223	2DF	3D9
45	0217	0315	041D	217	315	41D	217	315	41D
46	0229	0352	044E	229	352	44E	229	352	44E
47	023F	02C9	03C0	23F	2C9	3C0	23F	2C9	3C0
48	024F	02FA	03ED	24F	2FA	3ED	24F	2FA	3ED
49	0266	02E7	03C1	266	2E7	3C1	266	2E7	3C1
50	0277	0300	03CE	277	300	3CE	277	300	3CE
51	0293	02FB	03BE	293	2FB	3BE	293	2FB	3BE
52	02B6	032C	03EA	2B6	32C	3EA	2B6	32C	3EA
53	02CE	031D	03C9	2CE	31D	3C9	2CE	31D	3C9
54	02F6	032B	03C6	2F6	32B	3C6	2F6	32B	3C6
55	031B	0362	03DC	31B	362	3DC	31B	362	3DC
56	0349	036F	03E8	349	36F	3E8	349	36F	3E8
57	0369	0394	03EA	369	394	3EA	369	394	3EA
58	0395	0358	03A1	395	358	3A1	395	358	3A1
59	03C0	03B0	03E8	3C0	3B0	3E8	3C0	3B0	3E8
60	03E9	0344	0362	3E9	344	362	3E9	344	362
61	0409	0383	0393	409	383	393	409	383	393
62	043A	0360	0358	43A	360	358	43A	360	358
63	045D	03AC	038B	45D	3AC	38B	45D	3AC	38B
64	0488	0364	0331	488	364	331	488	364	331

Tabelle 5.2: *Vergleichstabelle für Hardware-Portmon-Matlab*

Werden mit Matlab alle Register ausgelesen und diese Daten mit den von PortMon aufgezeichneten Datenverkehr und den von der 7-Segment-Anzeige dargestellten

Werten verglichen, ist eine Übereinstimmung aller Daten festzuhalten (siehe Tabelle 5.3).

Daten	PortMon	Matlab	7-Segment-Anzeige
ADCsequence	0232	232	232
ADCrefOut	03FF	3FF	3FF
ref_diff1	0390	390	390
ref_diff2	0410	410	410

Tabelle 5.3: *Datenvergleich zwischen PortMon, Matlab und der 7-Segment-Anzeige*

In der Tabelle 5.4 wird die Kommunikation zwischen dem FPGA und Matlab mittels PortMon dargestellt. Von Matlab aus wird ein Befehl an das FPGA gesendet. Sie wird im FPGA ausgewertet und umgesetzt. Sind Daten von Matlab gefordert, werden diese an Matlab gesendet. Die Spalte *Prozess* gibt Auskunft darüber, ob Matlab sendet und das FPGA empfängt (*IRP_MJ_WRITE*) oder Matlab empfängt und das FPGA sendet (*IRP_MJ_READ*).

Die ersten beiden Zeilen geben den Befehl, alle Register vom ADC und DAC auszulesen. Diese werden in Zeile 3 zurückgesendet. Werden die Daten in Matlab ausgegeben, sind die gerade empfangenen Daten zu sehen. Die Frequenz wird nur mittels des Befehls von Zeile 5 übergeben, d.h. neue Werte der drei Eingangskanäle übertragen. Der Wert für die Frequenz ist in Zeile 20 in den letzten vier Bytes wiederzuerkennen. Die ersten drei von diesen vier letzten gibt die Zeit in Anzahl von den benötigten Takten für eine 1/64 Schwinung an. Sie wird in Matlab in Frequenz umgerechnet. Es ergibt sich für die Berechnung der Frequenz die Gleichung 5.1:

$$f = \frac{1}{Scale} = \frac{1}{timeScale \cdot 64 \cdot 80ns} \tag{5.1}$$

Scale entspricht der Zeit für eine komplette Schwingung. *timeScale* ist die vergangene Zeit für einen neuen Wert. Die 64 steht für die Anzahl von aufgenommenen Werten und zum Schluss der Takt, mit dem gearbeitet wird. Die Berechnung der Frequenz hat eine Abweichung von ca. 1% von der tatsächlich eingstellten Eingangsfrequenz. Dies hängt damit zusammen, dass die tatsächliche Zeit für eine komplette Periode durch 64 geteilt wird und das Ergebnis an Matlab übertragen wird, um Sendebytes zu sparen. Um die Frequenz genauer zu bestimmen, können mehr Sendebytes für eine komplette Peridoe spendiert oder ein höherer Takt für das VHDL-Modul für die Bestimmung der Periodendauer verwendet werden.

Nr.	Prozess	PortMon-Daten	Matlab-Daten
1	IRP_MJ_WRITE	Length 3: 23 31 32	

2	IRP_MJ_WRITE	Length 2: C0 00	
3	IRP_MJ_READ	Length 27: 23 32 32 32 03 80 04 50 00 00 00 00 00 00 00 00 00 00 00 00 00 03 FF 02 31 0A	Received Bytes: 26 ADCrefOut: 3FF ADCsequence: 231 ref_diff1: 380 ref_diff2: 450 ref_hb1: 0 ref_hb2: 0 u_broff1: 0 u_broff2: 0 u_off: 0 u_opt: 0 gain: 0 frequence: 65535 Hz
4	IRP_MJ_WRITE	Length 3: 23 31 32	
5	IRP_MJ_WRITE	Length 2: F0 00	
6	IRP_MJ_READ	Length 15: 23 33 33 38 37 04 49 0F FF 00 00 04 49 0F FF	
7	IRP_MJ_READ	Length 26: 00 00 04 49 0F FF 00 00 04 49 0F FF 00 00 04 49 0F FF 00 00 04 49 0F FF 00 00	
8	IRP_MJ_READ	Length 30: 04 49 0F FF 00 00 04 49 0F FF 00 00 04 49 0F FF 00 00 04 49 0F FF 00 00 04 49 0F FF 00 00	
9	IRP_MJ_READ	Length 26: 04 49 0F FF 00 00 04 49 0F FF 00 00 04 0A 0F FF 00 00 04 0A 00 00 00 00 04 0A	
10	IRP_MJ_READ	Length 25: 00 00 00 00 04 0A 00 00 00 00 04 0A 00 00 00 00 04 0A 00 00 00 00 04 0A 00	
11	IRP_MJ_READ	Length 31: 00 00 00 04 0A 00 00 0F FF 05 34 0F FF 00 00 05 44 0F FF 00 00 05 44 00 00 00 00 05 44 00 00	
12	IRP_MJ_READ	Length 25: 00 00 05 44 00 00 00 00 05 44 00 00 00 00 05 44 00 00 00 00 05 44 00 00 00	

13	IRP_MJ_READ	Length 26: 00 05 44 00 00 00 00 05 44 00 00 00 00 05 24 00 00 00 00 05 24 0F FF 00 00 05	
14	IRP_MJ_READ	Length 31: 24 0F FF 00 00 05 24 0F FF 00 00 05 24 0F FF 00 00 05 24 0F FF 0F FF 05 24 0F FF 0F FF 05 24	
15	IRP_MJ_READ	Length 25: 0F FF 0F FF 05 36 0F FF 0F FF 05 36 0F FF 0F FF 05 36 0F FF 0F FF 05 36 0F	
16	IRP_MJ_READ	Length 26: FF 0F FF 05 36 0F FF 0F FF 05 36 0F FF 00 00 05 36 0F FF 00 00 05 36 0F FF 00	
17	IRP_MJ_READ	Length 30: 00 05 36 0F FF 00 00 05 3D 0F FF 00 00 05 3D 00 00 00 00 05 3D 00 00 00 00 05 3D 00 00 00	
18	IRP_MJ_READ	Length 26: 00 05 3D 00 00 00 00 05 3D 00 00 00 00 05 3D 00 00 00 00 05 3D 00 00 00 00 05	
19	IRP_MJ_READ	Length 25: 32 00 00 00 00 05 32 00 00 00 00 05 32 00 00 00 00 05 32 00 00 00 00 05 32	
20	IRP_MJ_READ	Length 26: 00 00 00 00 05 32 00 00 0F FF 05 32 00 00 0F FF 05 32 00 00 0F FF 00 00 0B 0A	Received Bytes: 393 display read data
21	IRP_MJ_WRITE	Length 3: 23 31 32	
22	IRP_MJ_WRITE	Length 2: D0 0A	
23	IRP_MJ_WRITE	Length 3: 23 31 32	
24	IRP_MJ_WRITE	Length 2: A2 5F	
25	IRP_MJ_READ	Length 6: 23 31 32 03 FF 0A	Received Bytes: 6
26	IRP_MJ_WRITE	Length 3: 23 31 32	
27	IRP_MJ_WRITE	Length 2: D0 0A	
28	IRP_MJ_READ	Length 6: 23 31 32 02 5F 0A	Received Bytes: 6 ADCrefOut: 25F ADCsequence: 231 ref_diff1: 380 ref_diff2: 450 ref_hb1: 0 ref_hb2: 0

			u_broff1: 0 u_broff2: 0 u_off: 0 u_opt: 0 gain: 0 frequence: 1.776e+004 Hz
29	IRP_MJ_WRITE	Length 3: 23 31 32	
30	IRP_MJ_WRITE	Length 2: D0 0A	
31	IRP_MJ_WRITE	Length 3: 23 31 32	
32	IRP_MJ_WRITE	Length 2: A3 FF	
33	IRP_MJ_READ	Length 6: 23 31 32 02 5F 0A	Received Bytes: 6
34	IRP_MJ_WRITE	Length 3: 23 31 32	
35	IRP_MJ_WRITE	Length 2: C0 00	
36	IRP_MJ_READ	Length 27: 23 32 32 32 03 80 04 50 00 00 00 00 00 00 00 00 00 00 00 00 07 03 FF 02 32 0A	Received Bytes: 27 ADCrefOut: 3FF ADCsequence: 232 ref_diff1: 380 ref_diff2: 450 ref_hb1: 0 ref_hb2: 0 u_broff1: 0 u_broff2: 0 u_off: 0 u_opt: 0 gain: 7 frequence: 1.776e+004 Hz

Tabelle 5.4: *Portmon-Matlab-Vergleich*

Anhand der Tabelle 5.4 ist festzustellen, dass die gesendeten und empfangenen Daten übereinstimmen. Die Anzahl an Daten ist ebenfalls richtig. Nach jedem Sendeabschnitt ist ein *LF* (engl. linefeed, 0x0A) zu sehen, der die Übertragung für Abgeschlossen erklärt.

Es muss auf ein ganz bestimmtes Detail geachtet werden. In Matlab muss nach jedem Schreibprozess *set()* ein Leseprozess *get()* hinzugefügt werden, da der Treiber automatisch für einen Leseprozess sorgt. Aus diesem Grund muss die Funktion *invoke()* zusätzlich ausgeführt werden, um die empfangen Daten zu den entsprechenden Variablen zuordnen zu können.

Beim Vergleich der Daten anhand der beiden Tabellen 5.2, 5.3 und 5.4 ist der identische Inhalt zu erkennen. Hiermit ist die korrekte Funktionalität und Arbeitsweise

des Sendens von Matlab zum FPGA und umgekehrt sowie die gegenseitige Auswertung der Daten und dessen Umsetzung bewiesen.

6 Gesamtfunktion

In diesem Kapitel werden die verwendeten Einstellungen angesprochen und der gesamte Funktionsablauf dargestellt.

6.1 Verwendete Einstellungen

In diesem Abschnitt wird auf die verwendeten Einstellungen eingegangen.

6.1.1 Analog-Digital-Converter

Für die Referenzspannung des ADCs wird eine Spannung von $2,5V$ eingesetzt, die mit dem Wert $0x3FF$ programmiert wird. Diese Spannung bzw. dieses Signal wird ebenfalls als Eingangs- und Ausgangsspannung der beiden Referenzen genutzt.

Für den ADC kommt die Funktion *Sequence* zum Einsatz. Es sollen dabei drei Eingangskanäle mit ihren entsprechenden positiven und negativen Referenzen eingestellt werden. Die konvertierten Werte werden abwechselnd nach jedem Zyklus aus dem entsprechenden Register ausgelesen. In das Sequence-Register wird der Wert $0x230$ geschrieben. Dadurch stehen nach dem ersten Zyklus der erste und dritte Kanal bereit und nach dem zweiten Zyklus der vierte Kanal. Der zweite Kanal wird zwar ausgelesen, aber weder gespeichert noch weiterverarbeitet (siehe Abschnitt 2.2.1.1).

An den ersten beiden Eingangskanälen des ADCs ist das Signal *UDIFF* angeschlossen. An dem dritten Eingangskanal liegt das Signal *HB1* an und an dem vierten *HB2*. Die Referenzspannungen sind hier auf Masse gezogen.

Es sind folgende Eingangssignale an dem Vorverstärker-Modul angelegt (siehe Tabelle 6.1):

Signal	Frequenz	Amplitude V_{PP}	Offset	Phase
Halbbrücke1	5Hz	$10mV$	$1,5V$	$0°$
Halbbrücke2	5Hz	$10mV$	$1,5V$	$180°$

Tabelle 6.1: *Verwendete Einstellungen für die Eingangssignale*

6.1.2 Digital-Analog-Converter

Die Referenzspannungen für die Komparatoren werden für die Schwellenspannung *DIFF1* mit dem HEX-Wert $0x390$ ($912 \rightarrow 1,11V$) belegt und für die Schwellenspannung *DIFF2* mit dem Wert $0x410$ ($1040 \rightarrow 1,27V$). Die beiden anderen Schwellenspannungen werden zur Zeit nicht eingesetzt, sodass diese unberührt auf $0x000$ bleiben.

Für die Offsetkompensation des Vorverstärker-Moduls wird eine Spannung von $1,5V$ eingestellt. Sie entspricht dem hexadecimalen Wert $0x400$.

Die drei übrigen Ausgangsspannungen werden für diesen Betrieb nicht benötigt, sodass diese ebenfalls unberührt mit den Werten $0x000$ versehen bleiben.

6.1.3 Sonstige Einstellungen

Der Verstärkungsfaktor *AMPgain* beginnt mit der maximalen Verstärkung von '7' und wird entsprechend automatisch heruntergeregelt, sobald dies nötig ist.

Die benötigte Zeit für eine Schwingung wird ebenfalls auf das Maximum festgelegt und nach der ersten Berechnung geändert.

6.2 Funktionsablauf

Nachdem alle Einstellungen für diesen Betrieb gesetzt sind, werden in diesem Abschnitt der gesamte Ablauf eines Befehls von Matlab aus über das Empfangen der geforderten Daten bis zur Auswertung und Darstellung der Daten in Matlab gezeigt. Es werden viele Verweise auf Grafiken und Tabellen des vorherigen Kapitels 5 bezogen.

6.2.1 Vom Befehl zum Ergebnis

Das gelbe Signal in Bild 5.14 entspricht den Einstellungen aus Tabelle 6.1. Es sind in dem Bild nur das Halbbrückensignal 1 dargestellt und das dazugehörige bereits bestimmte Differenzsignal (lila dargestellt) des Vorverstärker-Moduls.

Die drei Eingangssignale sollen vom ADC erfasst und in Matlab verarbeitet und grafisch dargestellt werden. Das Ergebnis wird in Bild 6.1 gezeigt. Es werden die drei Eingangssignale gezeigt: 1. Udiff, 2. HB1 und 3. HB2. Die beiden Signale HB1 und HB2 weisen eine gezackte Schwingung auf. Dies hängt damit zusammen, dass die Amplituden sehr klein sind. Je größer die Amplituden, desto besser ist das Signal. Diese Signale sind weiterhin bereits um den Faktor 25 (Verstärkerstufe 1) im Vorverstärker-Modul verstärkt. Daher fallen die Sprünge stärker aus.

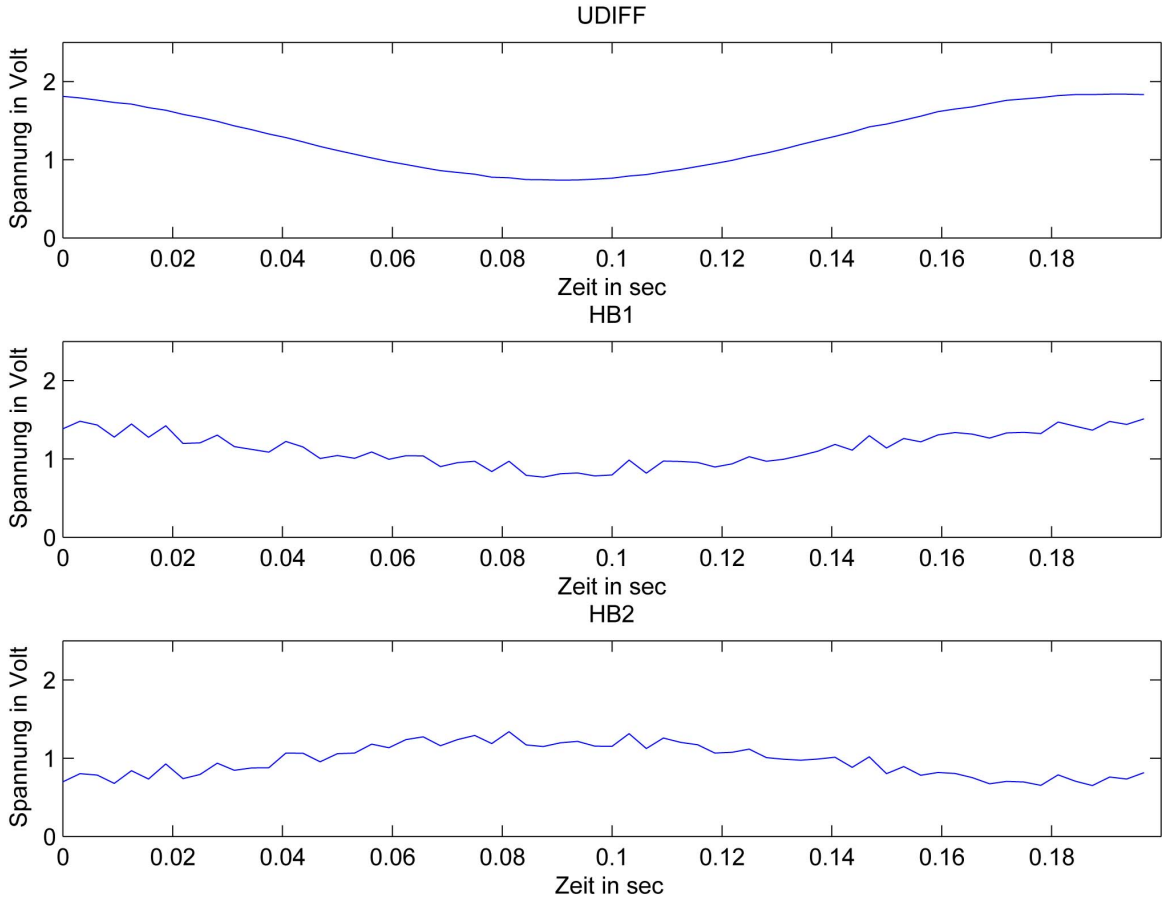


Bild 6.1: *Darstellung der vom FPGA gesendeten Daten der drei Eingangssignale*

Werden anschließend alle Register des ADCs und DACs ausgelesen und dargestellt, kommt folgendes Ergebnis heraus (siehe Tabelle 6.2).

ADCrefOut:	3FF
ADCsequence:	232
ref_diff1:	390
ref_diff2:	410
ref_hb1:	0
ref_hb2:	0
u_broff1:	0
u_broff2:	0
u_off:	400
u_opt:	0
gain:	1
frequence:	4.987e+000 Hz

Tabelle 6.2: *Registerausgabe in Matlab bei Verstärkerstufe 1 und 5Hz Eingangsfrequenz*

Wird das Bild 6.1 näher betrachtet, so fallen die Störungen auf. Sie sind nicht nur bei einem der drei Eingangssignale sondern bei allen erkennbar und zwar zu exakt gleichen Zeitpunkten und in der gleichen Richtung. Es wurde das Vorverstärker-Modul überprüft, welches richtig arbeitet. Es ist bewiesen, dass es ebenfalls nicht in dem Bereich zwischen dem Array im FPGA und Matlab liegt (siehe Tabellen 5.2 und 5.3). Somit muss sich das Problem zwischen den Eingangssignalen und dem Array im FPGA befinden. Es wird vermutet, dass es beim Abspeichern in das Array passiert. Die konvertierten Daten vom ADC werden in die entsprechenden Variablen gespeichert. Zum Speichern der Werte in die Arrays wird ein Zähler verwendet, der beim Erreichen des Signals *sampleIntervall* das Zeichen zum Werteabspeichern gibt. Die aktuellen Werte aus den Variablen werden den Registern im Array zugewiesen und dort gespeichert. Wenn die Werte in die Arrays geschrieben werden, wenn neue aktuelle Daten vom ADC kommen, liegen die Daten nicht sicher an. Um dieses Problem zu lösen, kann ein zusätzliches Array für jedes der drei Eingangssignale hinzugefügt werden. Dieses neue Array sorgt dafür, dass nur die Daten aus dem 1. bzw. 2. Feld des Arrays genommen werden dürfen, die sicher anliegen und nicht geändert werden.

6.2.2 Zwischenergebnisse

Um neue Werte aller drei Eingangssignale zu bekommen, wird der Befehl $0xF$ in Matlab an das FPGA gesendet. Dieser Befehl wird ausgewertet und dem Steuerpfad übergeben. Dieser sorgt dafür, dass die drei Arrays mit neuen Werten gefüllt werden. Nachdem sie gefüllt sind, werden diese an Matlab zurückgesendet (siehe Bild 5.9). Die gesendeten, empfangenen und ausgewerteten Daten stimmen übereinstimmen (vgl. Tabellen 5.2, 5.3 und 5.4), sodass die Anforderung, eine Platine zu

entwickeln, die analoge Signale in digitale konvertiert und an Matlab sendet, erfüllt ist.

Die übertragenen Daten der einzelnen Eingangssignale werden in drei Formate umgewandelt (siehe Tabelle 6.3; hier nur Udiff aufgelistet, im Anhang zusätzlich die Signale HB1 und HB2 siehe B.2): 1. Decimal, 2. Hexadecimal und 3. Spannung in Volt. Die Werte in Volt werden grafisch dargestellt (siehe Bild 6.1).

Nr.	Spannung [V]	decimal	hexadecimal
1	1,41235	1157	485
2	1,46606	1201	4B1
3	1,50635	1234	4D2
4	1,56738	1284	504
5	1,60645	1316	524
6	1,61621	1324	52C
7	1,62720	1333	535
8	1,67725	1374	55E
9	1,66992	1368	558
10	1,68457	1380	564
11	1,68945	1384	568
12	1,69678	1390	56E
13	1,71143	1402	57A
14	1,68091	1377	561
15	1,66992	1368	558
16	1,67969	1376	560
17	1,62842	1334	536
18	1,62231	1329	531
19	1,60767	1317	525
20	1,57227	1288	508
21	1,53076	1254	4E6
22	1,47705	1210	4BA
23	1,44287	1182	49E
24	1,40991	1155	483
25	1,33789	1096	448
26	1,30249	1067	42B
27	1,24146	1017	3F9
28	1,19751	981	3D5
29	1,15112	943	3AF
30	1,09497	897	381
31	1,03760	850	352
32	0,98022	803	323
33	0,93628	767	2FF

34	0,89844	736	2E0
35	0,84717	694	2B6
36	0,81909	671	29F
37	0,78247	641	281
38	0,76782	629	275
39	0,72876	597	255
40	0,70313	576	240
41	0,69336	568	238
42	0,66040	541	21D
43	0,67505	553	229
44	0,65674	538	21A
45	0,67139	550	226
46	0,66162	542	21E
47	0,67871	556	22C
48	0,68604	562	232
49	0,70313	576	240
50	0,73486	602	25A
51	0,77148	632	278
52	0,81421	667	29B
53	0,83862	687	2AF
54	0,87280	715	2CB
55	0,93018	762	2FA
56	0,94971	778	30A
57	1,01074	828	33C
58	1,06812	875	36B
59	1,11328	912	390
60	1,16089	951	3B7
61	1,21094	992	3E0
62	1,27563	1045	415
63	1,31470	1077	435
64	1,36475	1118	45E

Tabelle 6.3: *Die Umwandlung des Eingangssignals UDIFF in den drei Formaten: Spannung in Volt, Decimal, Hexadecimal*

7 Fazit, Bewertung & Ausblick

7.1 Fazit und Bewertung

In dieser Bachelorthesis wurde eine Platine entwickelt, die es ermöglicht, die drei analogen Signale des Vorverstärkers in digitale mit einer Auflösung von $12bit$ zu konvertieren. Im jetzigen Stand werden nur $11bit$ verwendet, da der ADC nur im positiven Bereich von $0V$ bis $2,5V$ Werte vom Sensor erhält. Um das Sensorsignal besser nutzen zu können, kann an den Referenzsignalen ein Offset von $0,4V$ gelegt werden, sodass der ADC Werte vom Sensor im Bereich von $0V$ bis $3,3V$ entgegennehmen kann.

Anhand der beiden Komparatoren *DIFF1* und *DIFF2* ist es möglich, jede beliebige Frequenz eines Signals zu bestimmen. Hier wurden zwei Komparatoren in Kombination mit einem Zustandsautomaten bevorzugt. Der Grund liegt an den einzelnen Komparatorsignalen, die sowohl bei steigender als auch bei fallender Flanke mehrere unerwünschte Pulse aufweisen. Durch den eingebauten Zustandsautomaten in Kombination mit zwei Komparatoren sind die Komparatorsignale nur zu zweit von Interesse. Der Zustandsautomat wechselt erst in den nächsten Zustand, sobald beide Signale gleich sind. Dadurch wird die Bestimmung der Periodendauer ermöglicht. Mit nur einem Komparator wäre dies nicht auf diese einfache Art und Weise umsetzbar, denn die unerwünschten Pulse müssten dann zusätzlich berücksichtigt werden.

Es wurde im FPGA eine Funktion erstellt, die die aufgenommene Zeit für eine komplette Periode durch 64 teilt, um genau 64 Werte für eine Periode abspeichern zu können. Dies gilt für alle drei Eingangssignale *U_DIFF*, *U_HB1* und *U_HB2*. Dieser Frequenzbereich liegt zwischen $1Hz$ und $2,5kHz$ für den ABS-Sensor. Erfolgreich getestet wurden über $10kHz$, sodass hier genug Puffer vorhanden ist.

In der gleichen Funktion für die Berechnung der Periodendauer wird ebenfalls die Bestimmung der Verstärkungsstufe für das Vorverstärker-Modul durchgeführt. Diese beiden Funktionen befinden sich im gleichen Zustandsautomaten, da sie die gleichen Signalabfragen tätigen. Dadurch kann Hardwarefläche eingespart werden.

Neben dem ADC befindet sich ebenfalls ein DAC auf der ADC-Platine. Der DAC ermöglicht das Setzen der Referenzspannungen für die Komparatoren und sorgt für die Offsetkompensation des Vorverstärker-Moduls, welches erfolgreich umgesetzt wurde.

Damit das Beschreiben und Auslesen der Register sowohl vom ADC als auch vom DAC funktionieren, wurde eine Steuerung diesbezüglich entwickelt, die besonders auf das Timing und das sichere Anlegen und Abholen der Daten achtet. Es wird zudem dafür gesorgt, dass bei der *Sequence*-Funktion, dem sogenannten „Scan-Modus“, keine Kanalwechsel passieren, da insgesamt drei Kanäle konvertiert werden müssen und jeweils zwei neue Werte nach jedem Zyklus zur Verfügung stehen. Hier gab es anfangs Probleme, dass die entsprechenden Werte ihren Kanälen nicht korrekt zugewiesen wurden. Dies konnte dadurch gelöst werden, indem die letzten beiden Bits des *Sequence*-Registers ausgelesen und ausgewertet wurden. Wenn neue Daten in die Arrays gespeichert werden sollen, müssen die letzten beiden Bits das Muster „01“ aufzeigen. Wenn dies der Fall ist, werden alle Arrays komplett mit neu konvertierten Werten gefüllt. Ansonsten muss solange gewartet werden, bis das Muster übereinstimmt.

Diese aufgenommenen Werte lassen sich durch eine weitere Funktion über die 7-Segment-Anzeige darstellen und mit den Tastern des NEXYS2-Boards durchklicken. Einige Register-Werte lassen sich ebenfalls durch bestimmte Switch-Einstellungen darstellen. Dies hat den besonderen Vorteil, dass die Werte direkt in der Hardware überprüft werden können. Dadurch lassen sich schnell Probleme beispielsweise in der Speicherung von Werten oder beim Senden der Werte an Matlab eingrenzen. Sie können ebenfalls mit den übertragenen Daten an Matlab verglichen werden, um die Funktionalität der Module zu überprüfen. Dies kam zum erfolgreichen Einsatz, als die serielle Schnittstelle mit dem PC aufgebaut wurde. Die Daten stimmten nicht überein, welches durch den Vergleich der 7-Segment-Anzeige und den in Matlab empfangenen Daten festgestellt werden konnte. Dies allein reichte zwar nicht, um den Fehler komplett eingrenzen zu können, wies aber in eine Richtung. Die Fehlersuche wurde anschließend mit dem Logicanalyzer durchgeführt, mit dem das Problem eingrenzen lies.

Die Kommunikation zwischen dem FPGA und dem PC konnte ebenfalls erfolgreich mit dem Fremdmodul [3] erstellt werden. Nach dem Aufbau der Verbindung vom FPGA zum PC konnten immer neu konvertierte Werte vom ADC an Matlab gesendet werden. Es wurde anschließend in VHDL ein Modul geschrieben, das ebenfalls das Empfangen ermöglichte. Dazu wurde eine Auswertung hinzugefügt, wodurch es Matlab ermöglicht wurde, die Kontrolle und Steuerung des FPGAs zu übernehmen. Dies konnte ebenfalls erfolgreich umgesetzt werden und hat den besonderen Vorteil, dass die Einstellungen im ADC und DAC im laufenden Betrieb geändert werden können, ohne den VHDL-Code ändern zu müssen.

Bei dieser seriellen Schnittstelle kann entweder nur empfangen oder gesendet werden. Dies ist im jetzigen Stand völlig ausreichend. Wenn aber später die Befehle schneller gesetzt und Daten angefordert werden, reicht dies nicht aus. Dann fehlt noch ein zusätzliches Modul, das für die Zwischenspeicherung der Befehle zuständig sein muss.

Die empfangenen Daten in Matlab werden mit einem erfolgreich umgesetzten Treiber entgegengenommen und an das m-File übergeben. Der Treiber sorgt dafür, dass das m-File übersichtlich bleibt und die Kommunikation bzw. Steuerung des FPGAs deutlich vereinfacht wird. Im m-File werden die Daten ausgewertet und können mit dem Matlab-Skript ausgegeben werden. Die Signaldaten lassen sich in einem Plot mit drei Darstellungsfenstern anzeigen.

Es konnten alle Aufgaben gelöst werden, außer in einem Punkt. Es handelt sich dabei um die Signale, die vom Vorverstärker-Modul der ADC-Platine übergeben werden. Diese weisen alle Störungen auf, die bei allen drei Signalen in die gleiche Richtung an der gleichen Stelle auftreten. Dieses Problem wurde bereits im Abschnitt 6.2.1 diskutiert.

7.2 Ausblick

Diese Arbeit lässt sich noch an einigen Stellen optimieren. Hierzu folgt eine Liste von den Stellen.

- Es kann die RS232-Schnittstelle mit gleichzeitigem Schreiben und Lesen erweitert werden. Dies wäre jetzt auch möglich, wobei aber die Befehle nicht im FPGA ausgewertet werden. Dies macht auch nur dann Sinn, wenn ein FIFO hinzugefügt wird, der die einzelnen Befehle speichert, sodass diese hintereinanderweg abgearbeitet werden können.
- Um die Kommunikation bzw. die Verbindung zwischen dem FPGA und dem PC zu beschleunigen, kann die Baudrate deutlich erhöht werden. Dies ist aber nicht mit der jetzigen Taktfrequenz möglich. Es müsste das Modul DCM (Digital Clock Manager) genutzt werden, mit dem jede beliebige Taktfrequenz einstellbar ist. In diesem Zusammenhang kann ebenfalls der aktuell genutzte Takt von $12,5MHz$ über den DCM betrieben werden, um beim Erstellen des Taktes mittels des Taktteilers Clock-Skews zu vermeiden.
- Eine andere Möglichkeit der Beschleunigung der Datenübertragung mit dem PC würde mit einem anderen Interface realisierbar sein. Das FPGA ist bereits mit einem MiniUSB mit dem PC verbunden, um den Bit-Stream auf das FPGA zu laden. Es wäre möglich, hier ebenfalls die Datenübertragung für

die Steuerung und die Kommunikation generell mit Hilfe dieses Interfaces zu realisieren.

- Eine weitere Optimierung der Arbeit ist in dem Steuerpfad der ADC-Platine zu finden. Hier sind viele Aufgaben mit dem gleichen Ablauf versehen, die aber mit ihrem eigenen Ablauf abgearbeitet werden. Dies ließe sich durch einen Zustandsablauf realisieren und würde mehrere Zustände einsparen können und somit für mehr Platz auf dem FPGA sorgen.
- Wie in Kapitel 6 festgestellt wurde, sind die Eingangssignale des ADCs mit Störungen versehen. Dieses Problem sollte beseitigt werden beispielsweise mit dem vorgeschlagenen Lösungsweg.
- Zu dem Schaltungsaufbau können noch einige Optimierungen getätigt werden. Hier würde es Sinn machen, die Referenzsignale für die Eingangssignale nicht auf Masse zu führen, sondern mit einer Spannung von $0,4V$ zu erhöhen. Dadurch ist es möglich, den Spannungsbereich von $0V$ auf $3,3V$ zu erweitern.

Literaturverzeichnis

[1] BOLL, R. ; OVERSHOTT, K. J.: Magnetic Sensors. In: GÖPEL, W. (Hrsg.) ; HESSE, J. (Hrsg.) ; ZEMEL, J. N. (Hrsg.): *Sensors a Comprehensive Survey* Bd. 5. Weinheim : VCH, 1989. – ISBN 3–527–26771–9. – insb. Kap. 9 von U. Dibbern

[2] CARTON, Philippe: *MiniUART IP Core Specification*. Version: September 2002. `http://opencores.org/websvn,filedetails?repname=miniuart2&path=%2Fminiuart2%2Ftrunk%2Fdoc%2FMiniUart.pdf`, Abruf: 28.04.2010

[3] CARTON, Philippe: *miniuart2*. Version: 2002. `http://opencores.org/project,miniuart2`, Abruf: 28.04.2010

[4] DEVICES, Analog: *AD5348*. Version: 2003. `http://www.analog.com/static/imported-files/data_sheets/AD5346_5347_5348.pdf`, Abruf: 26.02.2010

[5] DEVICES, Analog: *REF19x*. Version: 2003. `http://www.analog.com/static/imported-files/data_sheets/REF19xSeries.pdf`, Abruf: 24.02.2010

[6] DRESCHHOFF, Jan-Heiner: *VHDL-Implementierung des Ausgabeprotokolls von ABS-Sensoren*, Hochschule für Angewandte Wissenschaften Hamburg, Studienarbeit, Dezember 2009

[7] HEIKO POPPINGA: *Controller-Implementation und messtechnische Erprobung der Signalverarbeitung für die Diagnosefunktion von ABS-Sensoren*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, voraussichtlich April 2011

[8] HERVEILLE, Richard: *WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores*. Version: September 2002. `http://cdn.opencores.org/downloads/wbspec_b3.pdf`, Abruf: 19.03.2010

[9] INSTRUMENTS, Texas: *ADS7865*. Version: October 2008. `http://focus.ti.com/lit/ds/sbas441b/sbas441b.pdf`, Abruf: 26.02.2010

[10] JAN-HEINER DRESCHHOFF: *FPGA-Prototyp der Signalverarbeitung fur ABS-Sensoren mit Diagnosefunktion als VHDL-Implemenation*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, September 2010

[11] JEGENHORST, Niels: *Entwicklung eines Controllersystems zur Zustandserkennung von ABS-Sensoren*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, Oktober 2009

[12] KALIN IVANOV: *Fehlersichere Automatisierung eines Encoder-Messplatzes zur Untersuchung von ABS-Sensoren*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, voraussichtlich 2011

[13] KOCH, Lennart: *Aufwandsminimierte Schätzung von Harmonischen zur Zustandsbestimmung von ABS-Sensoren*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, April 2010

[14] MAHTOUF, Abdelkhalek: *Messungen und Signalanalyse an einem magnetischen Sensor*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, Dezember 2008

[15] MARKUS PIOREK: *Hard- und Software eines Messplatzes zur Harmonischen Analyse bei magnetischen Winkelsensoren*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, voraussichtlich März 2011

[16] PHILIPS: *KMI22/1 Rotational speed sensor for extended air gap application and direction detection*. 2000

[17] PHILIPS SEMICONDUCTORS: *Semiconductor Sensors Data Handbook SC17*. 2001

[18] SCHOERMER, Christian: *Automatisierter Radmessplatz fur ABS-Sensoren mit aktiven und passiven Encodern verschiedener Automobil-Hersteller*, Hochschule für Angewandte Wissenschaften Hamburg, laufende Diplomarbeit

[19] SCHOERMER, Christian: *AMR-Messbrücken fur ABS-Sensoren*, Hochschule für Angewandte Wissenschaften Hamburg / NXP Semiconductors, Studienarbeit, März 2008

[20] SIEBENMORGEN, Frank: *Ansteuerelektronik und Mikrocontrollersteuerung eines Kreuzspulenmessplatzes für ABS-Sensoren*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, Juni 2009

[21] STAHL, Martin: *Controllersystem zur Verstärkungsregelung und Offsetkompensation für ABS-Sensoren mit Diagnosefunktion*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, Februar 2010

[22] TEXAS INSTRUMENTS: *MSP430F15x, MSP430F16x, MSP430F161x Mixed Signal Microcontroller (SLAS368F)*. Version: Mai 2009, Oktober 2002. `http://focus.ti.com/lit/ds/symlink/msp430f1611.pdf`, Abruf: 24.07.2009

[23] XILINX: *XST User Guide, 2005 edition*. Version: 2005. `www.xilinx.com`, Abruf: 11.03.2010

[24] XILINX: *Contraints Guide, ug331 (v1.7) edition*. Version: 2010. `www.xilinx.com`, Abruf: 22.09.2010

A Schaltpläne, Entwürfe & Simulation

A.1 Pinbelegungen

In Bild A.1 wird die Pinbelegung des Verbindungsboards gezeigt.

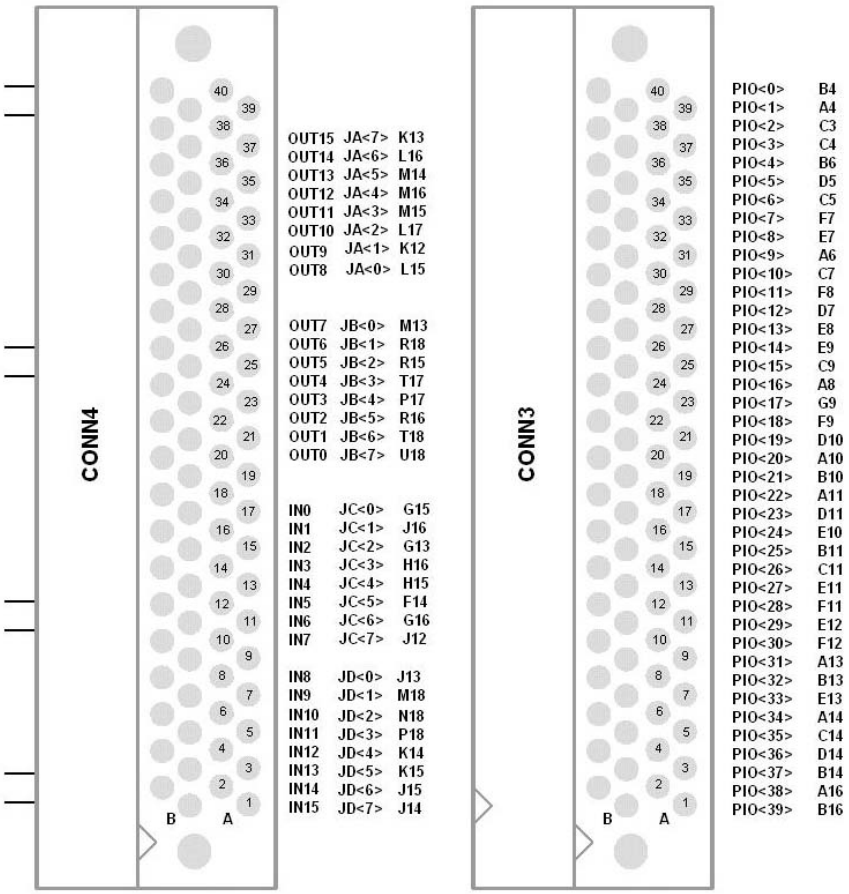


Bild A.1: *Pinbelegung des Verbindungsboards NEXYS2 To Modsys Connectionboard*

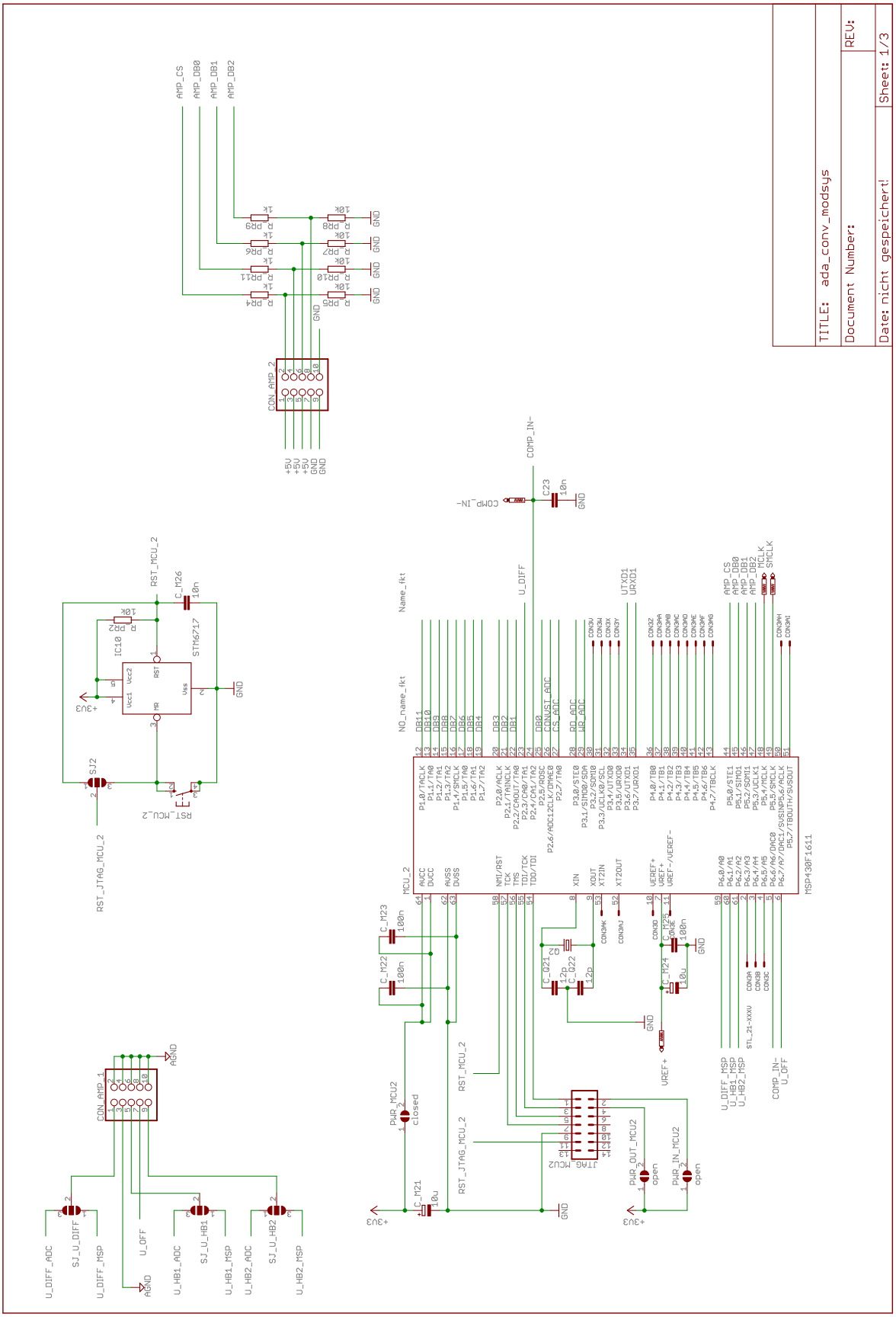
In Tabelle A.1 sind die Bezeichnungen für die Pinbelegung in der Hardware und dem FPGA im Vergleich dargestellt.

Pin-Nr.	Pin-Name	Pin-Bez. im FPGA	ADC/DAC	MSP
1	PIO<39>	DB<11>	DB11	P1.0
2	PIO<38>	DB<10>	DB10	P1.1
3	PIO<37>	DB<9>	DB9	P1.2
4	PIO<36>	DB<8>	DB8	P1.3
5	PIO<35>	DB<7>	DB7	P1.4
6	PIO<34>	DB<6>	DB6	P1.5
7	PIO<33>	DB<5>	DB5	P1.6
8	PIO<32>	DB<4>	DB4	P1.7
9	PIO<31>	DB<3>	DB3	P2.0
10	PIO<30>	DB<2>	DB2	P2.1
11	PIO<29>	DB<1>	DB1	P2.2
12	PIO<28>	DB<0>	DB0	P2.5
13	PIO<27>	ADC_CONVSTn	CONVST_ADC	P2.6
14	PIO<26>	ADC_CLK	CLK_ADC_IN	-
15	PIO<25>	ADC_CSn	CS_ADC	P2.7
16	PIO<24>	ADC_RDn	RD_ADC	P3.0
17	PIO<23>	ADC_WRn	WR_ADC	P3.1
18	PIO<22>	ADC_BUSY	BUSY_ADC	-
19	PIO<21>	DAC_A<2>	A2	-
20	PIO<20>	DAC_A<1>	A1	-
21	PIO<19>	DAC_A<0>	A0	-
22	PIO<18>	DAC_BUF	BUF	-
23	PIO<17>	DAC_GAIN	GAIN	-
24	PIO<16>	DAC_CSn	CS_DAC	-
25	PIO<15>	DAC_RDn	RD_DAC	-
26	PIO<14>	DAC_WRn	WR_DAC	-
27	PIO<13>	-	SIG_27	-
28	PIO<12>	-	SIG_28	-
29	PIO<11>	sigCompUdiff1	DIFF1	-
30	PIO<10>	sigCompUdiff2	DIFF2	-
31	PIO<9>	sigCompHB1	HB1	-
32	PIO<8>	sigCompHB2	HB2	-
33	PIO<7>	AMP_CS	AMP_CS	P5.3
34	PIO<6>	AMPgain<2>	AMP_DB2	P5.2
35	PIO<5>	AMPgain<1>	AMP_DB1	P5.1
36	PIO<4>	AMPgain<0>	AMP_DB0	P5.0
37	PIO<3>	-	SIG_37	-

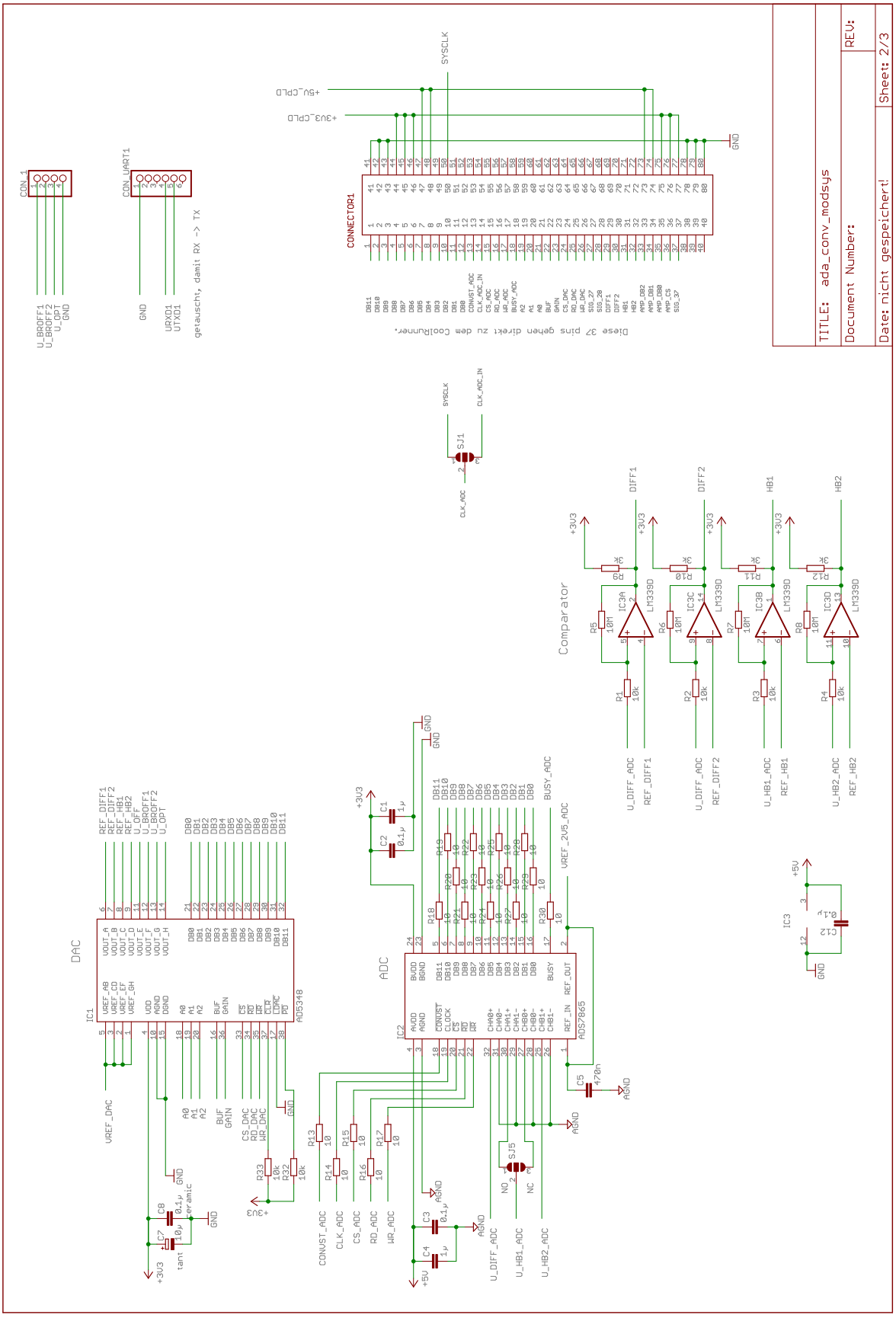
Tabelle A.1: *Pinbelegung im FPGA und auf der Platine*

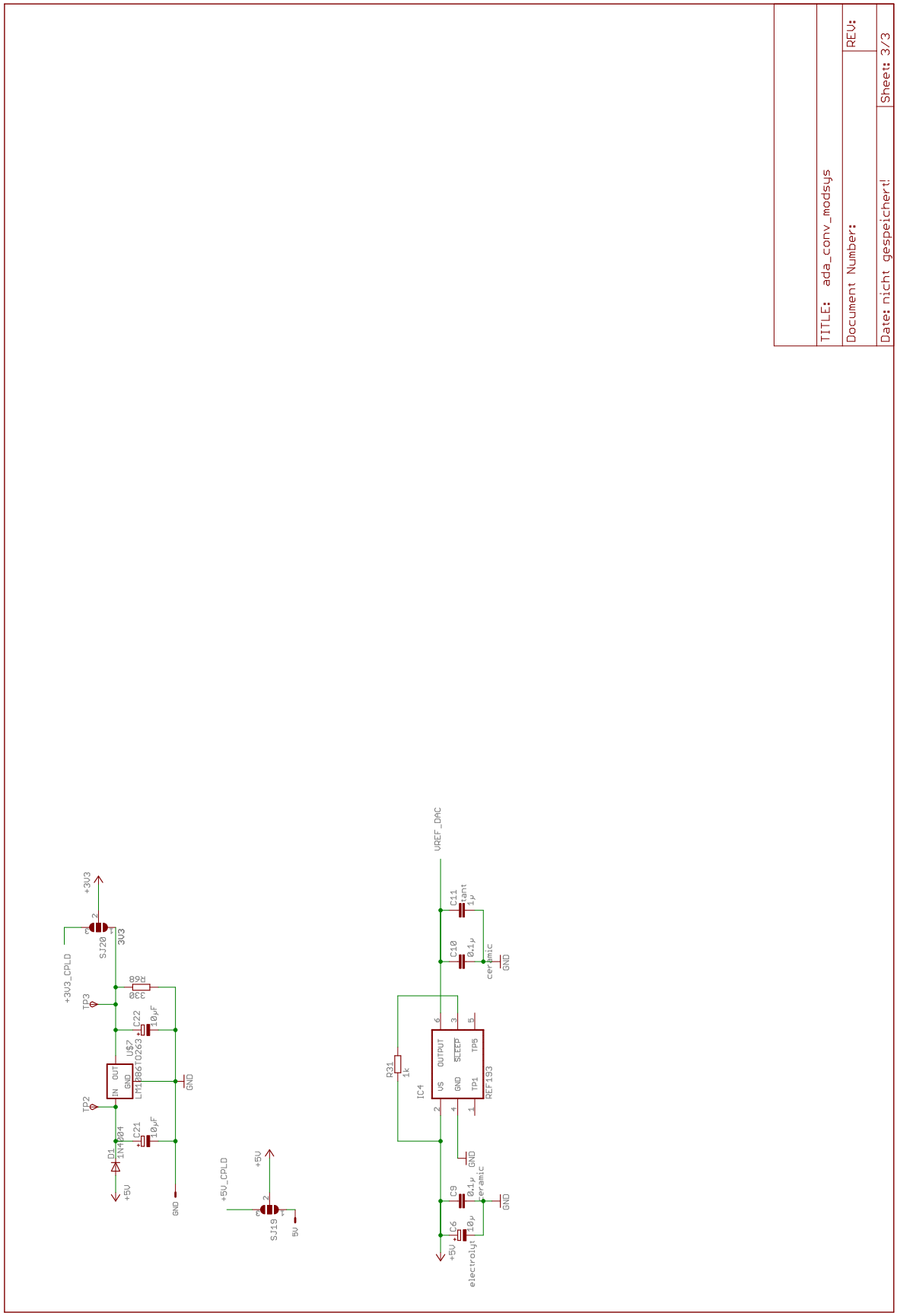
A.2 Hardware - Schaltungsaufbau

Hier werden die erstellten Schaltungspläne gezeigt.



12.02.2011 20:15:39 f=0.73 D:\Bachelor\board\ada_conv_modsys.sch (Sheet: 1/3)


TITLE: ada_conv_modsys
Document Number:
REV:
Date: nicht gespeichert!
Sheet: 2/3
DAC
ADC
Comparator
CONNECTOR1
CON_1
CON_UART1
SYSCLK
+3V3_CPLD
+5V_CPLD
getauscht, damit RX -> TX
Diese 37 pins gehen direkt zu dem CoolRunner.
U_BROFF1
U_BROFF2
U_DIFF1
U_DIFF2
GND
URXD1
UTXD1
SJ1
SJ5
IC1
IC2
IC3
IC4
AD5348
AD57865
LM339D
DIFF1
DIFF2
HB1
HB2
VREF_DAC
CONVST_ADC
CLK_ADC
CS_ADC
RD_ADC
WR_ADC
U_DIFF_ADC
U_HB1_ADC
U_HB2_ADC
REF_DIFF1
REF_DIFF2
REF_HB1
REF_HB2
BUSY_ADC
VREF_2V5_ADC


TITLE: ada_conv_modsys
Document Number:
REV:
Date: nicht gespeichert!
Sheet: 3/3
+3V3
SJ20
3V3
+3V3_CPLD
TP3
89Ω
33Ω
C22
10µF
U$7
LM1086IT263
IN
OUT
GND
TP2
C21
10µF
D1
1N4004
+5V
GND
+5V_CPLD
+5V
SJ19
5V
VREF_DAC
C11
tant
1µ
C10
0.1µ
ceramic
GND
R21
1k
IC4
OUTPUT
SLEEP
VS
GND
TP5
TP1
REF193
C9
0.1µ
ceramic
GND
C6
10µ
+5V
electroly

A.3 Hardware - Schaltungsentwurf

Hier werden das Toplayer und das Bottomlayer des Designs.

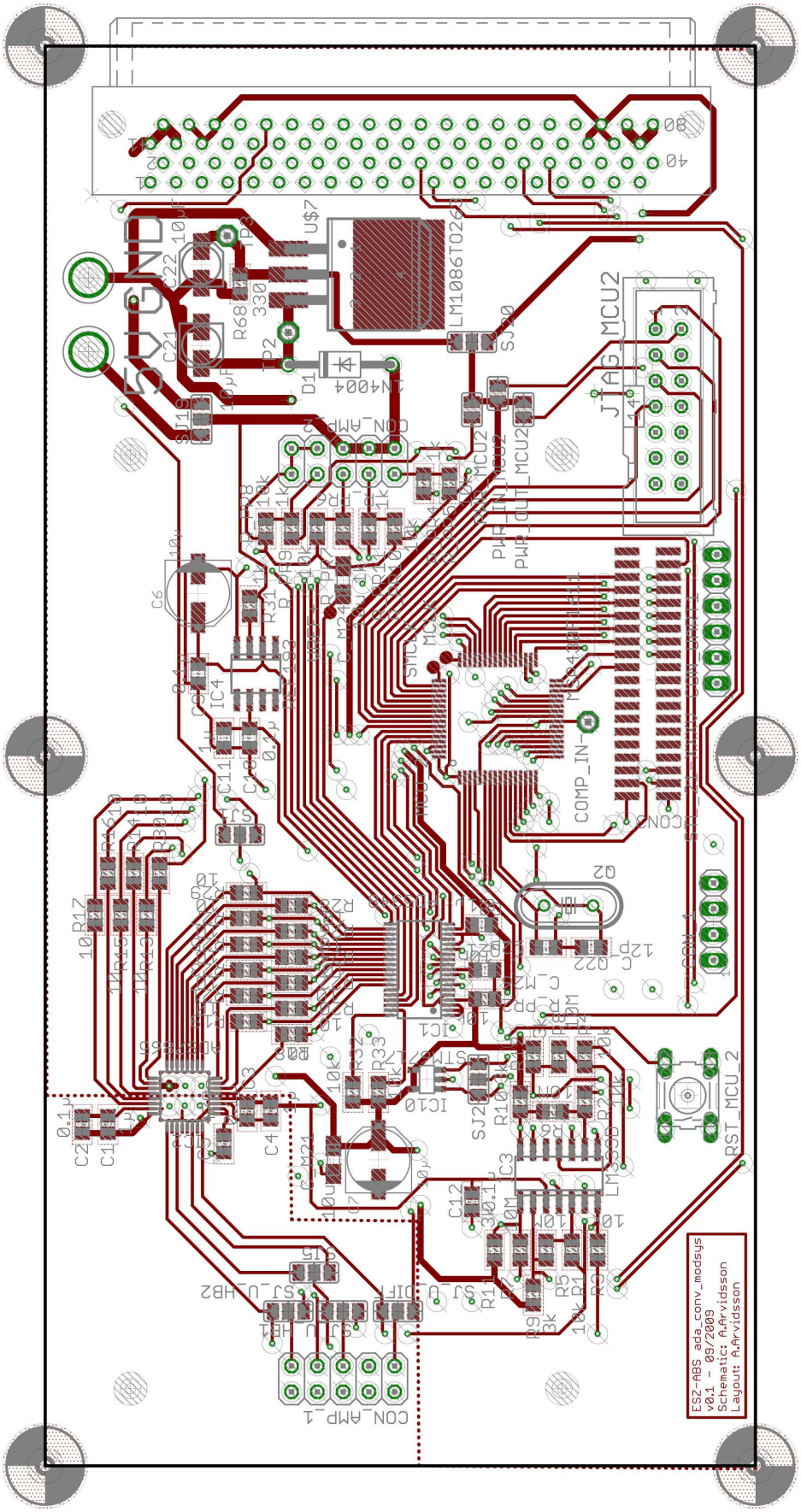


Bild A.2: *Toplayer des fertigen Designs*

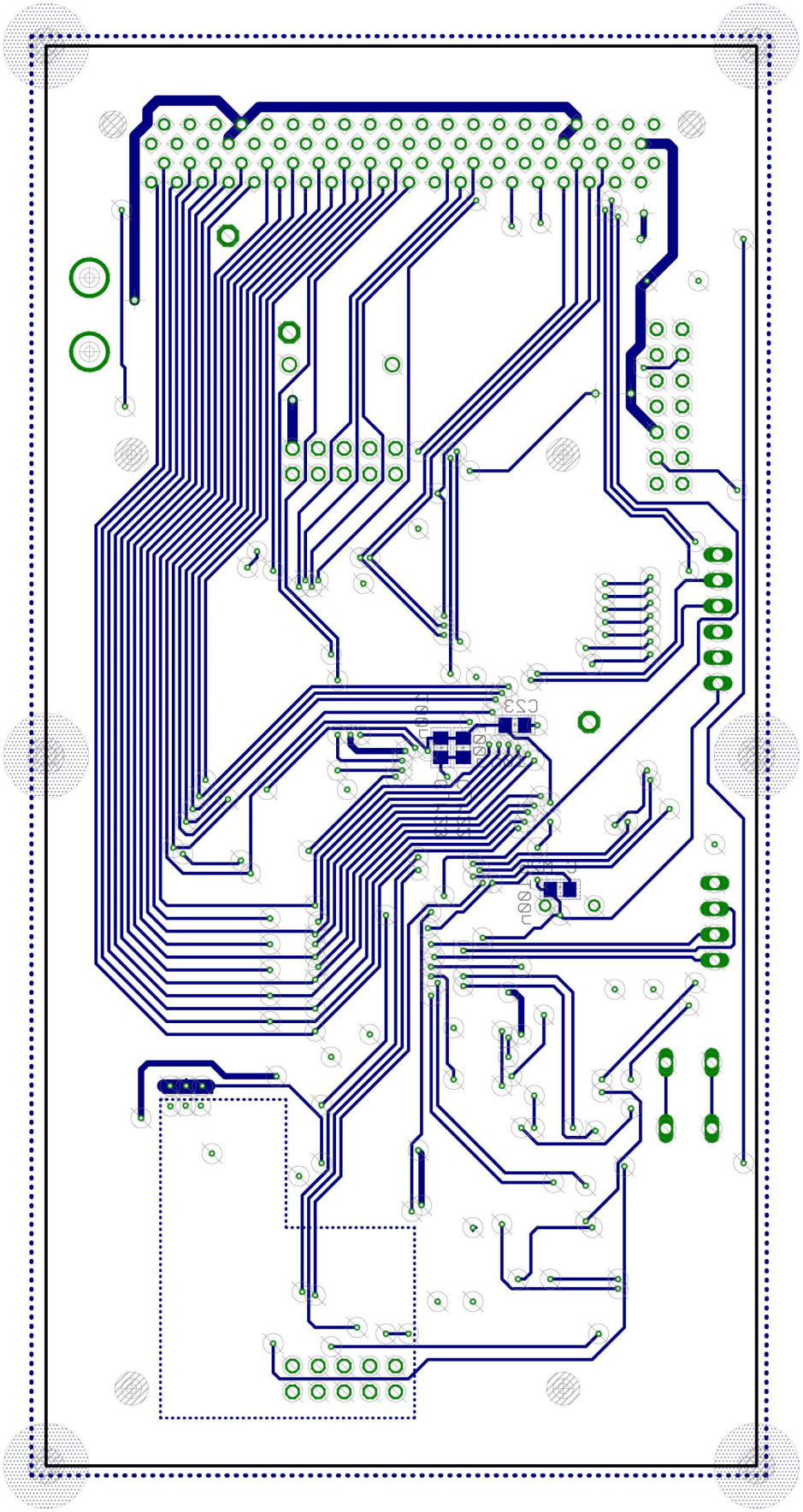


Bild A.3: *Bottomlayer des fertigen Designs*

A.4 Simulationen

A.4.1 Funktionale Simulation

In den nachfolgenden Simulationsbildern wird gezeigt:

- Schreibphase (Bild A.4)
- Lesephase (Bild A.5)
- Bestimmung der Periodendauer (Bild A.9)
- Bestimmung der Verstärkung (Bild A.10)
- Speicherung von neuen Daten in die drei Arrays (Teil 1 von 3) (Bild A.6)
- Speicherung von neuen Daten in die drei Arrays (Teil 2 von 3) (Bild A.7)
- Speicherung von neuen Daten in die drei Arrays (Teil 3 von 3) (Bild A.8)

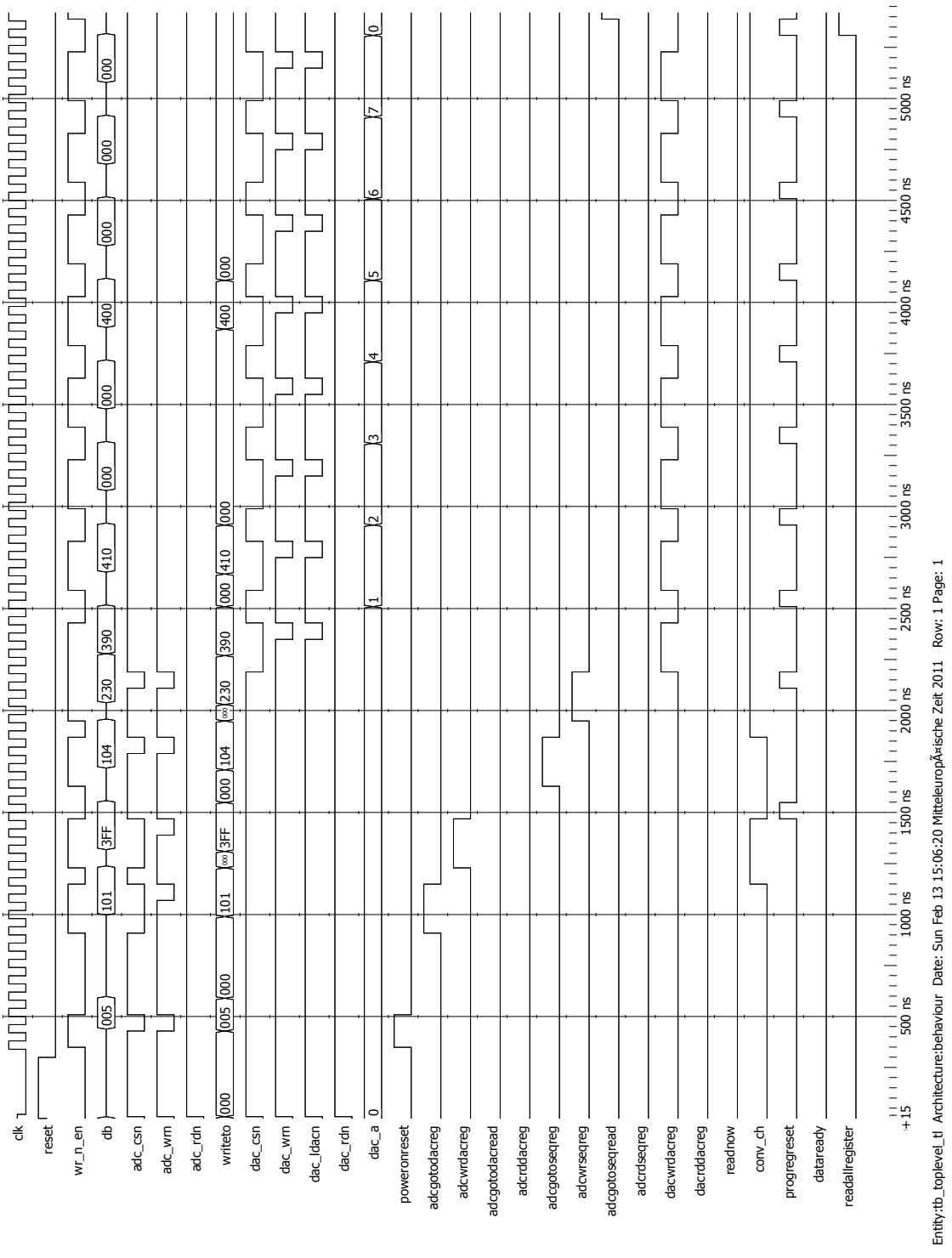


Bild A.4: *Schreibphase der Initialisierung*

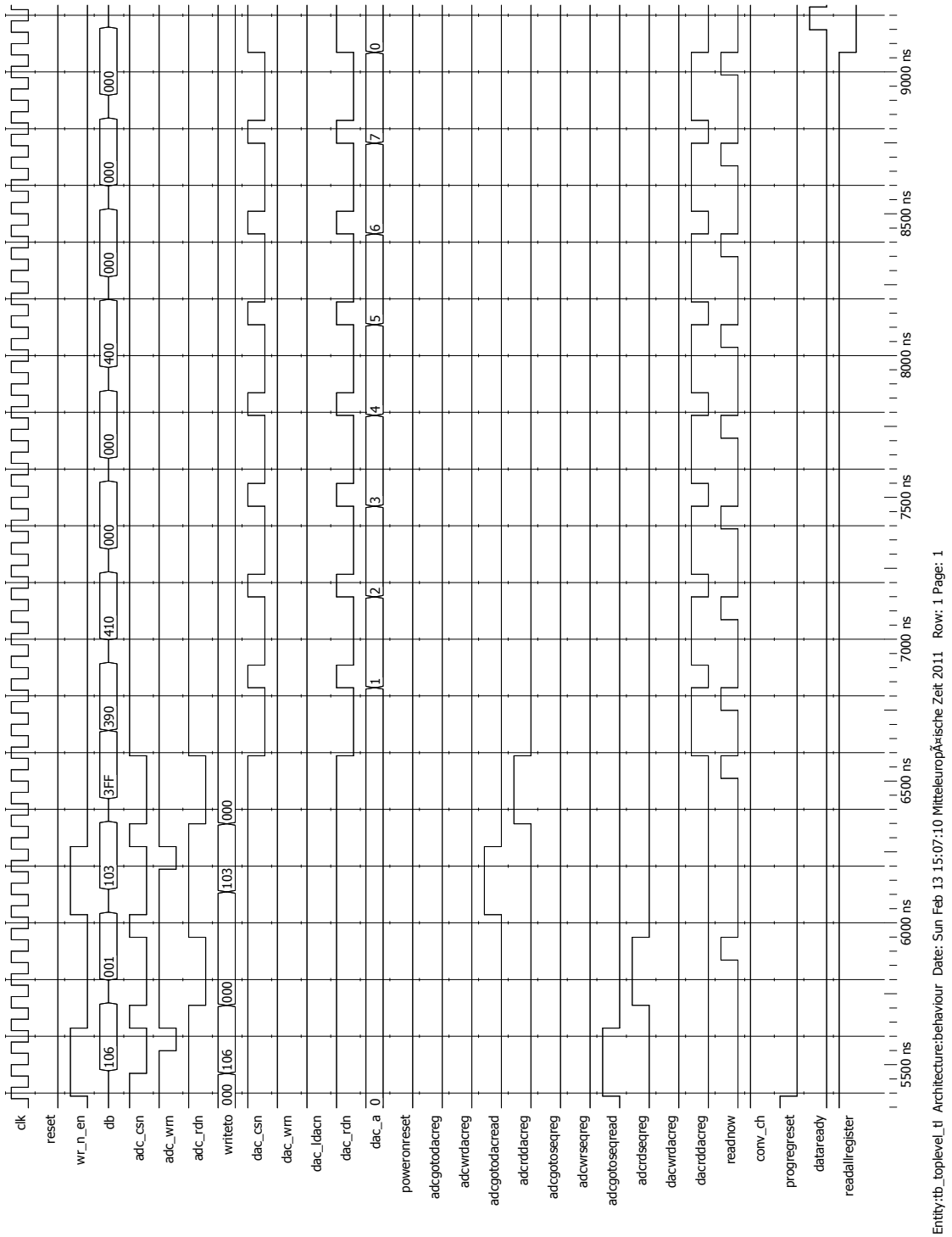


Bild A.5: *Lesephase der Initialisierung*

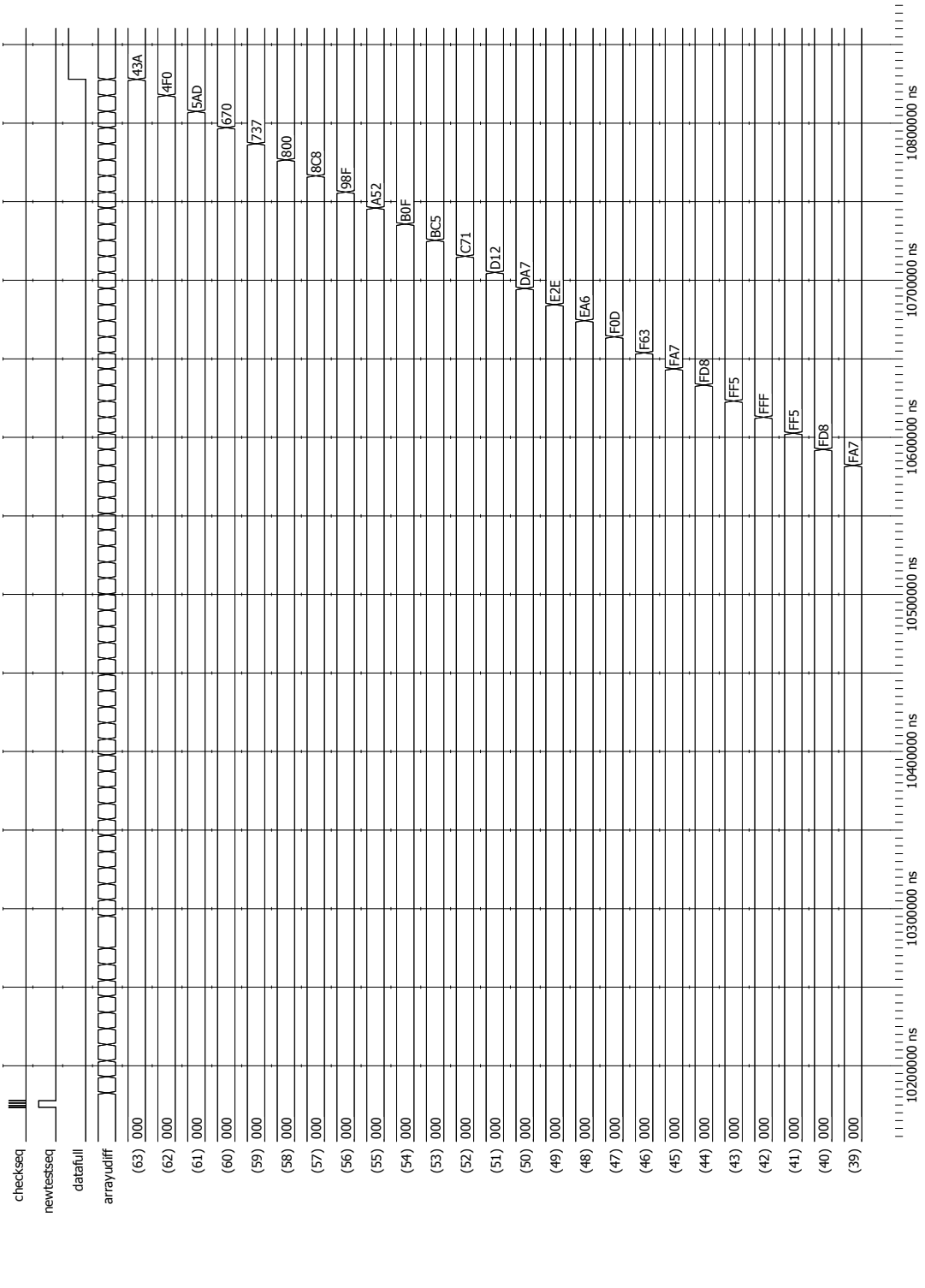


Bild A.6: *Speichern neuer Daten in das Array arrayudiff Teil 1 von 3*

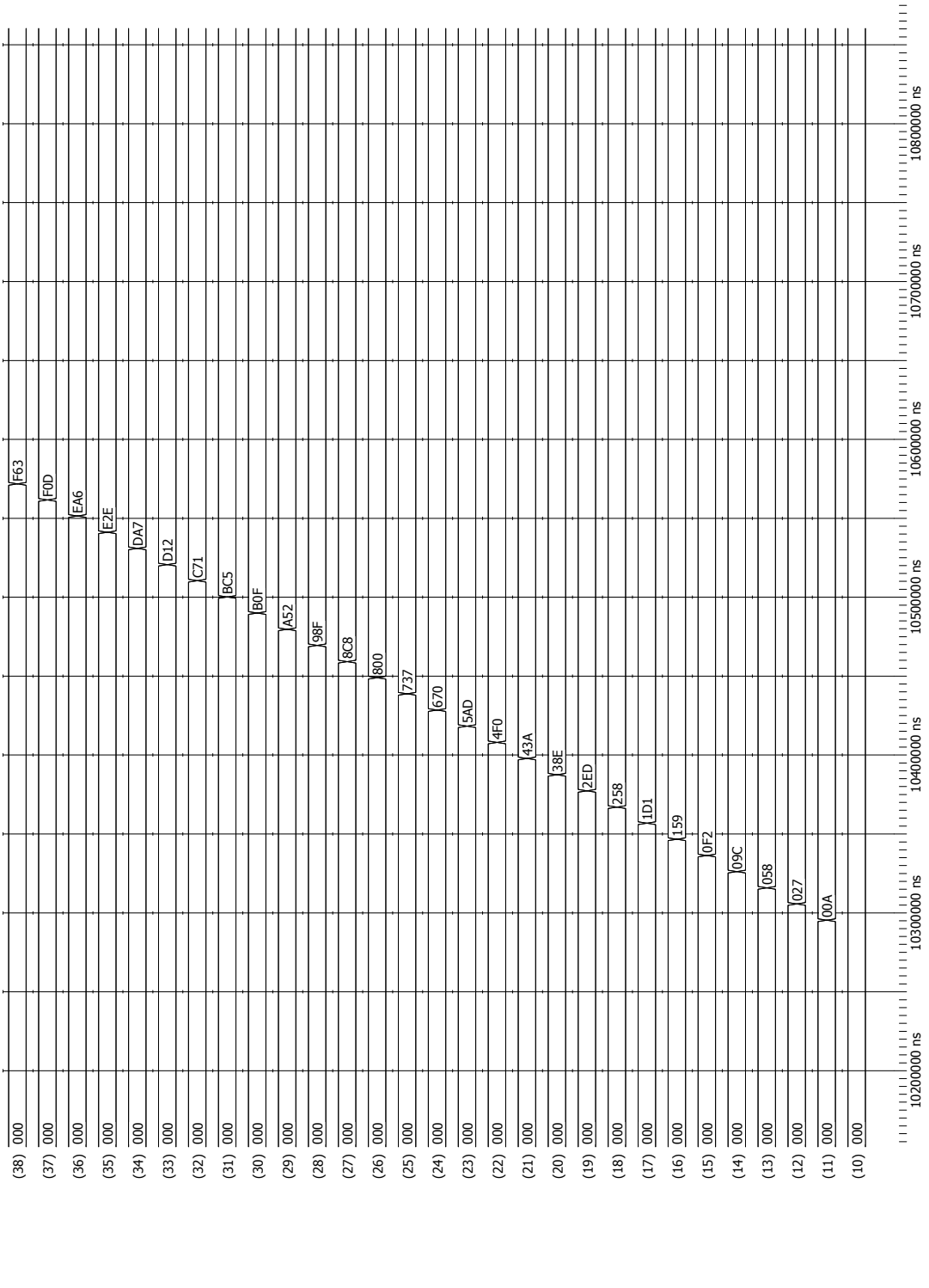


Bild A.7: *Speichern neuer Daten in das Array arrayudiff Teil 2 von 3*

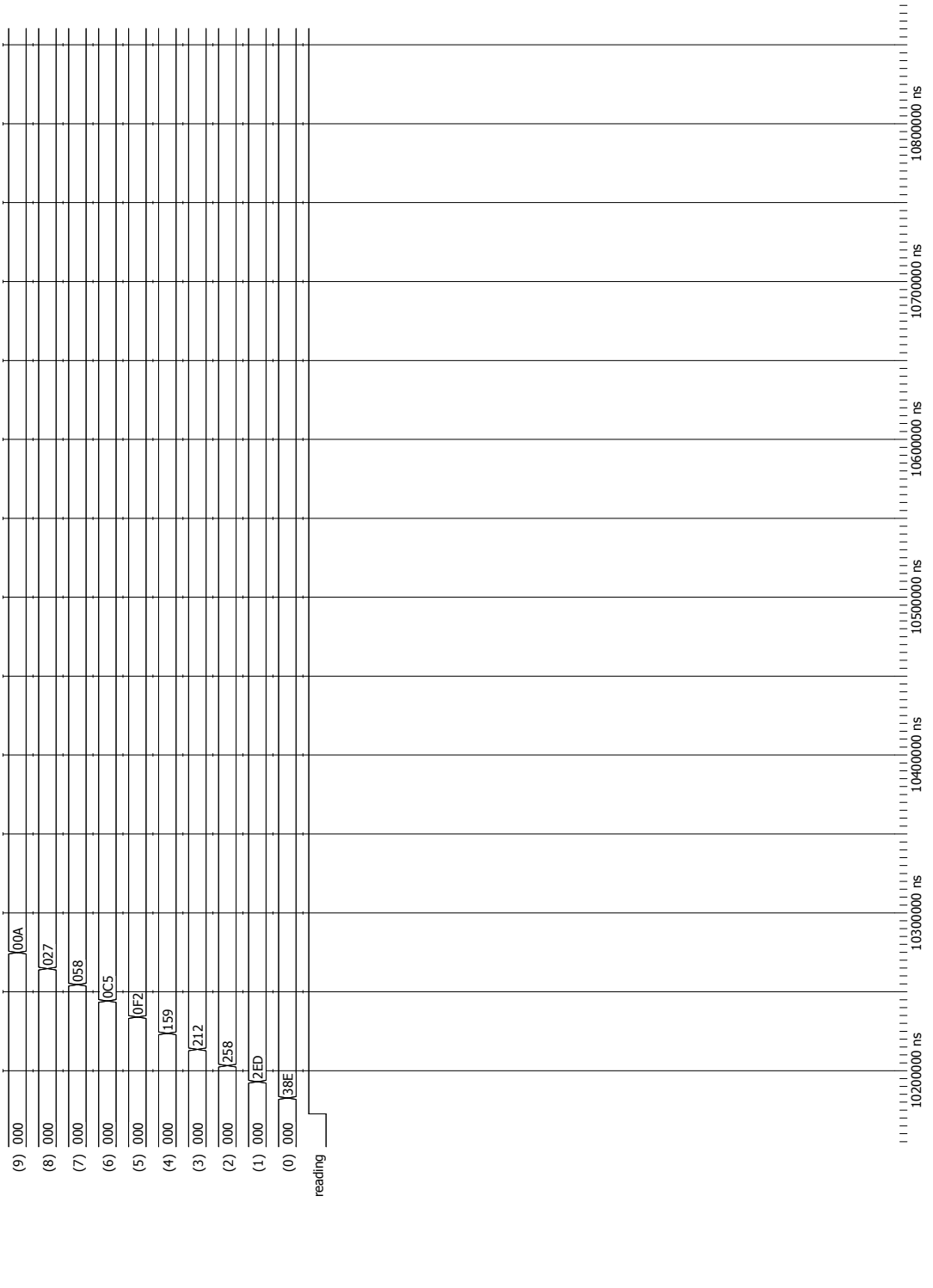


Bild A.8: *Speichern neuer Daten in das Array arrayudiff Teil 3 von 3*

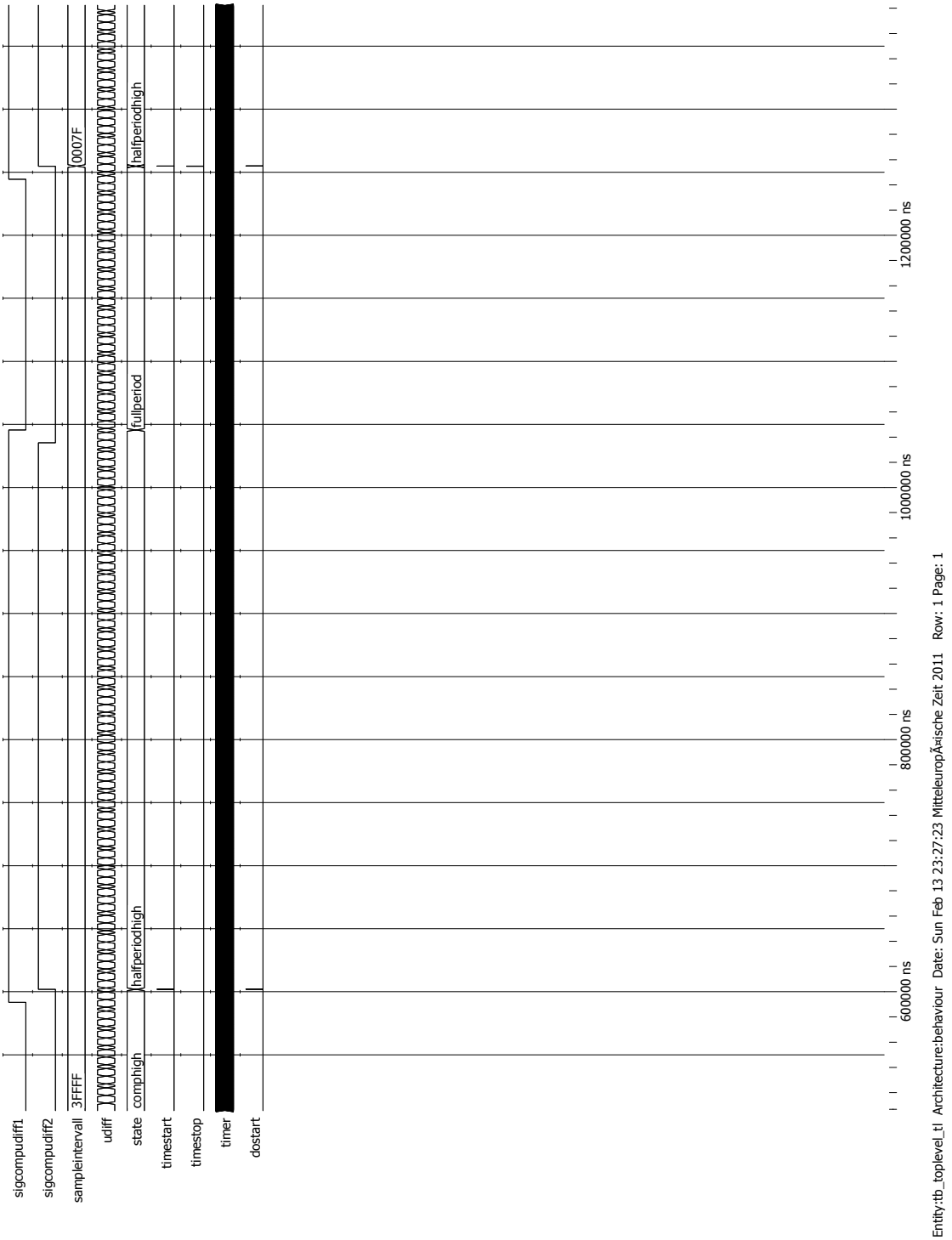


Bild A.9: *Bestimmung der Periodendauer*

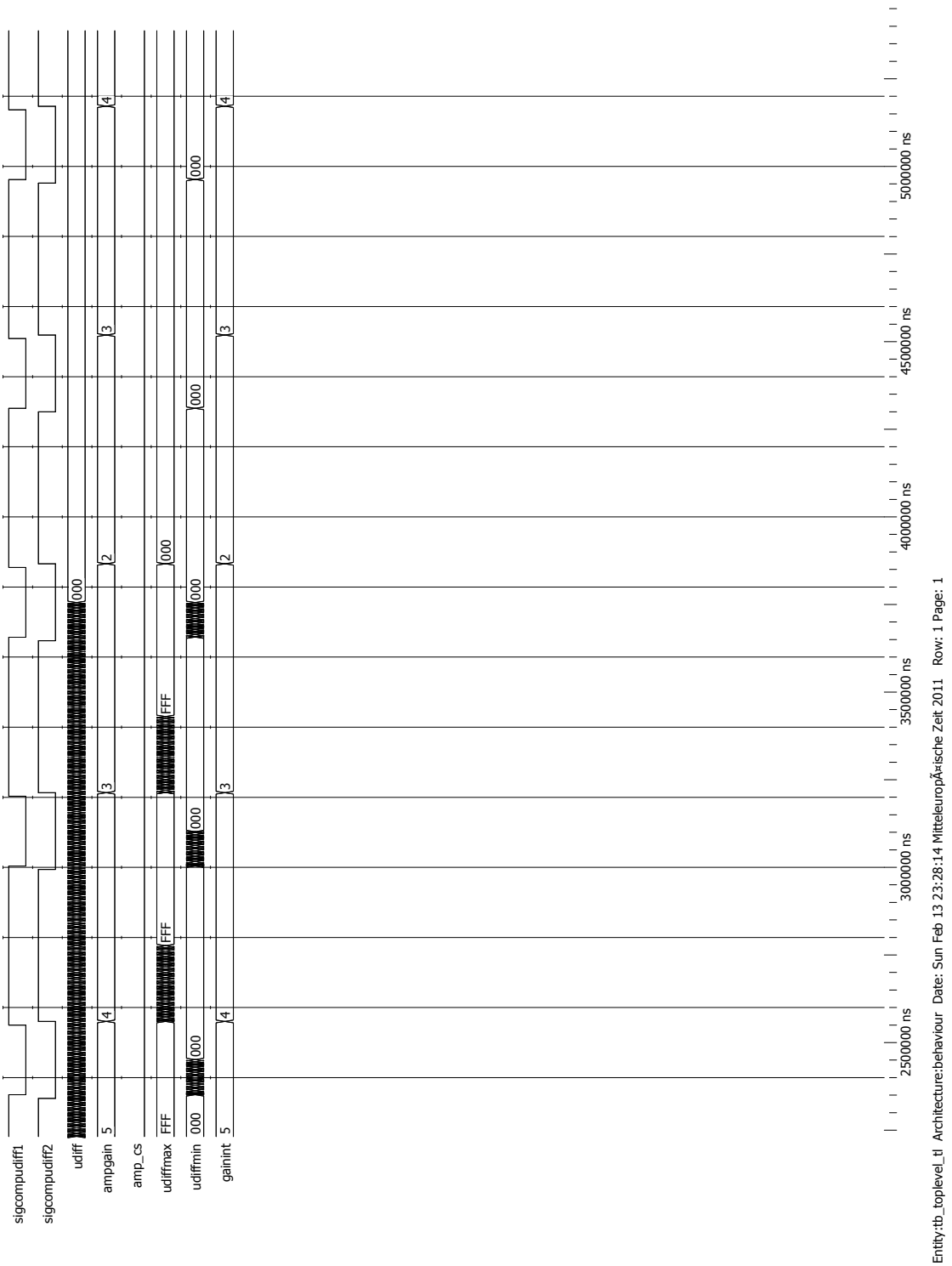


Bild A.10: *Bestimmung der Verstärkung*

B Darstellungen und Tabellen

B.1 Hardware - VHDL

B.1.1 Darstellungen

Hier sind die beiden Darstellungen für die Toplevel *ada_conv_modsys_TL* und *Mini-UART_TL* aufgeführt mit den einzelnen verwendeten VHDL-Modulen.

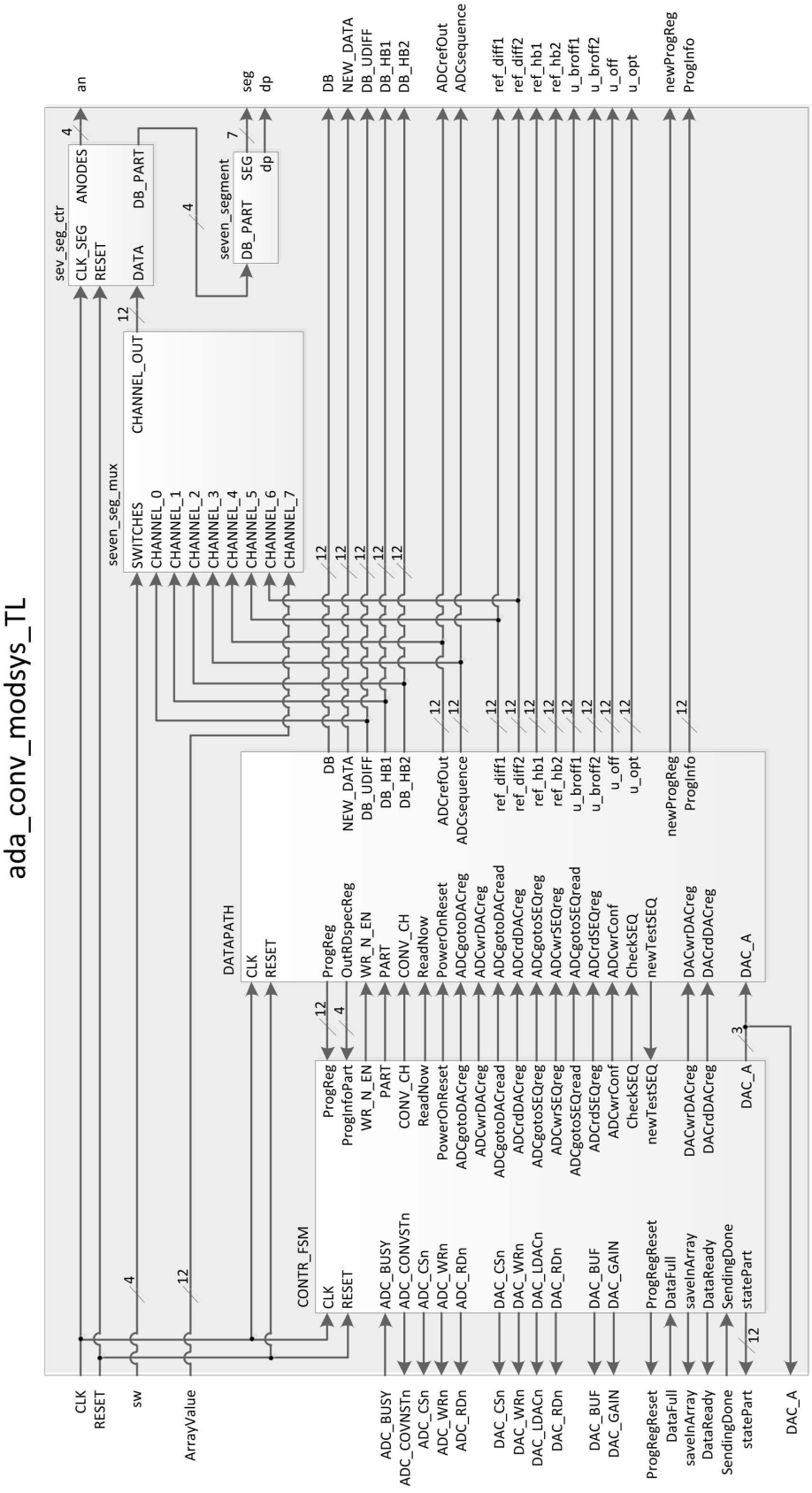


Bild B.1: *Toplevel vom ada_conv_modsys-Modul*

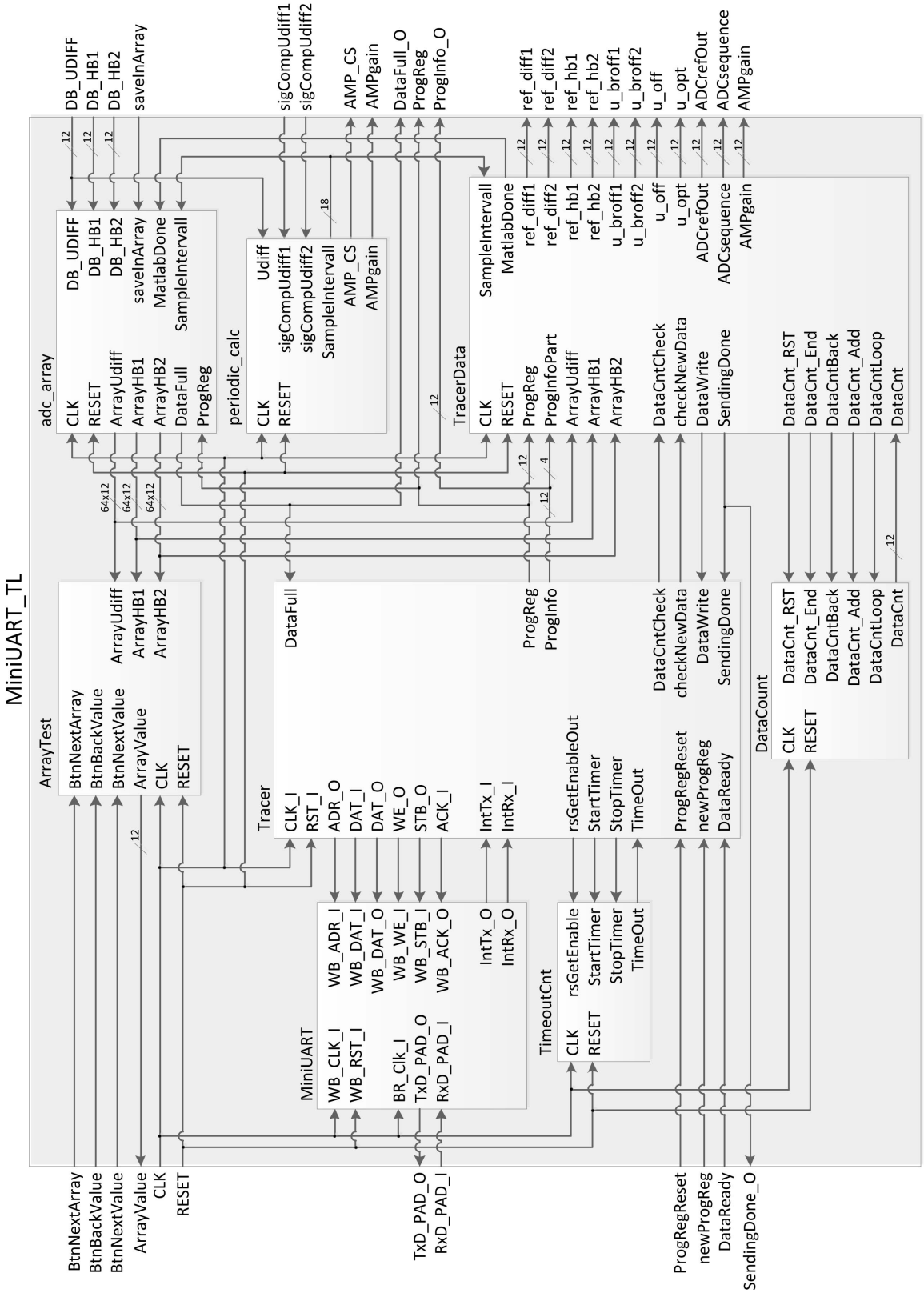


Bild B.2: *Toplevel vom MiniUART-Modul*

B.1.2 Zustandsausgabe

Bei dieser Tabelle B.1 handelt es sich um die Zustandsausgabe der Zustände des Steuerpfades von dem Toplevel *ada_conv_modsys_TL*. Diese Tabelle gibt an, welcher Zustand welche codierten Wert besitzt. Diese können dann auf dem Logicanalyzer angezeigt werden, um zum einen den Zustandsautomaten zu überprüfen und zum anderen, falls Fehler auftauchen, diese zu lokalisieren.

Nr.	Zustand	binär	hex
1	sChangeWork	000000	00
	sChangeWork_s0		
2	sConfig_s0	000001	01
	sConfig_s1		
3	sReadDone	000010	02
4	sADCwrPOR_s0	000011	03
5	sADCwrPOR_s1	000100	04
6	sADCwrPOR_s2	000101	05
7	sADCwrPOR_s3	000110	06
8	sADCwrDAC_s0	000111	07
9	sADCwrDAC_s1	001000	08
10	sADCwrDAC_s2	001001	09
11	sADCwrDAC_s3	001010	0A
12	sADCwrDAC_s4	001011	0B
13	sADCwrDAC_s5	001100	0C
14	sADCwrDAC_s6	001101	0D
15	sADCwrDAC_s7	001110	0E
16	sADCwrDAC_s8	001111	0F
17	sADCwrSEQ_s0	010000	10
18	sADCwrSEQ_s1	010001	11
19	sADCwrSEQ_s2	010010	12
20	sADCwrSEQ_s3	010011	13
21	sADCwrSEQ_s4	010100	14
22	sADCwrSEQ_s5	010101	15
23	sADCwrSEQ_s6	010110	16
24	sADCwrSEQ_s7	010111	17
25	sDACwrDAC_s0	011000	18
26	sDACwrDAC_s1	011001	19
27	sDACwrDAC_s2	011010	1A
28	sDACwrDAC_s3	011011	1B
29	sDACwrDAC_s4	011100	1C
30	sADCrdSEQ_s0	011101	1D

31	sADCrdSEQ_s1	011110	1E
32	sADCrdSEQ_s2	011111	1F
33	sADCrdSEQ_s3	100000	20
34	sADCrdSEQ_s4	100001	21
35	sADCrdSEQ_s5	100010	22
36	sADCrdSEQ_s6	100011	23
37	sADCrdSEQ_s7	100100	24
38	sADCcheckSEQ_s0	100101	25
39	sADCcheckSEQ_s1	100110	26
40	sADCcheckSEQ_s2	100111	27
41	sADCrdDAC_s0	101000	28
42	sADCrdDAC_s1	101001	29
43	sADCrdDAC_s2	101010	2A
44	sADCrdDAC_s3	101011	2B
45	sADCrdDAC_s4	101100	2C
46	sADCrdDAC_s5	101101	2D
47	sADCrdDAC_s6	101110	2E
48	sDACrdDAC_s0	101111	2F
49	sDACrdDAC_s1	110000	30
50	sDACrdDAC_s2	110001	31
51	sDACrdDAC_s3	110010	32
52	sADCrdBUSY_s0	110011	33
53	sADCrdBUSY_s1	110100	34
54	sADCrdBUSY_s2	110101	35
55	sADCrdBUSY_s3	110110	36
56	sADCrdBUSY_s4	110111	37
57	sADCrdCONV_s0	111000	38
58	sADCrdCONV_s1	111001	39
59	sADCrdCONV_s2	111010	3A
60	sADCrdCONV_s3	111011	3B
61	sADCrdCONV_s4	111100	3C
62	sADCrdCONV_s5	111101	3D
63	sADCrdCONV_s6	111110	3E
64	sADCrdCONV_s7	111111	3F

Tabelle B.1: *Zustandscodierung*

B.2 Software - Matlab

In der Tabelle B.2 werden die in Matlab empfangenen Daten dargestellt in den verschiedenen Formaten. Die Spalte *hex* ist der Wert, der vom FPGA übermittelt wird, aber in Matlab im Format *decimal* angezeigt wird. Die Spalte *Volt* ist das Format, das grafisch in dem Plot dargestellt wird. Diese drei Formate sind für alle drei Eingangssignale *UDIFF*, *HB1* und *HB2* aufgelistet.

	UDIFF			HB1			HB2		
Nr.	Volt	decimal	hex	Volt	decimal	hex	Volt	decimal	hex
1	1,41235	1157	485	1,16211	952	3B8	0,87891	720	2D0
2	1,46606	1201	4B1	1,16089	951	3B7	0,85083	697	2B9
3	1,50635	1234	4D2	1,16699	956	3BC	0,83862	687	2AF
4	1,56738	1284	504	1,25854	1031	407	0,90210	739	2E3
5	1,60645	1316	524	1,28540	1053	41D	0,92285	756	2F4
6	1,61621	1324	52C	1,27075	1041	411	0,86182	706	2C2
7	1,62720	1333	535	1,13770	932	3A4	0,75195	616	268
8	1,67725	1374	55E	1,25977	1032	408	0,83984	688	2B0
9	1,66992	1368	558	1,15356	945	3B1	0,74951	614	266
10	1,68457	1380	564	1,22192	1001	3E9	0,79834	654	28E
11	1,68945	1384	568	1,19263	977	3D1	0,75806	621	26D
12	1,69678	1390	56E	1,17798	965	3C5	0,75806	621	26D
13	1,71143	1402	57A	1,31836	1080	438	0,89111	730	2DA
14	1,68091	1377	561	1,25977	1032	408	0,86060	705	2C1
15	1,66992	1368	558	1,13647	931	3A3	0,72266	592	250
16	1,67969	1376	560	1,29639	1062	426	0,87158	714	2CA
17	1,62842	1334	536	1,12549	922	39A	0,73364	601	259
18	1,62231	1329	531	1,22314	1002	3EA	0,84717	694	2B6
19	1,60767	1317	525	1,30127	1066	42A	0,93140	763	2FB
20	1,57227	1288	508	1,15112	943	3AF	0,79712	653	28D
21	1,53076	1254	4E6	1,17920	966	3C6	0,84351	691	2B3
22	1,47705	1210	4BA	1,20605	988	3DC	0,89966	737	2E1
23	1,44287	1182	49E	1,15479	946	3B2	0,86182	706	2C2
24	1,40991	1155	483	1,12305	920	398	0,84351	691	2B3
25	1,33789	1096	448	1,12427	921	399	0,87769	719	2CF
26	1,30249	1067	42B	1,09985	901	385	0,87769	719	2CF
27	1,24146	1017	3F9	1,10840	908	38C	0,91309	748	2EC
28	1,19751	981	3D5	1,09009	893	37D	0,91431	749	2ED
29	1,15112	943	3AF	1,07666	882	372	0,92651	759	2F7
30	1,09497	897	381	1,07178	878	36E	0,95581	783	30F
31	1,03760	850	352	0,98511	807	327	0,88623	726	2D6
32	0,98022	803	323	1,00708	825	339	0,93628	767	2FF

33	0,93628	767	2FF	1,01318	830	33E	0,96802	793	319
34	0,89844	736	2E0	1,02295	838	346	1,00220	821	335
35	0,84717	694	2B6	0,93994	770	302	0,93018	762	2FA
36	0,81909	671	29F	0,98389	806	326	1,00220	821	335
37	0,78247	641	281	0,96069	787	313	0,99121	812	32C
38	0,76782	629	275	0,96436	790	316	1,00464	823	337
39	0,72876	597	255	0,98145	804	324	1,04248	854	356
40	0,70313	576	240	0,91309	748	2EC	0,97412	798	31E
41	0,69336	568	238	0,95215	780	30C	1,02539	840	348
42	0,66040	541	21D	0,86548	709	2C5	0,95947	786	312
43	0,67505	553	229	0,95459	782	30E	1,04126	853	355
44	0,65674	538	21A	0,93872	769	301	1,03149	845	34D
45	0,67139	550	226	0,94849	777	309	1,03516	848	350
46	0,66162	542	21E	0,95215	780	30C	1,05225	862	35E
47	0,67871	556	22C	0,96313	789	315	1,04980	860	35C
48	0,68604	562	232	0,93872	769	301	1,02295	838	346
49	0,70313	576	240	0,95093	779	30B	1,02173	837	345
50	0,73486	602	25A	0,95825	785	311	1,01074	828	33C
51	0,77148	632	278	0,98022	803	323	1,02173	837	345
52	0,81421	667	29B	0,99976	819	333	1,01685	833	341
53	0,83862	687	2AF	1,06323	871	367	1,07178	878	36E
54	0,87208	715	2CB	1,01563	832	340	1,00586	824	338
55	0,93018	762	2FA	0,99731	817	331	0,95581	783	30F
56	0,94971	778	30A	0,94360	773	305	0,89600	734	2DE
57	1,01074	828	33C	1,08276	887	377	0,98511	807	327
58	1,06812	875	36B	1,03271	846	34E	0,91919	753	2F1
59	1,11328	912	390	1,04370	855	357	0,91187	747	2EB
60	1,16089	951	3B7	1,07544	881	371	0,92041	754	2F2
61	1,21094	992	3E0	1,09619	898	382	0,91431	749	2ED
62	1,27563	1045	415	1,08887	892	37C	0,87769	719	2CF
63	1,31470	1077	435	1,12671	923	39B	0,89233	731	2DB
64	1,36475	1118	45E	1,16333	953	3B9	0,89600	734	2DE

Tabelle B.2: *Die Umwandlung der drei Eingangssignals UDIFF, HB1 und HB2 in den drei Formaten: Spannung in Volt, Decimal, Hexadecimal*

C Quellcode

In der Tabelle C.1 sind alle VHDL-Codes aufgelistet.

Name	Link zum Quellcode
Toplevel_TL	C.1.1
Toplevel_ada_conv_modsys	C.1.2
Toplevel_MiniUART	C.1.3
ADC-Platine: Steuerpfad	C.1.4
ADC-Platine: Datenpfad	C.1.5
7-Segement-Anzeige: Steuerung	C.1.6
7-Segment-Anzeige: Anzeige	C.1.7
7-Segment-Anzeige: Multiplexer	C.1.8
MiniUART	C.1.9
Rxunit	C.1.10
Txunit	C.1.11
utils	C.1.12
Steuerpfad für MiniUART	C.1.13
Datenpfad für MiniUART	C.1.14
Zähler für MiniUART	C.1.15
Speicherung der drei Signale in Arrays	C.1.16
Ausgabe der drei Arrays	C.1.17
Timeout für MiniUART	C.1.18
Berechnung der Periodendauer und Verstärkungsstufe	C.1.19
UCF-Datei für das NEXYS2-Board	C.1.20
ADC-Testbench	C.1.21
DAC-Testbench	C.1.22
Signalerzeugung für Simulation	C.1.23
Toplevel-Testbench	C.1.24
Signal in Abhängigkeit von der Verstärkungsstufe	C.1.25
.do-Datei für die Simulation	C.1.26

Tabelle C.1: *Liste des Quellcodes*

C.1 VHDL

C.1.1 Toplevel_TL

```
1  ------------------------------------------------------------------
2  --  Toplevel for testing ada_conv_modsys-board
3  --
4  --  Author: A. Arvidsson
5  ------------------------------------------------------------------
6
7  --<makepackage>
8  --  <component = Toplevel_TL>
9  --    <name = arvidsson>
10 --    <project = ada_conv_modsys>
11 --  </component>
12 --</makepackage>
13
14 library unisim;
15   use unisim.vcomponents.all;
16
17 library ieee;
18   use ieee.std_logic_1164.all;
19   use ieee.numeric_std.all;
20
21 library work;
22   use work.global_pkg.all;
23   use work.CLK_DIV_pkg.all;
24   use work.ada_conv_modsys_TL_pkg.all;
25   use work.MiniUART_TL_pkg.all;
26   use work.debounce_pkg.all;
27
28 entity Toplevel_TL is
29   port (
30     SYS_CLK_S      : in std_logic; -- Systemtakt
31     Btn0, Btn1, Btn2  : in std_logic; -- allgemeiner Resets
32     RESET          : in std_logic;
33
34     sw  : in bit_vector(3 downto 0);
35
36     sigCompUdiff1 : in bit;
37     sigCompUdiff2 : in bit;
38
39     --- ADC ---
40     ADC_BUSY              : in bit;
41     DB                    : inout std_logic_vector(11 downto 0);
42     ADC_CSn               : out bit;
43     ADC_WRn               : out bit;
44     ADC_RDn               : out bit;
45     ADC_CONVSTn           : out std_logic;
46     NEW_DATA_O            : out bit_vector(1 downto 0); -- new data is available
47     DB_UDIFF_O            : out std_logic_vector(11 downto 0);  -- ADC-Conv_Ausgabe
48     DB_HB1_O, DB_HB2_O    : out std_logic_vector(11 downto 0);  -- ADC-Conv_Ausgabe
49     ADC_CLK               : out std_logic;  -- ADC-Takt
50     ---------
51     --- DAC ---
52     DAC_CSn          : out bit;
53     DAC_WRn          : out bit;
54     DAC_RDn          : out bit;
55     DAC_LDACn        : out bit;
56     DAC_BUF          : out bit;
```

```
57      DAC_GAIN      : out bit;
58      DAC_A           : out bit_vector(2 downto 0);
59      ------------
60      -- Vorverstärkermodul --
61      AMP_CS     : out bit;
62      AMPgain_O : out std_logic_vector(2 downto 0);
63      ---------------------------
64      ---- MiniUART ----
65      TxD_PAD_O : out std_logic;  -- Tx RS232 Line
66      RxD_PAD_I : in  std_logic;  -- Rx RS232 Line
67      TxD : out std_logic;  -- Tx RS232 Line
68      RxD : out std_logic;  -- Rx RS232 Line
69      ------------------
70      ---- 7-Segment ----
71      seg : out bit_vector(6 downto 0);
72      an  : out bit_vector(3 downto 0);
73      dp  : out bit;
74      ------------------
75      statePart : out bit_vector(5 downto 0)
76    );
77  end Toplevel_TL;


80  architecture behaviour of Toplevel_TL is
81  signal CLK, CLK_O     : std_logic;

83  signal BtnNextArray  : bit;
84  signal BtnBackValue  : bit;
85  signal BtnNextValue  : bit;
86  signal ArrayValue : std_logic_vector(11 downto 0);

88  -- MiniUART
89  signal NEW_DATA  : bit_vector(1 downto 0);
90  signal DB_UDIFF  : std_logic_vector(11 downto 0);
91  signal DB_HB1    : std_logic_vector(11 downto 0);
92  signal DB_HB2    : std_logic_vector(11 downto 0);
93  signal ProgReg   : std_logic_vector(3 downto 0);
94  signal ProgInfo  : std_logic_vector(11 downto 0);
95  signal newProgReg : bit;
96  signal ProgRegReset : bit;

98  signal TxD_PAD : std_logic;
99  signal DataFull : bit;
100 signal saveInArray : bit;
101 signal ADCsequence : std_logic_vector(11 downto 0);

103 signal ref_diff1 : std_logic_vector(11 downto 0);
104 signal ref_diff2 : std_logic_vector(11 downto 0);
105 signal ref_hb1 : std_logic_vector(11 downto 0);
106 signal ref_hb2 : std_logic_vector(11 downto 0);
107 signal u_broff1 : std_logic_vector(11 downto 0);
108 signal u_broff2 :std_logic_vector(11 downto 0);
109 signal u_off : std_logic_vector(11 downto 0);
110 signal u_opt : std_logic_vector(11 downto 0);

112 signal ADCrefOut : std_logic_vector(11 downto 0);
113 signal DataReady : bit;
114 signal SendingDone : bit;

116 begin

118   RxD <= RxD_PAD_I;
119   TxD <= TxD_PAD;
```

```
120   TxD_PAD_O <= TxD_PAD;
121   DB_UDIFF_O <= DB_UDIFF;
122   DB_HB1_O   <= DB_HB1;
123   DB_HB2_O   <= DB_HB2;
124   NEW_DATA_O <= NEW_DATA;

126   I_DIV : CLK_DIV
127     port map (
128       SYS_CLK_S => SYS_CLK_S,
129       RESET     => RESET,
130       CLK_OUT   => CLK_O
131     );

133   BUFG_CLK_OUT : BUFG
134     port map (O => CLK,
135               I => CLK_O);

137   I_debounce : debounce
138     port map (
139       CLK             => CLK,
140       RESET           => RESET,
141       sw              => sw(0),
142       Btn0            => Btn0,
143       Btn1            => Btn1,
144       Btn2            => Btn2,
145       BtnNextArray    => BtnNextArray,
146       BtnBackValue    => BtnBackValue,
147       BtnNextValue    => BtnNextValue
148     );

150   I_ada : ada_conv_modsys_TL
151     port map (
152       CLK    => CLK,
153       RESET => RESET,

155       ArrayValue       => ArrayValue,

157       sw  => sw(3 downto 1),

159       -- ADC --
160       ADC_BUSY           => ADC_BUSY,
161       DB                 => DB,
162       ADC_CSn            => ADC_CSn,
163       ADC_WRn            => ADC_WRn,
164       ADC_RDn            => ADC_RDn,
165       ADC_CONVSTn        => ADC_CONVSTn,
166       NEW_DATA_O     => NEW_DATA,
167       DB_UDIFF_O     => DB_UDIFF,
168       DB_HB1_O       => DB_HB1,
169       DB_HB2_O       => DB_HB2,
170       ---------
171       -- DAC --
172       DAC_CSn            => DAC_CSn,
173       DAC_WRn            => DAC_WRn,
174       DAC_LDACn          => DAC_LDACn,
175       DAC_RDn            => DAC_RDn,
176       DAC_BUF            => DAC_BUF,
177       DAC_GAIN           => DAC_GAIN,
178       DAC_A              => DAC_A,
179       ---------
180       -- Outputs --
181         -- DAC --
182       ref_diff1_O      => ref_diff1,
```

```
183         ref_diff2_O      => ref_diff2,
184         ref_hb1          => ref_hb1,
185         ref_hb2          => ref_hb2,
186         u_broff1         => u_broff1,
187         u_broff2         => u_broff2,
188         u_off            => u_off,
189         u_opt            => u_opt,
190           -- ADC --
191         ADCrefOut_O      => ADCrefOut,
192         ADCsequence_O    => ADCsequence,

194         ProgReg    => ProgReg,
195         ProgInfo   => ProgInfo,
196         newProgReg => newProgReg,

198         ProgRegReset => ProgRegReset,

200         DataFull => DataFull,
201         saveInArray => saveInArray,
202         DataReady => DataReady,
203         SendingDone => SendingDone,

205         ---- 7-Segment ----
206         seg => seg,
207         an  => an,
208         dp  => dp,
209         ---------------------
210         statePart => statePart
211       );

213     I_UART : MiniUART_TL
214       port map (
215         CLK        => CLK,
216         RESET      => RESET,

218         BtnNextArray  => BtnNextArray,
219         BtnBackVAlue  => BtnBackValue,
220         BtnNextValue  => BtnNextValue,
221         ArrayValue    => ArrayValue,

223         DB_UDIFF  => DB_UDIFF,
224         DB_HB1    => DB_HB1,
225         DB_HB2    => DB_HB2,

227         sigCompUdiff1 => sigCompUdiff1,
228         sigCompUdiff2 => sigCompUdiff2,

230         --- Inputs ---
231           -- DAC --
232         ref_diff1        => ref_diff1,
233         ref_diff2        => ref_diff2,
234         ref_hb1          => ref_hb1,
235         ref_hb2          => ref_hb2,
236         u_broff1         => u_broff1,
237         u_broff2         => u_broff2,
238         u_off            => u_off,
239         u_opt            => u_opt,
240           -- ADC --
241         ADCrefOut        => ADCrefOut,
242         ADCsequence      => ADCsequence,
243         AMPgain_O        => AMPgain_O,
244         AMP_CS           => AMP_CS,
```

```
246         ProgReg      => ProgReg,
247         ProgInfo_O   => ProgInfo,
248         newProgReg   => newProgReg,

250         ProgRegReset => ProgRegReset,

252         DataFull_O => DataFull,
253         saveInArray => saveInArray,
254         DataReady => DataReady,
255         SendingDone_O => SendingDone,

257         ---- MiniUART ----
258         TxD_PAD_O => TxD_PAD,
259         RxD_PAD_I => RxD_PAD_I
260       );

262     OBUF_ADC_CLK : OBUF
263       port map (O => ADC_CLK,
264                 I => CLK);

266 end behaviour;
```

C.1.2 Toplevel_ada_conv_modsys

```
1  ----------------------------------------------------------------
2  --  Toplevel for testing ada_conv_modsys-board
3  --
4  --  Author: A. Arvidsson
5  ----------------------------------------------------------------

7  -- <makepackage>
8  --      <component = ada_conv_modsys_TL>
9  --      <name = arvidsson>
10 --      <project = ada_conv_modsys_master>
11 --       </component>
12 -- </makepackage>

14 library ieee;
15   use ieee.std_logic_1164.all;
16   use ieee.numeric_std.all;

18 Library UNISIM;
19   use UNISIM.vcomponents.all;

21 library work;
22   use work.global_pkg.all;
23   use work.CONTR_FSM_pkg.all;
24   use work.Datapath_pkg.all;
25   -- use work.CalcGain_pkg.all;
26   use work.seven_segment_pkg.all;
27   use work.sev_seg_ctr_pkg.all;
28   use work.seven_seg_mux_pkg.all;


31 entity ada_conv_modsys_TL is
32   port (
33     CLK   : in std_logic; -- Takt
34     RESET : in std_logic; -- allgemeiner Reset
```

```
36      ArrayValue      : in std_logic_vector(11 downto 0);

38      sw  : in bit_vector(2 downto 0);

40      -- ADC --
41      ADC_BUSY    : in bit;
42      DB          : inout std_logic_vector(11 downto 0);
43      ADC_CSn     : out bit;
44      ADC_WRn     : out bit;
45      ADC_RDn     : out bit;
46      ADC_CONVSTn : out std_logic;
47      NEW_DATA_O  : out bit_vector(1 downto 0); -- new data is available
48      DB_UDIFF_O  : out std_logic_vector(11 downto 0);  -- ADC-Conv_Ausgabe
49      DB_HB1_O    : out std_logic_vector(11 downto 0);  -- ADC-Conv_Ausgabe
50      DB_HB2_O    : out std_logic_vector(11 downto 0);  -- ADC-Conv_Ausgabe
51      --------

53      -- DAC --
54      DAC_CSn    : out bit;
55      DAC_WRn    : out bit;
56      DAC_LDACn  : out bit;
57      DAC_RDn    : out bit;
58      DAC_BUF    : out bit;
59      DAC_GAIN   : out bit;
60      DAC_A      : out bit_vector(2 downto 0);
61      --------

63      ---------- Outputs ----------
64        -- DAC --
65      ref_diff1_O : out std_logic_vector(11 downto 0);
66      ref_diff2_O : out std_logic_vector(11 downto 0);
67      ref_hb1     : out std_logic_vector(11 downto 0);
68      ref_hb2     : out std_logic_vector(11 downto 0);
69      u_broff1    : out std_logic_vector(11 downto 0);
70      u_broff2    : out std_logic_vector(11 downto 0);
71      u_off       : out std_logic_vector(11 downto 0);
72      u_opt       : out std_logic_vector(11 downto 0);
73        -- ADC --
74      ADCrefOut_O    : out std_logic_vector(11 downto 0);
75      ADCsequence_O : out std_logic_vector(11 downto 0);
76      -- AMPgain        : out std_logic_vector( 2 downto 0);
77      -- AMP_CS         : out bit;

79      ProgReg     : in std_logic_vector(3 downto 0);
80      ProgInfo    : in std_logic_vector(11 downto 0);
81      newProgReg  : in bit;

83      ProgRegReset : out bit;

85      DataFull    : in bit;
86      saveInArray : out bit;
87      DataReady   : out bit;
88      SendingDone : in bit;

90      -- sigCompUdiff1 : in bit;
91      -- sigCompUdiff2 : in bit;

93      ---- 7-Segment ----
94      seg : out bit_vector(6 downto 0);
95      an  : out bit_vector(3 downto 0);
96      dp  : out bit;
97      ------------------
98      statePart : out bit_vector(5 downto 0)
```

```
99    );
100 end ada_conv_modsys_TL;

102 architecture behaviour of ada_conv_modsys_TL is
103 signal WR_N_EN          : bit;
104 signal DAC_A_I          : bit_vector(2 downto 0);  -- intern
105 signal PART             : bit;
106 signal CONV_CH          : bit;
107 signal DB_UDIFF         : std_logic_vector(11 downto 0);
108 signal DB_HB1           : std_logic_vector(11 downto 0);
109 signal DB_HB2           : std_logic_vector(11 downto 0);
110 -- ADC --
111 signal PowerOnReset     : bit;
112 signal ADCgotoDACreg    : bit;
113 signal ADCwrDACreg      : bit;
114 signal ADCgotoDACread   : bit;
115 signal ADCrdDACreg      : bit;
116 signal ADCgotoSEQreg    : bit;
117 signal ADCwrSEQreg      : bit;
118 signal ADCgotoSEQread   : bit;
119 signal ADCrdSEQreg      : bit;
120 signal ADCwrConf        : bit;
121 signal CheckSEQ         : bit;
122 signal newTestSEQ       : bit;
123 -- DAC --
124 signal DACwrDACreg      : bit;
125 signal DACrdDACreg      : bit;

127 signal ReadNow          : bit;
128 signal NEW_DATA         : bit_vector(1 downto 0);

130 signal DB_PART          : std_logic_vector(3 downto 0);
131 signal DATA             : std_logic_vector(11 downto 0);

133 signal ADCrefOut        : std_logic_vector(11 downto 0);
134 signal ADCsequence      : std_logic_vector(11 downto 0);
135 signal OutRDspecReg     : std_logic_vector( 3 downto 0);
136 signal ref_diff1        : std_logic_vector(11 downto 0);
137 signal ref_diff2        : std_logic_vector(11 downto 0);

139 -- component BUFG
140    -- port (O : out std_logic;
141          -- I : in std_logic);
142 -- end component;

144 begin

146   -- outgoing
147   DB_UDIFF_O     <= DB_UDIFF;
148   DB_HB1_O       <= DB_HB1;
149   DB_HB2_O       <= DB_HB2;
150   NEW_DATA_O     <= NEW_DATA;
151   DAC_A          <= DAC_A_I;
152   ADCsequence_O  <= ADCsequence;
153   ADCrefOut_O    <= ADCrefOut;
154   ref_diff1_O    <= ref_diff1;
155   ref_diff2_O    <= ref_diff2;

157   I_FSM : CONTR_FSM
158     port map (
159       CLK        => CLK,
160       RESET      => RESET,
161       ADC_BUSY   => ADC_BUSY,
```

```
163         ProgReg         => ProgReg,
164         ProgInfoPart    => OutRDspecReg, -- only need the last 4 bit... if ProgReg = x"D"

166         -- ADC --
167         ADC_CONVSTn     => ADC_CONVSTn,
168         WR_N_EN         => WR_N_EN,
169         PART            => PART,
170         CONV_CH         => CONV_CH,
171         ADC_CSn         => ADC_CSn,
172         ADC_WRn         => ADC_WRn,
173         ADC_RDn         => ADC_RDn,
174         ReadNow         => ReadNow,
175         PowerOnReset    => PowerOnReset,
176         ADCgotoDACreg   => ADCgotoDACreg,
177         ADCwrDACreg     => ADCwrDACreg,
178         ADCgotoDACread  => ADCgotoDACread,
179         ADCrdDACreg     => ADCrdDACreg,
180         ADCgotoSEQreg   => ADCgotoSEQreg,
181         ADCwrSEQreg     => ADCwrSEQreg,
182         ADCgotoSEQread  => ADCgotoSEQread,
183         ADCrdSEQreg     => ADCrdSEQreg,
184         ADCwrConf       => ADCwrConf,
185         CheckSEQ        => CheckSEQ,
186         newTestSEQ      => newTestSEQ,
187         -- DAC --
188         DAC_CSn         => DAC_CSn,
189         DAC_WRn         => DAC_WRn,
190         DAC_LDACn       => DAC_LDACn,
191         DAC_RDn         => DAC_RDn,
192         DAC_BUF         => DAC_BUF,
193         DAC_GAIN        => DAC_GAIN,
194         DACwrDACreg     => DACwrDACreg,
195         DACrdDACreg     => DACrdDACreg,
196         DAC_A           => DAC_A_I,
197         ------
198         ProgRegReset    => ProgRegReset,
199         DataFull        => DataFull,
200         saveInArray     => saveInArray,
201         DataReady       => DataReady,
202         SendingDone     => SendingDone,
203         statePart       => statePart
204       );
205     I_DATA : DATAPATH
206       port map (
207         CLK             => CLK,
208         RESET           => RESET,

210         WR_N_EN         => WR_N_EN,
211         PART            => PART,
212         CONV_CH         => CONV_CH,
213         ReadNow         => ReadNow,
214         -- ADC --
215         PowerOnReset    => PowerOnReset,
216         ADCgotoDACreg   => ADCgotoDACreg,
217         ADCwrDACreg     => ADCwrDACreg,
218         ADCgotoDACread  => ADCgotoDACread,
219         ADCrdDACreg     => ADCrdDACreg,
220         ADCgotoSEQreg   => ADCgotoSEQreg,
221         ADCwrSEQreg     => ADCwrSEQreg,
222         ADCgotoSEQread  => ADCgotoSEQread,
223         ADCrdSEQreg     => ADCrdSEQreg,
224         ADCwrConf       => ADCwrConf,
```

```
225         CheckSEQ          => CheckSEQ,
226         newTestSEQ        => newTestSEQ,
227         -- DAC --
228         DACwrDACreg       => DACwrDACreg,
229         DACrdDACreg       => DACrdDACreg,
230         ----------
231         DAC_A               => DAC_A_I,
232         DB                => DB,
233         NEW_DATA          => NEW_DATA,
234         DB_UDIFF          => DB_UDIFF,
235         DB_HB1            => DB_HB1,
236         DB_HB2            => DB_HB2,
237         ---- Outputs ----
238           -- DAC --
239         ref_diff1         => ref_diff1,
240         ref_diff2         => ref_diff2,
241         ref_hb1           => ref_hb1,
242         ref_hb2           => ref_hb2,
243         u_broff1          => u_broff1,
244         u_broff2          => u_broff2,
245         u_off             => u_off,
246         u_opt             => u_opt,
247           -- ADC --
248         ADCrefOut         => ADCrefOut,
249         ADCsequence       => ADCsequence,
250         OutRDspecReg      => OutRDspecReg,

252         ProgReg           => ProgReg,
253         ProgInfo          => ProgInfo,
254         newProgReg        => newProgReg
255       );
256     -- I_GAIN : CalcGain
257       -- port map (
258         -- CLK             => CLK,
259         -- RESET           => RESET,
260         -- sigCompUdiff1 => sigCompUdiff1,
261         -- sigCompUdiff2 => sigCompUdiff2,
262         -- Udiff           => DB_UDIFF,
263         -- AMPgain         => AMPgain,
264         -- AMP_CS          => AMP_CS
265       -- );
266     I_sev_seg : sev_seg_ctr
267       port map (
268         CLK_SEG => CLK,
269         RESET   => RESET,
270         DATA    => DATA,
271         ANODES  => an,
272         DB_PART => DB_PART
273       );
274     I_seg : seven_segment
275       port map (
276         DB_PART => DB_PART,
277         SEG     => seg,
278         dp      => dp
279       );
280     I_seg_mux : seven_seg_mux
281       port map (
282         SWITCHES    => sw,
283         CHANNEL_0   => DB_UDIFF,
284         CHANNEL_1   => DB_HB1,
285         CHANNEL_2   => DB_HB2,
286         CHANNEL_3   => ADCsequence,
287         CHANNEL_4   => ADCrefOut,
```

```
288         CHANNEL_5   => ref_diff1,
289         CHANNEL_6   => ref_diff2,
290         CHANNEL_7   => ArrayValue,
291         CHANNEL_OUT => DATA
292       );

294 end behaviour;
```

C.1.3 Toplevel_MiniUART

```
1  ----------------------------------------------------------------
2  --  Toplevel for testing MiniUART
3  --
4  --  Author: A.Arvidsson
5  ----------------------------------------------------------------

7  -- <makepackage>
8  --      <component = MiniUART_TL>
9  --      <name = arvidsson>
10 --      <project = ada_conv_modsys_master>
11 --       </component>
12 -- </makepackage>

14 library ieee;
15   use ieee.std_logic_1164.all;
16   use ieee.numeric_std.all;

18 Library UNISIM;
19   use UNISIM.vcomponents.all;

21 library work;
22   use work.global_pkg.all;
23   use work.miniuart_pkg.all;
24   use work.Tracer_pkg.all;
25   use work.TracerData_pkg.all;
26   use work.DataCount_pkg.all;
27   use work.adc_array_pkg.all;
28   use work.ArrayTest_pkg.all;
29   use work.TimeoutCnt_pkg.all;
30   use work.period_calc_pkg.all;


33 entity MiniUART_TL is
34   port (
35     CLK        : in std_logic;
36     RESET      : in std_logic;

38     BtnNextArray  : in bit;
39     BtnBackValue  : in bit;
40     BtnNextValue  : in bit;
41     ArrayValue    : out std_logic_vector(11 downto 0);

43     DB_UDIFF   : in std_logic_vector(11 downto 0);
44     DB_HB1     : in std_logic_vector(11 downto 0);
45     DB_HB2     : in std_logic_vector(11 downto 0);

47     sigCompUdiff1 : in bit;
48     sigCompUdiff2 : in bit;
```

```
50      ------------ Inputs ------------
51        -- DAC --
52      ref_diff1    : in std_logic_vector(11 downto 0);
53      ref_diff2    : in std_logic_vector(11 downto 0);
54      ref_hb1      : in std_logic_vector(11 downto 0);
55      ref_hb2      : in std_logic_vector(11 downto 0);
56      u_broff1     : in std_logic_vector(11 downto 0);
57      u_broff2     : in std_logic_vector(11 downto 0);
58      u_off        : in std_logic_vector(11 downto 0);
59      u_opt        : in std_logic_vector(11 downto 0);
60        -- ADC --
61      ADCrefOut    : in std_logic_vector(11 downto 0);
62      ADCsequence  : in std_logic_vector(11 downto 0);
63      AMPgain_O    : out std_logic_vector( 2 downto 0);
64      AMP_CS       : out bit;

66      ProgReg      : out std_logic_vector(3 downto 0);
67      ProgInfo_O   : out std_logic_vector(11 downto 0);
68      newProgReg   : out bit;

70      ProgRegReset : in bit;

72      DataFull_O     : out bit;
73      saveInArray    : in bit;
74      DataReady      : in bit;
75      SendingDone_O  : out bit;

77      ---- MiniUART ----
78      TxD_PAD_O : out std_logic;  -- Tx RS232 Line
79      RxD_PAD_I : in  std_logic   -- Rx RS232 Line
80    );
81  end MiniUART_TL;

83  architecture behaviour of MiniUART_TL is
84  -- Tracer/Wishbone
85  signal ADR_O         : std_logic_vector(1 downto 0); -- Adress bus
86  signal DAT_I         : std_logic_vector(7 downto 0); -- DataIn Bus
87  signal DAT_O         : std_logic_vector(7 downto 0); -- DataOut Bus
88  signal WE_O          : std_logic;  -- Write Enable
89  signal STB_O         : std_logic;  -- Strobesignal
90  signal ACK_I         : std_logic;    -- Acknowledge
91  signal IntTx_O       : std_logic;  -- Transmit interrupt: indicate waiting for Byte
92  signal IntRx_O       : std_logic;  -- Receive interrupt: indicate Byte received
93  ----- extra -----
94  signal ArrayUdiff    : DataArray;
95  signal ArrayHB1      : DataArray;
96  signal ArrayHB2      : DataArray;
97  signal DataFull      : bit;
98  signal rsGetEnable   : bit;
99  signal MatlabDone    : bit;
100 signal ProgRegInt    : std_logic_vector(3 downto 0);
101 signal SendingDone   : bit;
102 signal checkNewData  : bit;
103 signal DataCntCheck  : bit;
104 signal ProgInfoPart  : std_logic_vector(3 downto 0);
105 signal DataWrite     : std_logic_vector(7 downto 0);
106 -- --- DataCount
107 signal DataCnt_RST   : bit;
108 signal DataCnt_End   : bit;
109 signal DataCntBack   : bit;
110 signal DataCnt_Add   : bit;
111 signal DataCntLoop   : bit;
112 signal DataCnt       : unsigned(3 downto 0);
```

```
114 signal StartTimer : bit;
115 signal StopTimer  : bit;
116 signal TimeOut    : bit;

118 signal AMPgain  : std_logic_vector(2 downto 0);

120 signal SampleIntervall : unsigned((SampleIntervallMax - 1) downto 0);

122 signal ProgInfo : std_logic_vector(11 downto 0);

124 begin

126   ProgReg        <= ProgRegInt;
127   DataFull_O     <= DataFull;
128   SendingDone_O  <= SendingDone;
129   ProgInfo_O     <= ProgInfo;
130   ProgInfoPart   <= ProgInfo(3 downto 0);
131   AMPgain_O      <= AMPgain;

133   I_UART : MiniUART
134     generic map (BRDIVISOR => 163) -- with BR_CLK_I = 12,5MHz = CLK => BaudRate=19200
135     port map (
136       -- Wishbone signals
137       WB_CLK_I  => CLK,
138       WB_RST_I  => RESET,
139       WB_ADR_I  => ADR_O,
140       WB_DAT_I  => DAT_O,
141       WB_DAT_O  => DAT_I,
142       WB_WE_I   => WE_O,
143       WB_STB_I  => STB_O,
144       WB_ACK_O  => ACK_I,
145       -- process signals
146       IntTx_O   => IntTx_O,
147       IntRx_O   => IntRx_O,
148       BR_Clk_I  => CLK,
149       TxD_PAD_O => TxD_PAD_O,
150       RxD_PAD_I => RxD_PAD_I
151     );

153   I_TRAC : TRACER
154     port map (
155       --- Wishbone ---
156       CLK_I     => CLK,
157       RST_I     => RESET,
158       ADR_O     => ADR_O,
159       DAT_I     => DAT_I,
160       DAT_O     => DAT_O,
161       WE_O      => WE_O,
162       STB_O     => STB_O,
163       ACK_I     => ACK_I,
164       --- non-Wishbone ---
165       IntTx_I   => IntTx_O,
166       IntRx_I   => IntRx_O,
167       --- extra ---
168       DataFull        => DataFull,
169       rsGetEnableOut  => rsGetEnable,
170       SendingDone     => SendingDone,
171       ------------------
172       StartTimer  => StartTimer,
173       StopTimer   => StopTimer,
174       TimeOut     => TimeOut,
175       ------------------
```

```
176         ProgReg         => ProgRegInt,
177         ProgInfo        => ProgInfo,
178         ProgRegReset    => ProgRegReset,
179         newProgReg      => newProgReg,
180         DataWrite       => DataWrite,
181         ----------------
182         checkNewData    => checkNewData,
183         DataCntCheck    => DataCntCheck,
184         DataReady       => DataReady
185       );
186     I_TraceData : TracerData
187       port map (
188         CLK             => CLK,
189         RESET           => RESET,
190         ProgReg         => ProgRegInt,
191         ProgInfoPart    => ProgInfoPart,
192         ArrayUdiff      => ArrayUdiff,
193         ArrayHB1        => ArrayHB1,
194         ArrayHB2        => ArrayHB2,
195         SampleIntervall => SampleIntervall,
196         ref_diff1       => ref_diff1,
197         ref_diff2       => ref_diff2,
198         ref_hb1         => ref_hb1,
199         ref_hb2         => ref_hb2,
200         u_broff1        => u_broff1,
201         u_broff2        => u_broff2,
202         u_off           => u_off,
203         u_opt           => u_opt,
204         ADCrefOut       => ADCrefOut,
205         ADCsequence     => ADCsequence,
206         AMPgain         => AMPgain,
207         DataCntCheck    => DataCntCheck,
208         checkNewData    => checkNewData,
209         DataWrite       => DataWrite,
210         MatlabDone      => MatlabDone,
211         SendingDone     => SendingDone,
212         DataCnt_RST     => DataCnt_RST,
213         DataCnt_End     => DataCnt_End,
214         DataCntBack     => DataCntBack,
215         DataCnt_Add     => DataCnt_Add,
216         DataCntLoop     => DataCntLoop,
217         DataCnt         => DataCnt
218       );
219     I_CNT : DataCount
220       port map (
221         CLK          => CLK,
222         RESET        => RESET,
223         DataCnt_RST  => DataCnt_RST,
224         DataCnt_End  => DataCnt_End,
225         DataCntBack  => DataCntBack,
226         DataCnt_Add  => DataCnt_Add,
227         DataCntLoop  => DataCntLoop,
228         DataCnt      => DataCnt
229       );
230     I_ADC_ARRAY : adc_array
231       port map (
232         CLK             => CLK,
233         RESET           => RESET,
234         DB_UDIFF        => DB_UDIFF,
235         DB_HB1          => DB_HB1,
236         DB_HB2          => DB_HB2,
237         ArrayUdiff      => ArrayUdiff,
238         ArrayHB1        => ArrayHB1,
```

```
239         ArrayHB2          => ArrayHB2,
240         DataFull          => DataFull,
241         saveInArray       => saveInArray,
242         MatlabDone        => MatlabDone,
243         ProgReg           => ProgRegInt,
244         SampleIntervall => SampleIntervall
245       );
246     I_ArrayTest : ArrayTest
247       port map (
248         CLK             => CLK,
249         RESET           => RESET,
250         BtnNextArray    => BtnNextArray,
251         BtnBackValue    => BtnBackValue,
252         BtnNextValue    => BtnNextValue,
253         ArrayUdiff      => ArrayUdiff,
254         ArrayHB1        => ArrayHB1,
255         ArrayHB2        => ArrayHB2,
256         ArrayValue      => ArrayValue
257       );
258     I_TimeOut : TimeoutCnt
259       port map (
260         CLK           => CLK,
261         RESET         => RESET,
262         rsGetEnable => rsGetEnable,
263         StartTimer    => StartTimer,
264         StopTimer     => StopTimer,
265         TimeOut       => TimeOut
266       );
267     I_Period : period_calc
268       port map (
269         CLK               => CLK,
270         RESET             => RESET,
271         sigCompUdiff1     => sigCompUdiff1,
272         sigCompUdiff2     => sigCompUdiff2,
273         SampleIntervall => SampleIntervall,
274         Udiff             => DB_UDIFF,
275         AMPgain           => AMPgain,
276         AMP_CS            => AMP_CS
277       );

279 end behaviour;
```

C.1.4 ADC-Platine: Steuerpfad

```
1  ----------------------------------------------------------------------
2  --  Control-FSM for signal-control
3  --
4  --  Author: A. Arvidsson
5  ----------------------------------------------------------------------

7  -- <makepackage>
8  --      <component = CONTR_FSM>
9  --      <name = arvidsson>
10 --      <project = ada_conv_modsys>
11 --       </component>
12 -- </makepackage>

14 library ieee;
15   use ieee.std_logic_1164.all;
```

```
16  use ieee.numeric_std.all;

18  library work;
19  use work.global_pkg.all;

21  entity CONTR_FSM is
22  port (
23  CLK             : in std_logic;
24  RESET           : in std_logic;
25  -- ADC --
26  ADC_BUSY        : in bit;
27  --
28  ProgReg         : in std_logic_vector(3 downto 0);
29  ProgInfoPart    : in std_logic_vector(3 downto 0);
30  --
31  ADC_CONVSTn     : out std_logic; -- Start-Bit
32  WR_N_EN         : out bit; -- Write-enable
33  PART            : out bit; -- Config- / Sequencer-Register-Auswahl
34  CONV_CH         : out bit; -- channel which is conversioned
35  ADC_CSn         : out bit;
36  ADC_WRn         : out bit;
37  ADC_RDn         : out bit;
38  ReadNow         : out bit;
39  PowerOnReset    : out bit;
40  ADCgotoDACreg   : out bit;
41  ADCwrDACreg     : out bit;
42  ADCgotoDACread  : out bit;
43  ADCrdDACreg     : out bit;
44  ADCgotoSEQreg   : out bit;
45  ADCwrSEQreg     : out bit;
46  ADCgotoSEQread  : out bit;
47  ADCrdSEQreg     : out bit;
48  ADCwrConf       : out bit;
49  CheckSEQ        : out bit; -- check sequence-channel
50  newTestSEQ      : in bit;
51  ------------
52  -- DAC --
53  DAC_CSn         : out bit;
54  DAC_WRn         : out bit;
55  DAC_LDACn       : out bit;
56  DAC_RDn         : out bit;
57  DAC_BUF         : out bit;
58  DAC_GAIN        : out bit;
59  DACwrDACreg     : out bit;
60  DACrdDACreg     : out bit;
61  DAC_A           : out bit_vector(2 downto 0);
62  ------------
63  ProgRegReset    : out bit;
64  DataFull        : in bit;
65  saveInArray     : out bit;
66  DataReady       : out bit;
67  SendingDone     : in bit;
68  statePart       : out bit_vector(5 downto 0)
69  );
70  end CONTR_FSM;

72  architecture FSM of CONTR_FSM is
73  type STATE_TYPE is (-- init --
74                      sADCwrPOR_s0,sADCwrPOR_s1,sADCwrPOR_s2,sADCwrPOR_s3,  -- PowerOnReset
                            directly after RESET or PowerOn
75                      -- ADC --
76                      sConfig_s0,sConfig_s1,
```

```
77                      sADCwrDAC_s0,sADCwrDAC_s1,sADCwrDAC_s2,sADCwrDAC_s3,sADCwrDAC_s4,
                            sADCwrDAC_s5,sADCwrDAC_s6,sADCwrDAC_s7,sADCwrDAC_s8,    -- DAC of
                            ADC
78                      sADCwrSEQ_s0,sADCwrSEQ_s1,sADCwrSEQ_s2,sADCwrSEQ_s3,sADCwrSEQ_s4,
                            sADCwrSEQ_s5,sADCwrSEQ_s6,sADCwrSEQ_s7,                 -- write
                            sequence-register of ADC
79                      sADCcheckSEQ_s0,sADCcheckSEQ_s1,sADCcheckSEQ_s2,
                                                        -- to check next converted
                            channels
80                      sADCrdSEQ_s0,sADCrdSEQ_s1,sADCrdSEQ_s2,sADCrdSEQ_s3,sADCrdSEQ_s4,
                            sADCrdSEQ_s5,sADCrdSEQ_s6,sADCrdSEQ_s7,     -- read sequence-
                            register of ADC
81                      sADCrdDAC_s0,sADCrdDAC_s1,sADCrdDAC_s2,sADCrdDAC_s3,sADCrdDAC_s4,
                            sADCrdDAC_s5,sADCrdDAC_s6,                  -- read DAC-register of
                             ADC
82                      sADCrdBUSY_s0,sADCrdBUSY_s1,sADCrdBUSY_s2,sADCrdBUSY_s3,sADCrdBUSY_s4,
                                    -- call BUSY-flag
83                      sADCrdCONV_s0,sADCrdCONV_s1,sADCrdCONV_s2,sADCrdCONV_s3,sADCrdCONV_s4,
                            sADCrdCONV_s5,sADCrdCONV_s6,sADCrdCONV_s7,  -- read conversions of
                             ADC
84                      -- DAC --
85                      sDACwrDAC_s0,sDACwrDAC_s1,sDACwrDAC_s2,sDACwrDAC_s3,sDACwrDAC_s4,  --
                            write DAC-register of DAC
86                      sDACrdDAC_s0,sDACrdDAC_s1,sDACrdDAC_s2,sDACrdDAC_s3,               --
                            read DAC-register of DAC
87                      -- center --
88                      sChangeWork,sChangeWork_s0,             -- sChangeWork
89                      sReadDone                               -- ReadDone
90                      );
91  signal STATE : STATE_TYPE;
92  signal CONV_CHANNEL : bit;
93  signal Reading  : bit;
94  signal firstRun : bit;
95  signal SeqOK    : bit;
96  signal saveInArrayX : bit;
97  signal newCheckSEQ  : bit;
98  signal initRun : bit;
99  signal DAC_A_X : bit_vector(2 downto 0);
100 signal oneWaitCycle : bit;
101 signal waitTwoCycles : unsigned(1 downto 0);
102 signal readAllRegister : bit;
103 signal readOneRegister : bit;
104 signal readAllDone : bit;
105 alias CallRegister : std_logic_vector(3 downto 0) is ProgInfoPart;

107 begin

109   READ_CONV : process(RESET,CLK,CONV_CHANNEL,DAC_A_X,saveInArrayX)
110   begin
111     if RESET = '1' then
112       STATE <= sADCwrPOR_s0 after 9 ns;
113       ---------- ADC ------------------------------
114       ADC_CSn <= '1' after 9 ns;  -- defaults
115       ADC_WRn <= '1' after 9 ns;
116       WR_N_EN <= '0' after 9 ns;
117       ADC_RDn <= '1' after 9 ns;
118       PART <= '0' after 9 ns;
119       CONV_CHANNEL <= '0' after 9 ns;
120       ---------------------------------------
121       ReadNow <= '0' after 9 ns;
122       ADC_CONVSTn <= '1' after 9 ns;
123       PowerOnReset <= '0' after 9 ns;
124       ADCgotoDACreg <= '0' after 9 ns;
```

```
125         ADCwrDACreg <= '0' after 9 ns;
126         ADCgotoDACread <= '0' after 9 ns;
127         ADCrdDACreg <= '0' after 9 ns;
128         ADCgotoSEQreg <= '0' after 9 ns;
129         ADCwrSEQreg <= '0' after 9 ns;
130         ADCgotoSEQread <= '0' after 9 ns;
131         ADCrdSEQreg <= '0' after 9 ns;
132         ADCwrConf <= '0' after 9 ns;

134         CheckSEQ <= '0' after 9 ns;
135         newCheckSEQ <= '0' after 9 ns;
136         ------------------------------------------
137         ------------- DAC -------------
138         DAC_CSn <= '1' after 9 ns;
139         DAC_WRn <= '1' after 9 ns;
140         DAC_RDn <= '1' after 9 ns;
141         DAC_LDACn <= '1' after 9 ns;
142         DAC_BUF <= '0' after 9 ns;
143         DAC_GAIN <= '1' after 9 ns;
144         DACwrDACreg <= '0' after 9 ns;
145         DACrdDACreg <= '0' after 9 ns;
146         DAC_A_X <= "000" after 9 ns;
147         ---------------------------------

149         Reading <= '0' after 9 ns;
150         initRun <= '1' after 9 ns;
151         firstRun <= '1' after 9 ns;
152         ProgRegReset <= '0' after 9 ns;

154         waitTwoCycles <= (others => '0') after 9 ns;
155         statePart <= (others => '0') after 9 ns;

157         saveInArrayX <= '0' after 9 ns;
158         SeqOK <= '0' after 9 ns;
159         oneWaitCycle <= '0' after 9 ns;
160         readAllRegister <= '0' after 9 ns;
161         readOneRegister <= '0' after 9 ns;
162         readAllDone <= '0' after 9 ns;
163         DataReady <= '0' after 9 ns;

165       elsif CLK = '1' and CLK'event then

167         ---------- ADC ----------------------------
168         ADC_CSn <= '1' after 9 ns;  -- defaults
169         ADC_WRn <= '1' after 9 ns;
170         WR_N_EN <= '0' after 9 ns;
171         ADC_RDn <= '1' after 9 ns;
172         PART <= '0' after 9 ns;
173         -------------------------------------
174         ReadNow <= '0' after 9 ns;
175         ADC_CONVSTn <= '1' after 9 ns;
176         PowerOnReset <= '0' after 9 ns;
177         ADCgotoDACreg <= '0' after 9 ns;
178         ADCwrDACreg <= '0' after 9 ns;
179         ADCgotoDACread <= '0' after 9 ns;
180         ADCrdDACreg <= '0' after 9 ns;
181         ADCgotoSEQreg <= '0' after 9 ns;
182         ADCwrSEQreg <= '0' after 9 ns;
183         ADCgotoSEQread <= '0' after 9 ns;
184         ADCrdSEQreg <= '0' after 9 ns;
185         ADCwrConf <= '0' after 9 ns;

187         CheckSEQ <= '0' after 9 ns;
```

```
188         ------------------------------------------------
189         ------------------ DAC -------------------
190         DAC_CSn <= '1' after 9 ns;
191         DAC_WRn <= '1' after 9 ns;
192         DAC_RDn <= '1' after 9 ns;
193         DAC_LDACn <= '1' after 9 ns;
194         DACwrDACreg <= '0' after 9 ns;
195         DACrdDACreg <= '0' after 9 ns;
196         ------------------------------------
197         ProgRegReset <= '0' after 9 ns;
198         DataReady <= '0' after 9 ns;
199
200         statePart <= (others => '0') after 9 ns;  -- sChangeWork + sChangeWork_s0
201
202         case STATE is
203           -- sADCwrPOR
204           when sADCwrPOR_s0 =>  -- Power-On-Reset
205             initRun <= '1' after 9 ns;
206             WR_N_EN <= '1' after 9 ns;
207             PowerOnReset <= '1' after 9 ns;
208             STATE <= sADCwrPOR_s1 after 9 ns;
209             statePart <= "000011" after 9 ns;
210           when sADCwrPOR_s1 =>
211             ADC_CSn <= '0' after 9 ns;
212             ADC_WRn <= '0' after 9 ns;  -- now write data into ADC
213             WR_N_EN <= '1' after 9 ns;
214             PowerOnReset <= '1' after 9 ns;
215             STATE <= sADCwrPOR_s2 after 9 ns;
216             statePart <= "000100" after 9 ns;
217           when sADCwrPOR_s2 =>
218             CONV_CHANNEL <= '0' after 9 ns;
219             STATE <= sADCwrPOR_s3 after 9 ns;
220             waitTwoCycles <= (others => '0') after 9 ns;
221             statePart <= "000101" after 9 ns;
222           when sADCwrPOR_s3 =>
223             if waitTwoCycles = "10" then
224               STATE <= sADCwrDAC_s0 after 9 ns;
225             else
226               waitTwoCycles <= waitTwoCycles + 1 after 9 ns;
227               STATE <= sADCwrPOR_s3 after 9 ns;
228             end if;
229             statePart <= "000110" after 9 ns;
230           ------------------- ADC ------------------------------------------------------------
231           ---------------- write into DAC of ADC ------------
232           -- sADCwrDAC
233           when sADCwrDAC_s0 =>
234             if ADC_BUSY = '0' then
235               STATE <= sADCwrDAC_s1 after 9 ns;
236             else
237               STATE <= sADCwrDAC_s0 after 9 ns;
238             end if;
239             statePart <= "000111" after 9 ns;
240           when sADCwrDAC_s1 =>
241             ADC_CSn <= '0' after 9 ns;
242             WR_N_EN <= '1' after 9 ns;
243             ADCgotoDACreg <= '1' after 9 ns;  -- ADCgotoDACreg
244             STATE <= sADCwrDAC_s2 after 9 ns;
245             statePart <= "001000" after 9 ns;
246           when sADCwrDAC_s2 =>  -- Daten liegen sicher an
247             ADC_CSn <= '0' after 9 ns;
248             WR_N_EN <= '1' after 9 ns;
249             ADCgotoDACreg <= '1' after 9 ns;  -- ADCgotoDACreg
250             STATE <= sADCwrDAC_s3 after 9 ns;
```

```
251             statePart <= "001001" after 9 ns;
252           when sADCwrDAC_s3 =>
253             ADC_CSn <= '0' after 9 ns;
254             ADC_WRn <= '0' after 9 ns;
255             WR_N_EN <= '1' after 9 ns;
256             ADCgotoDACreg <= '1' after 9 ns;  -- ADCgotoDACreg
257             STATE <= sADCwrDAC_s4 after 9 ns;
258             statePart <= "001010" after 9 ns;
259           when sADCwrDAC_s4 =>
260             STATE <= sADCwrDAC_s5 after 9 ns;
261             statePart <= "001011" after 9 ns;
262           when sADCwrDAC_s5 =>
263             ADC_CSn <= '0' after 9 ns;
264             WR_N_EN <= '1' after 9 ns;
265             ADCwrDACreg <= '1' after 9 ns;  -- ADCwrDACreg
266             STATE <= sADCwrDAC_s6 after 9 ns;
267             statePart <= "001100" after 9 ns;
268           when sADCwrDAC_s6 => -- damit Daten sicher anliegen
269             ADC_CSn <= '0' after 9 ns;
270             WR_N_EN <= '1' after 9 ns;
271             ADCwrDACreg <= '1' after 9 ns;  -- ADCwrDACreg
272             STATE <= sADCwrDAC_s7 after 9 ns;
273             statePart <= "001101" after 9 ns;
274           when sADCwrDAC_s7 =>
275             ADC_CSn <= '0' after 9 ns;
276             ADC_WRn <= '0' after 9 ns;
277             WR_N_EN <= '1' after 9 ns;
278             ADCwrDACreg <= '1' after 9 ns;  -- ADCwrDACreg
279             STATE <= sADCwrDAC_s8 after 9 ns;
280             statePart <= "001110" after 9 ns;
281           when sADCwrDAC_s8 =>
282             CONV_CHANNEL <= '0' after 9 ns;
283             ProgRegReset <= '1' after 9 ns;
284             if initRun = '1' then
285               STATE <= sADCwrSEQ_s0 after 9 ns; -- InitStateSeqADC
286             else
287               STATE <= sChangeWork after 9 ns;  -- sADCwrSEQ_s1
288             end if;
289             statePart <= "001111" after 9 ns;
290           ---------------- write into Sequence-Register of ADC ----------------
291           -- sADCwrSEQ
292           when sADCwrSEQ_s0 =>
293             if ADC_BUSY = '0' then
294               STATE <= sADCwrSEQ_s1 after 9 ns;
295             else
296               STATE <= sADCwrSEQ_s0 after 9 ns;
297             end if;
298             statePart <= "010000" after 9 ns;
299           when sADCwrSEQ_s1 =>  -- Sequence-Register beschreiben
300             firstRun <= '1' after 9 ns;
301             WR_N_EN <= '1' after 9 ns;
302             ADCgotoSEQreg <= '1' after 9 ns;
303             STATE <= sADCwrSEQ_s2 after 9 ns;
304             statePart <= "010001" after 9 ns;
305           when sADCwrSEQ_s2 =>  -- daten liegen sich an
306             WR_N_EN <= '1' after 9 ns;
307             ADCgotoSEQreg <= '1' after 9 ns;
308             STATE <= sADCwrSEQ_s3 after 9 ns;
309             statePart <= "010010" after 9 ns;
310           when sADCwrSEQ_s3 =>
311             ADC_CSn <= '0' after 9 ns;
312             ADC_WRn <= '0' after 9 ns;
313             WR_N_EN <= '1' after 9 ns;
```

```
314             ADCgotoSEQreg <= '1' after 9 ns;
315             STATE <= sADCwrSEQ_s4 after 9 ns; --S1
316             statePart <= "010011" after 9 ns;
317           when sADCwrSEQ_s4 =>
318             CONV_CHANNEL <= '1' after 9 ns;
319             STATE <= sADCwrSEQ_s5 after 9 ns;
320             statePart <= "010100" after 9 ns;
321           when sADCwrSEQ_s5 =>
322             WR_N_EN <= '1' after 9 ns;
323             ADCwrSEQreg <= '1' after 9 ns;
324             STATE <= sADCwrSEQ_s6 after 9 ns;
325             statePart <= "010101" after 9 ns;
326           when sADCwrSEQ_s6 =>
327             WR_N_EN <= '1' after 9 ns;
328             ADCwrSEQreg <= '1' after 9 ns;
329             STATE <= sADCwrSEQ_s7 after 9 ns;
330             statePart <= "010110" after 9 ns;
331           when sADCwrSEQ_s7 =>
332             ADC_CSn <= '0' after 9 ns;
333             ADC_WRn <= '0' after 9 ns;
334             WR_N_EN <= '1' after 9 ns;
335             ADCwrSEQreg <= '1' after 9 ns;
336             ProgRegReset <= '1' after 9 ns;
337             if initRun = '1' then
338               STATE <= sDACwrDAC_s0 after 9 ns; -- InitStateDACofDAC
339             else
340               STATE <= sChangeWork after 9 ns;  -- S2
341             end if;
342             statePart <= "010111" after 9 ns;

344           ---------- start conversion and reading ----------
345           when sConfig_s0 =>
346             WR_N_EN <= '1' after 9 ns;
347             ADCwrConf <= '1' after 9 ns;
348             STATE <= sConfig_s1 after 9 ns;
349             statePart <= "000001" after 9 ns;
350           when sConfig_s1 =>
351             ADC_CSn <= '0' after 9 ns;
352             ADC_WRn <= '0' after 9 ns;
353             WR_N_EN <= '1' after 9 ns;
354             ADCwrConf <= '1' after 9 ns;
355             ProgRegReset <= '1' after 9 ns;
356             STATE <= sChangeWork after 9 ns;
357             statePart <= "000010" after 9 ns;

359 ------------------------------------------------------------------------
360 ---------------------------- Change Work -------------------------------
361 ------------------------------------------------------------------------
362           when sChangeWork =>
363             if DataFull = '1' and SendingDone = '1' then
364               saveInArrayX <= '0' after 9 ns;
365             end if;
366             STATE <= sChangeWork_s0 after 9 ns;

368           when sChangeWork_s0 =>
369             readOneRegister <= '0' after 9 ns;
370             readAllDone <= '0' after 9 ns;

372             case ProgReg is
373               when x"0" =>  -- default
374                 if Reading = '1' then
375                   Reading <= '0' after 9 ns;
376                   SeqOK <= '0' after 9 ns;
```

```
377                 saveInArrayX <= '0' after 9 ns;
378               end if;
379               -- Signalabfragen
380               if readAllRegister = '1' then
381                 STATE <= sADCrdSEQ_s0 after 9 ns;
382               else
383                 STATE <= sADCrdBUSY_s0 after 9 ns;  -- dadurch werden immer neue Daten
                        geholt
384               end if;
385             when x"1" =>  -- ref_diff1
386               DAC_A_X <= "000" after 9 ns;
387               STATE <= sDACwrDAC_s0 after 9 ns;
388             when x"2" =>  -- ref_diff2
389               DAC_A_X <= "001" after 9 ns;
390               STATE <= sDACwrDAC_s0 after 9 ns;
391             when x"3" =>  -- ref_hb1
392               DAC_A_X <= "010" after 9 ns;
393               STATE <= sDACwrDAC_s0 after 9 ns;
394             when x"4" =>  -- ref_hb2
395               DAC_A_X <= "011" after 9 ns;
396               STATE <= sDACwrDAC_s0 after 9 ns;
397             when x"5" =>  -- u_broff1
398               DAC_A_X <= "101" after 9 ns;
399               STATE <= sDACwrDAC_s0 after 9 ns;
400             when x"6" =>  -- u_broff2
401               DAC_A_X <= "110" after 9 ns;
402               STATE <= sDACwrDAC_s0 after 9 ns;
403             when x"7" =>  -- u_off
404               DAC_A_X <= "100" after 9 ns;
405               STATE <= sDACwrDAC_s0 after 9 ns;
406             when x"8" =>  -- u_opt
407               DAC_A_X <= "111" after 9 ns;
408               STATE <= sDACwrDAC_s0 after 9 ns;
409             when x"9" =>  -- StateConfig to write anything directly into Configregister
410               STATE <= sConfig_s0 after 9 ns;
411             when x"A" =>  -- ADC_refOut
412               STATE <= sADCwrDAC_s0 after 9 ns;
413             when x"B" =>  -- ADC_sequence
414               STATE <= sADCwrSEQ_s0 after 9 ns;
415             when x"C" =>  -- alle Register von ADC und DAC auslesen
416               readAllDone <= '1' after 9 ns;
417               if readAllDone = '0' then
418                 readAllRegister <= '1' after 9 ns;
419                 STATE <= sADCrdSEQ_s0 after 9 ns;
420               else
421                 STATE <= sChangeWork after 9 ns;
422               end if;
423             when x"D" =>  -- spezielle/gewünschte Register auslesen
424               readOneRegister <= '1' after 9 ns;
425               if readOneRegister = '0' then
426                 STATE <= sDACrdDAC_s0 after 9 ns;
427                 case ProgInfoPart is
428                   when x"1" => -- ref_diff1; VoutA
429                     DAC_A_X <= "000" after 9 ns;
430                   when x"2" => -- ref_diff2; VoutB
431                     DAC_A_X <= "001" after 9 ns;
432                   when x"3" => -- ref_hb1; VoutC
433                     DAC_A_X <= "010" after 9 ns;
434                   when x"4" => -- ref_hb2; VoutD
435                     DAC_A_X <= "011" after 9 ns;
436                   when x"5" => -- u_broff1; VoutF
437                     DAC_A_X <= "101" after 9 ns;
438                   when x"6" => -- u_broff2; VoutG
```

```
439                 DAC_A_X <= "110" after 9 ns;
440               when x"7" => -- u_off; VoutE
441                 DAC_A_X <= "100" after 9 ns;
442               when x"8" => -- u_opt; VoutH
443                 DAC_A_X <= "111" after 9 ns;
444               when x"9" => -- AMPgain
445                 DataReady <= '1' after 9 ns;
446                 STATE <= sChangeWork after 9 ns;
447               when x"A" => -- ADCrefOut
448                 STATE <= sADCrdDAC_s0 after 9 ns;
449               when x"B" => -- ADCsequence
450                 STATE <= sADCrdSEQ_s0 after 9 ns;
451               when others => STATE <= sChangeWork after 9 ns;
452             end case;
453           else
454             STATE <= sChangeWork after 9 ns;
455           end if; -- only one time
456         when x"E" =>  -- Reset and set default values
457           STATE <= sADCwrPOR_s0 after 9 ns;
458         when x"F" =>  -- get newValues (write into arrays)
459           Reading <= '1' after 9 ns;
460           if firstRun = '1' or SeqOK = '0' or Reading = '0' then
461             STATE <= sADCrdSEQ_s0 after 9 ns;
462           else
463             STATE <= sADCrdBUSY_s0 after 9 ns;
464           end if;
465         when others =>
466           STATE <= sChangeWork after 9 ns;
467       end case;

469       ---------------------------- BUSY-FLAG ------------------------------------------
470       -- sADCrdBUSY
471       when sADCrdBUSY_s0 =>                                  -- BUSY-flag abfragen
472         if ADC_BUSY = '0' then
473           STATE <= sADCrdBUSY_s1 after 9 ns;
474         else
475           STATE <= sADCrdBUSY_s0 after 9 ns;
476         end if;
477         statePart <= "110011" after 9 ns;
478       when sADCrdBUSY_s1 =>
479         if ADC_BUSY = '1' then
480           if newCheckSEQ = '1' or SeqOK = '0' then
481             STATE <= sADCrdSEQ_s0 after 9 ns;
482           else
483             if firstRun = '1' or (saveInArrayX = '0' and ProgReg = x"F") or oneWaitCycle
                    = '1' then
484               CONV_CHANNEL <= '1' after 9 ns;
485               STATE <= sADCrdBUSY_s3 after 9 ns;
486             else
487               STATE <= sADCrdBUSY_s2 after 9 ns;
488             end if;
489           end if;
490         else
491           ADC_CONVSTn <= '0' after 9 ns;
492           STATE <= sADCrdBUSY_s1 after 9 ns;
493         end if;
494         statePart <= "110100" after 9 ns;
495       when sADCrdBUSY_s3 =>
496         firstRun <= '0' after 9 ns;
497         oneWaitCycle <= '0' after 9 ns;
498         STATE <= sADCrdBUSY_s4 after 9 ns;
499         statePart <= "110110" after 9 ns;
500       when sADCrdBUSY_s4 =>
```

```
501             if ADC_BUSY = '0' then
502               if SeqOK = '1' then
503                 saveInArrayX <= '1' after 9 ns;
504               else
505                 saveInArrayX <= '0' after 9 ns;
506               end if;
507               STATE <= sADCrdBUSY_s1 after 9 ns;
508             else
509               STATE <= sADCrdBUSY_s4 after 9 ns;
510             end if;
511             statePart <= "110111" after 9 ns;
512           when sADCrdBUSY_s2 =>
513             CONV_CHANNEL <= not CONV_CHANNEL after 9 ns;  -- toggeln
514             STATE <= sADCrdCONV_s0 after 9 ns;
515             statePart <= "110101" after 9 ns;
516           ------------------------------------------ LESEN
                ------------------------------------------
517           -- sADCrdCONV
518           when sADCrdCONV_s0 => -- Werte aus dem ADC auslesen
519             ADC_CSn <= '0' after 9 ns;
520             ADC_RDn <= '0' after 9 ns;
521             PART <= '0' after 9 ns;  -- choose right Register to save data in (1.part)
522             STATE <= sADCrdCONV_s1 after 9 ns;
523             statePart <= "111000" after 9 ns;
524           when sADCrdCONV_s1 => -- Daten liegen sicher an
525             ADC_CSn <= '0' after 9 ns;
526             ADC_RDn <= '0' after 9 ns;
527             PART <= '0' after 9 ns;
528             STATE <= sADCrdCONV_s2 after 9 ns;
529             statePart <= "111001" after 9 ns;
530           when sADCrdCONV_s2 =>
531             ADC_CSn <= '0' after 9 ns;
532             ADC_RDn <= '0' after 9 ns;
533             ReadNow <= '1' after 9 ns;
534             PART <= '0' after 9 ns;
535             STATE <= sADCrdCONV_s3 after 9 ns;
536             statePart <= "111010" after 9 ns;
537           when sADCrdCONV_s3 =>
538             STATE <= sADCrdCONV_s4 after 9 ns;
539             statePart <= "111011" after 9 ns;
540           when sADCrdCONV_s4 =>
541             ADC_CSn <= '0' after 9 ns;
542             ADC_RDn <= '0' after 9 ns;
543             PART <= '1' after 9 ns;  -- choose right Register to save data in(2.part)
544             STATE <= sADCrdCONV_s5 after 9 ns;
545             statePart <= "111100" after 9 ns;
546           when sADCrdCONV_s5 =>
547             ADC_CSn <= '0' after 9 ns;
548             ADC_RDn <= '0' after 9 ns;
549             PART <= '1' after 9 ns;
550             STATE <= sADCrdCONV_s6 after 9 ns;
551             statePart <= "111101" after 9 ns;
552           when sADCrdCONV_s6 =>
553             ADC_CSn <= '0' after 9 ns;
554             ADC_RDn <= '0' after 9 ns;
555             ReadNow <= '1' after 9 ns;
556             PART <= '1' after 9 ns;
557             STATE <= sADCrdCONV_s7 after 9 ns;
558             statePart <= "111110" after 9 ns;
559           when sADCrdCONV_s7 =>
560             STATE <= sChangeWork after 9 ns;  -- entweder Sequence oder STANDBY
561             statePart <= "111111" after 9 ns;
```

```
563         --
            ------------------------------------------------------------------------------

564         ---------------------- read sequence-register of ADC ----------------------
565         -- sADCrdSEQ
566         when sADCrdSEQ_s0 =>
567           WR_N_EN <= '1' after 9 ns;
568           ADCgotoSEQread <= '1' after 9 ns; -- ADCgotoSEQread
569           STATE <= sADCrdSEQ_s1 after 9 ns;
570           statePart <= "011101" after 9 ns;
571         when sADCrdSEQ_s1 =>
572           ADC_CSn <= '0' after 9 ns;
573           WR_N_EN <= '1' after 9 ns;
574           ADCgotoSEQread <= '1' after 9 ns; -- ADCgotoSEQread
575           STATE <= sADCrdSEQ_s2 after 9 ns;
576           statePart <= "011110" after 9 ns;
577         when sADCrdSEQ_s2 =>
578           ADC_CSn <= '0' after 9 ns;
579           ADC_WRn <= '0' after 9 ns;
580           WR_N_EN <= '1' after 9 ns;
581           ADCgotoSEQread <= '1' after 9 ns; -- ADCgotoSEQread
582           STATE <= sADCrdSEQ_s3 after 9 ns;
583           statePart <= "011111" after 9 ns;
584         when sADCrdSEQ_s3 =>
585           STATE <= sADCrdSEQ_s4 after 9 ns;
586           statePart <= "100000" after 9 ns;
587         when sADCrdSEQ_s4 =>
588           ADC_CSn <= '0' after 9 ns;
589           ADC_RDn <= '0' after 9 ns;
590           ADCrdSEQreg <= '1' after 9 ns;  -- ADCrdSEQreg
591           STATE <= sADCrdSEQ_s5 after 9 ns;
592           statePart <= "100001" after 9 ns;
593         when sADCrdSEQ_s5 =>
594           ADC_CSn <= '0' after 9 ns;
595           ADC_RDn <= '0' after 9 ns;
596           ADCrdSEQreg <= '1' after 9 ns;  -- ADCrdSEQreg
597           STATE <= sADCrdSEQ_s6 after 9 ns;
598           statePart <= "100010" after 9 ns;
599         when sADCrdSEQ_s6 =>
600           ADC_CSn <= '0' after 9 ns;
601           ADC_RDn <= '0' after 9 ns;
602           ReadNow <= '1' after 9 ns;
603           ADCrdSEQreg <= '1' after 9 ns;  -- ADCrdSEQreg
604           STATE <= sADCrdSEQ_s7 after 9 ns;
605           statePart <= "100011" after 9 ns;
606         when sADCrdSEQ_s7 =>
607           if readAllRegister = '1' then
608             STATE <= sADCrdDAC_s0 after 9 ns;
609           elsif ProgReg = x"D" then
610             STATE <= sReadDone after 9 ns;
611           else
612             STATE <= sADCcheckSEQ_s0 after 9 ns;
613           end if;
614           statePart <= "100100" after 9 ns;

616         --
            ------------------------------------------------------------------------------

617         ---------- check Sequence-register which channel is converted next ---------
618         -- sADCcheckSEQ
619         when sADCcheckSEQ_s0 =>
620           CheckSEQ <= '1' after 9 ns;
621           STATE <= sADCcheckSEQ_s1 after 9 ns;
```

```
622           statePart <= "100101" after 9 ns;
623         when sADCcheckSEQ_s1 => -- wait state
624           STATE <= sADCcheckSEQ_s2 after 9 ns;
625           statePart <= "100110" after 9 ns;
626         when sADCcheckSEQ_s2 =>
627           newCheckSEQ <= '0' after 9 ns;
628           if newTestSEQ = '1' then
629             newCheckSEQ <= '1' after 9 ns;
630           else
631             SeqOK <= '1' after 9 ns;  -- für ProgReg = x"F"
632             oneWaitCycle <= '1' after 9 ns;
633           end if;
634           STATE <= sADCrdBUSY_s0 after 9 ns;
635           statePart <= "100111" after 9 ns;

637         ------------------------------------------------------------
638         ------------------ read DAC-register of ADC ----------------
639         -- sADCrdDAC
640         when sADCrdDAC_s0 =>
641           ADC_CSn <= '0' after 9 ns;
642           WR_N_EN <= '1' after 9 ns;
643           ADCgotoDACread <= '1' after 9 ns; -- ADCgotoDACread
644           STATE <= sADCrdDAC_s1 after 9 ns;
645           statePart <= "101000" after 9 ns;
646         when sADCrdDAC_s1 =>
647           ADC_CSn <= '0' after 9 ns;
648           WR_N_EN <= '1' after 9 ns;
649           ADCgotoDACread <= '1' after 9 ns; -- ADCgotoDACread
650           STATE <= sADCrdDAC_s2 after 9 ns;
651           statePart <= "101001" after 9 ns;
652         when sADCrdDAC_s2 =>
653           ADC_CSn <= '0' after 9 ns;
654           ADC_WRn <= '0' after 9 ns;
655           WR_N_EN <= '1' after 9 ns;
656           ADCgotoDACread <= '1' after 9 ns; -- ADCgotoDACread
657           STATE <= sADCrdDAC_s3 after 9 ns;
658           statePart <= "101010" after 9 ns;
659         when sADCrdDAC_s3 =>
660           STATE <= sADCrdDAC_s4 after 9 ns;
661           statePart <= "101011" after 9 ns;
662         when sADCrdDAC_s4 =>
663           ADC_CSn <= '0' after 9 ns;
664           ADC_RDn <= '0' after 9 ns;
665           ADCrdDACreg <= '1' after 9 ns;  -- ADCrdDACreg
666           STATE <= sADCrdDAC_s5 after 9 ns;
667           statePart <= "101100" after 9 ns;
668         when sADCrdDAC_s5 =>
669           ADC_CSn <= '0' after 9 ns;
670           ADC_RDn <= '0' after 9 ns;
671           ADCrdDACreg <= '1' after 9 ns;  -- ADCrdDACreg
672           STATE <= sADCrdDAC_s6 after 9 ns;
673           statePart <= "101101" after 9 ns;
674         when sADCrdDAC_s6 =>
675           ADC_CSn <= '0' after 9 ns;
676           ADC_RDn <= '0' after 9 ns;
677           ReadNow <= '1' after 9 ns;
678           ADCrdDACreg <= '1' after 9 ns;  -- ADCrdDACreg
679           if readAllRegister = '1' then
680             STATE <= sDACrdDAC_s0 after 9 ns;
681           else
682             STATE <= sReadDone after 9 ns;
683           end if;
684           statePart <= "101110" after 9 ns;
```

```
685         ------------------------------------------------------------------

687         ------------------ DAC ---------------------------------------------
688         ------------------- write DAC-value of DAC ---------
689         -- sDACwrDAC
690         when sDACwrDAC_s0 =>
691           DAC_CSn <= '0' after 9 ns;
692           WR_N_EN <= '1' after 9 ns;
693           DACwrDACreg <= '1' after 9 ns;  -- DACwrDACreg
694           STATE <= sDACwrDAC_s1 after 9 ns;
695           statePart <= "011000" after 9 ns;
696         when sDACwrDAC_s1 =>
697           DAC_CSn <= '0' after 9 ns;
698           WR_N_EN <= '1' after 9 ns;
699           DACwrDACreg <= '1' after 9 ns;  -- DACwrDACreg
700           STATE <= sDACwrDAC_s2 after 9 ns;
701           statePart <= "011001" after 9 ns;
702         when sDACwrDAC_s2 =>
703           DAC_CSn <= '0' after 9 ns;
704           DAC_WRn <= '0' after 9 ns;
705           WR_N_EN <= '1' after 9 ns;
706           DAC_GAIN <= '1' after 9 ns;
707           DAC_BUF <= '0' after 9 ns;
708           DAC_LDACn <= '0' after 9 ns;  -- synchronous update mode
709           DACwrDACreg <= '1' after 9 ns;  -- DACwrDACreg
710           STATE <= sDACwrDAC_s3 after 9 ns;
711           statePart <= "011010" after 9 ns;
712         when sDACwrDAC_s3 =>
713           STATE <= sDACwrDAC_s4 after 9 ns;
714           statePart <= "011011" after 9 ns;
715         when sDACwrDAC_s4 =>
716           ProgRegReset <= '1' after 9 ns;
717           if initRun = '1' then
718             STATE <= sDACwrDAC_s0 after 9 ns;
719             case DAC_A_X is
720               when "000" => DAC_A_X <= "001" after 9 ns;
721               when "001" => DAC_A_X <= "010" after 9 ns;
722               when "010" => DAC_A_X <= "011" after 9 ns;
723               when "011" => DAC_A_X <= "100" after 9 ns;
724               when "100" => DAC_A_X <= "101" after 9 ns;
725               when "101" => DAC_A_X <= "110" after 9 ns;
726               when "110" => DAC_A_X <= "111" after 9 ns;
727               when "111" => DAC_A_X <= "000" after 9 ns;
728                             initRun <= '0' after 9 ns;
729                             readAllRegister <= '1' after 9 ns;
730                             STATE <= sADCrdSEQ_s0 after 9 ns;
731             end case;
732           else
733             STATE <= sChangeWork after 9 ns;
734           end if;
735           statePart <= "011100" after 9 ns;
736         ------------------------------------------------------------------
737         ------------------ READ DAC-register of DAC ------------------------
738         -- sDACrdDAC
739         when sDACrdDAC_s0 =>
740           DAC_CSn <= '0' after 9 ns;
741           DAC_RDn <= '0' after 9 ns;
742           DACrdDACreg <= '1' after 9 ns;
743           STATE <= sDACrdDAC_s1 after 9 ns;
744           statePart <= "101111" after 9 ns;
745         when sDACrdDAC_s1 =>
746           DAC_CSn <= '0' after 9 ns;
747           DAC_RDn <= '0' after 9 ns;
```

```
748             DACrdDACreg <= '1' after 9 ns;
749             STATE <= sDACrdDAC_s2 after 9 ns;
750             statePart <= "110000" after 9 ns;
751           when sDACrdDAC_s2 =>
752             DAC_CSn <= '0' after 9 ns;
753             DAC_RDn <= '0' after 9 ns;
754             ReadNow <= '1' after 9 ns;
755             DACrdDACreg <= '1' after 9 ns;
756             STATE <= sDACrdDAC_s3 after 9 ns;
757             statePart <= "110001" after 9 ns;
758           when sDACrdDAC_s3 =>
759             if readAllRegister = '1' then
760               STATE <= sDACrdDAC_s0 after 9 ns;
761               case DAC_A_X is
762                 when "000" => DAC_A_X <= "001" after 9 ns;
763                 when "001" => DAC_A_X <= "010" after 9 ns;
764                 when "010" => DAC_A_X <= "011" after 9 ns;
765                 when "011" => DAC_A_X <= "100" after 9 ns;
766                 when "100" => DAC_A_X <= "101" after 9 ns;
767                 when "101" => DAC_A_X <= "110" after 9 ns;
768                 when "110" => DAC_A_X <= "111" after 9 ns;
769                 when "111" => DAC_A_X <= "000" after 9 ns;
770                               readAllRegister <= '0' after 9 ns;
771                               STATE <= sReadDone after 9 ns;
772               end case;
773             else
774               STATE <= sChangeWork after 9 ns;
775               case ProgInfoPart is
776                 when x"1" | x"2" | x"3" | x"4" | x"5" | x"6" | x"7" | x"8" => STATE <=
                        sReadDone after 9 ns;
777                 when others => null;
778               end case;
779             end if;
780             statePart <= "110010" after 9 ns;
781           ------------------------------------------------------------
782           when sReadDone =>
783             DataReady <= '1' after 9 ns;
784             STATE <= sChangeWork after 9 ns;
785             statePart <= "000010" after 9 ns;
786         end case;

788       end if; -- CLK

790       CONV_CH      <= CONV_CHANNEL;
791       DAC_A        <= DAC_A_X;
792       saveInArray <= saveInArrayX;

794     end process READ_CONV;

796   end FSM;
```

C.1.5 ADC-Platine: Datenpfad

```
1   ------------------------------------------------------------
2   -- Datapath for reading and writing data
3   --
4   -- Author: A. Arvidsson
5   ------------------------------------------------------------
```

```
7  -- <makepackage>
8  --   <global>
9  --     constant ADC_Width : integer := 12;
10 --   </global>
11 --     <component = Datapath>
12 --     <name = arvidsson>
13 --     <project = ada_conv_modsys>
14 --     </component>
15 -- </makepackage>

17 library ieee;
18   use ieee.std_logic_1164.all;
19   use ieee.numeric_std.all;

21 library work;
22   use work.global_pkg.all;

24 entity Datapath is
25   port(
26     CLK              : in std_logic;
27     RESET            : in std_logic;
28     --
29     WR_N_EN          : in bit;
30     PART             : in bit;
31     CONV_CH          : in bit;
32     ReadNow          : in bit;
33     -- ADC --
34     PowerOnReset     : in bit;
35     ADCgotoDACreg    : in bit;
36     ADCwrDACreg      : in bit;
37     ADCgotoDACread   : in bit;
38     ADCrdDACreg      : in bit;
39     ADCgotoSEQreg    : in bit;
40     ADCwrSEQreg      : in bit;
41     ADCgotoSEQread   : in bit;
42     ADCrdSEQreg      : in bit;
43     ADCwrConf        : in bit;
44     CheckSEQ         : in bit; -- check sequence-channel
45     newTestSEQ       : out bit;  -- to test again until channel ist right
46     -- DAC --
47     DACwrDACreg      : in bit;
48     DACrdDACreg      : in bit;
49     ---------
50     DAC_A              : in bit_vector(2 downto 0);
51     DB               : inout std_logic_vector(11 downto 0);  -- Ein-/Ausgangsdaten
52     NEW_DATA         : out bit_vector(1 downto 0);
53     DB_UDIFF         : out std_logic_vector(11 downto 0);
54     DB_HB1           : out std_logic_vector(11 downto 0);
55     DB_HB2           : out std_logic_vector(11 downto 0);
56     ---------- Outputs ----------
57       -- DAC --
58     ref_diff1        : out std_logic_vector(11 downto 0);
59     ref_diff2        : out std_logic_vector(11 downto 0);
60     ref_hb1          : out std_logic_vector(11 downto 0);
61     ref_hb2          : out std_logic_vector(11 downto 0);
62     u_broff1         : out std_logic_vector(11 downto 0);
63     u_broff2         : out std_logic_vector(11 downto 0);
64     u_off            : out std_logic_vector(11 downto 0);
65     u_opt            : out std_logic_vector(11 downto 0);
66       -- ADC --
67     ADCrefOut        : out std_logic_vector(11 downto 0);
68     ADCsequence      : out std_logic_vector(11 downto 0);
69     OutRDspecReg     : out std_logic_vector( 3 downto 0);
```

```
70      -- ------------------------------------
71      ProgReg    : in std_logic_vector(3 downto 0);
72      ProgInfo   : in std_logic_vector(11 downto 0);
73      newProgReg : in bit
74      ------------------------------------
75      );
76  end Datapath;

78  architecture behaviour of Datapath is
79  constant DB_CHx0     : std_logic_vector(11 downto 0) := "000100000000";  -- x"100"
80  constant DB_CHx1     : std_logic_vector(11 downto 0) := "110100000000";  -- x"D00"
81  constant DB_SEQ      : std_logic_vector(11 downto 0) := x"104";  -- ins Sequence-Register
        wechseln
82  constant DB_SEQ_CTR  : std_logic_vector(11 downto 0) := x"230";

84  signal WriteTo     : std_logic_vector(11 downto 0);
85  signal ReadFrom    : std_logic_vector(11 downto 0);
86  signal WriteNow    : bit;
87  signal count       : bit;

89  -- read from Matlab
90    -- DAC --
91  signal progDAC_A     : std_logic_vector(11 downto 0);
92  signal progDAC_B     : std_logic_vector(11 downto 0);
93  signal progDAC_C     : std_logic_vector(11 downto 0);
94  signal progDAC_D     : std_logic_vector(11 downto 0);
95  signal progDAC_E     : std_logic_vector(11 downto 0);
96  signal progDAC_F     : std_logic_vector(11 downto 0);
97  signal progDAC_G     : std_logic_vector(11 downto 0);
98  signal progDAC_H     : std_logic_vector(11 downto 0);
99    -- ADC --
100 signal progADCconf   : std_logic_vector(11 downto 0);
101 signal progRefOut    : std_logic_vector(11 downto 0);
102 signal progSequence  : std_logic_vector(11 downto 0);
103 signal rdSpecialReg  : std_logic_vector( 3 downto 0);

105 signal ADCseq : std_logic_vector(11 downto 0);

107 begin

109   REG_DATA : process(RESET,CLK) -- control which data to write or read
110   begin
111     if RESET = '1' then
112       DB_UDIFF  <= x"000" after 9 ns; -- default
113       DB_HB1    <= x"000" after 9 ns;
114       DB_HB2    <= x"000" after 9 ns;
115       NEW_DATA  <=   "10" after 9 ns; -- default (nicht vergeben)
116       ADCrefOut <= x"000" after 9 ns;
117       ADCseq    <= x"000" after 9 ns;
118       ref_diff1 <= x"000" after 9 ns;
119       ref_diff2 <= x"000" after 9 ns;
120       ref_hb1   <= x"000" after 9 ns;
121       ref_hb2   <= x"000" after 9 ns;
122       u_broff1  <= x"000" after 9 ns;
123       u_broff2  <= x"000" after 9 ns;
124       u_off     <= x"000" after 9 ns;
125       u_opt     <= x"000" after 9 ns;
126       WriteTo   <= (others => '0') after 9 ns;
127       WriteNow  <= '0' after 9 ns;
128       count     <= '0' after 9 ns;
129     elsif CLK = '1' and CLK'event then
130       NEW_DATA <= "10" after 9 ns; -- default (nicht vergeben)
131       WriteTo <= (others => '0') after 9 ns;
```

```
132         WriteNow <= '0' after 9 ns;

134         count <= '0' after 9 ns;

136         if ReadNow = '1' then
137           if DACrdDACreg = '1' then -- read DAC-register
138             case DAC_A is
139               when "000" => ref_diff1 <= ReadFrom after 9 ns;
140               when "001" => ref_diff2 <= ReadFrom after 9 ns;
141               when "010" => ref_hb1   <= ReadFrom after 9 ns;
142               when "011" => ref_hb2   <= ReadFrom after 9 ns;
143               when "101" => u_broff1  <= ReadFrom after 9 ns;
144               when "110" => u_broff2  <= ReadFrom after 9 ns;
145               when "100" => u_off     <= ReadFrom after 9 ns;
146               when "111" => u_opt     <= ReadFrom after 9 ns;
147             end case;
148           elsif ADCrdSEQreg = '1' then
149             ADCseq <= ReadFrom after 9 ns;
150           elsif ADCrdDACreg = '1' then
151             ADCrefOut <= ReadFrom after 9 ns;
152           elsif CONV_CH = '0' then
153             if PART = '0' then  -- PART = '0' ist 1.Teil vom Konvertieren, '1' der 2.Teil
154               DB_UDIFF <= ReadFrom after 9 ns;
155               NEW_DATA <= "00" after 9 ns;
156             elsif PART = '1' then
157               DB_HB1 <= ReadFrom after 9 ns;
158               NEW_DATA <= "01" after 9 ns;
159             end if; -- PART
160           elsif CONV_CH = '1' then
161             if PART = '0' then
162               NEW_DATA <= "10" after 9 ns;
163             elsif PART = '1' then
164               DB_HB2 <= ReadFrom after 9 ns;
165               NEW_DATA <= "11" after 9 ns;
166             end if; -- PART
167           end if; -- CONV_CH
168         else         -- ReadNow

170           -- set values to ADC and DAC
171           if WR_N_EN = '1' then
172             WriteNow <= '1' after 9 ns;
173             -- get values and written to output to use it further in vhdl-codes
174             if PowerOnReset = '1' then  -- SW-reset of ADC
175               WriteTo <= x"005" after 9 ns;
176             elsif DACwrDACreg = '1' then  -- write to DAC of DAC
177               case DAC_A is
178                 when "000" => WriteTo <= progDAC_A after 9 ns;
179                 when "001" => WriteTo <= progDAC_B after 9 ns;
180                 when "010" => WriteTo <= progDAC_C after 9 ns;
181                 when "011" => WriteTo <= progDAC_D after 9 ns;
182                 when "100" => WriteTo <= progDAC_E after 9 ns;
183                 when "101" => WriteTo <= progDAC_F after 9 ns;
184                 when "110" => WriteTo <= progDAC_G after 9 ns;
185                 when "111" => WriteTo <= progDAC_H after 9 ns;
186                               if count = '0' then
187                                 count <= '1' after 9 ns;
188                               end if;
189               end case;
190             elsif ADCgotoSEQread = '1' then -- get sequencer-register of ADC
191               WriteTo <= x"106" after 9 ns;
192             elsif ADCgotoDACread = '1' then -- get DAC-register of ADC
193               WriteTo <= x"103" after 9 ns;
194             elsif ADCwrConf = '1' then
```

```
195             WriteTo <= progADCconf after 9 ns;
196           elsif ADCgotoDACreg = '1' then  -- goto DACreg of ADC
197             WriteTo <= x"101" after 9 ns; -- goto DAC-register of ADC
198           elsif ADCwrDACreg = '1' then  -- write to dac-reg of adc
199             WriteTo <= progRefOut after 9 ns;
200           elsif ADCgotoSEQreg = '1' then  -- goto sequence-register of adc
201             WriteTo <= DB_SEQ after 9 ns; -- DB_SEQ
202           elsif ADCwrSEQreg = '1' then  -- write to seq-reg of adc
203             WriteTo <= progSequence after 9 ns;
204           end if; -- DAC
205         end if; -- WR_N_EN

207       end if; -- ReadNow
208     end if; -- RESET and CLK
209   end process REG_DATA;

211   ADCsequence <= ADCseq;

213   OUTPUT : process (WriteNow, DB, WriteTo)     -- Behavioral representation
214   begin                                        -- of tri-states.
215     if WriteNow = '0' then
216       DB <= (others => 'Z') after 9 ns;
217       ReadFrom <= DB after 9 ns;
218     elsif WriteNow = '1' then
219       DB <= WriteTo after 9 ns;
220       ReadFrom <= DB after 9 ns;
221     end if;
222   end process OUTPUT;

224   Auswerten : process(RESET,CLK)
225   begin
226     if RESET = '1' then
227       progDAC_A      <= x"390" after 9 ns; -- x"380"
228       progDAC_B      <= x"410" after 9 ns; -- x"450"
229       progDAC_C      <= (others => '0') after 9 ns;
230       progDAC_D      <= (others => '0') after 9 ns;
231       progDAC_E      <= x"400" after 9 ns;  -- 2048, aber da doppelter Bereich => 1024
232       progDAC_F      <= (others => '0') after 9 ns;
233       progDAC_G      <= (others => '0') after 9 ns;
234       progDAC_H      <= (others => '0') after 9 ns;
235       progADCconf    <= (others => '0') after 9 ns;
236       progRefOut     <= x"3FF" after 9 ns;
237       progSequence   <= DB_SEQ_CTR after 9 ns;
238       rdSpecialReg   <= (others => '0') after 9 ns;
239     elsif CLK = '1' and CLK'event then
240       if newProgReg = '1' then
241         case ProgReg is
242           when x"1" => progDAC_A     <= ProgInfo after 9 ns;
243           when x"2" => progDAC_B     <= ProgInfo after 9 ns;
244           when x"3" => progDAC_C     <= ProgInfo after 9 ns;
245           when x"4" => progDAC_D     <= ProgInfo after 9 ns;
246           when x"5" => progDAC_F     <= ProgInfo after 9 ns;
247           when x"6" => progDAC_G     <= ProgInfo after 9 ns;
248           when x"7" => progDAC_E     <= ProgInfo after 9 ns;
249           when x"8" => progDAC_H     <= ProgInfo after 9 ns;
250           when x"9" => progADCconf   <= ProgInfo after 9 ns;
251           when x"A" => progRefOut    <= ProgInfo after 9 ns;
252           when x"B" => progSequence  <= ProgInfo after 9 ns;
253           when x"D" => rdSpecialReg  <= ProgInfo(3 downto 0) after 9 ns;
254           when others => null;
255         end case;
256       end if;
257     end if; -- RESET and CLK
```

```
258   end process Auswerten;

260   OutRDspecReg <= rdSpecialReg;

262   DataCheck : process(RESET,CLK)
263   begin
264     if RESET = '1' then
265       newTestSEQ <= '0' after 9  ns;
266     elsif CLK = '1' and CLK'event then
267       if CheckSEQ = '1' then
268         case ADCseq(1 downto 0) is
269           when "01" => newTestSEQ <= '0' after 9 ns;
270           when others => newTestSEQ <= '1' after 9 ns;
271         end case;
272       end if; -- CheckSEQ
273     end if; -- RESET and CLK
274   end process DataCheck;

276 end behaviour;
```

C.1.6 7-Segment-Anzeige: Steuerung

```
1  ----------------------------------------------------------------
2  --  seven-segment-controller for segment-output
3  --
4  --  Author: A. Arvidsson
5  ----------------------------------------------------------------

7  -- <makepackage>
8  --      <component = SEV_SEG_CTR>
9  --      <name = arvidsson>
10 --      <project = ada_conv_modsys>
11 --       </component>
12 -- </makepackage>

14 library IEEE;
15   use IEEE.STD_LOGIC_1164.ALL;
16   use ieee.numeric_std.all;

18 library work;
19   use work.global_pkg.all;

21 entity SEV_SEG_CTR is
22   port(
23     CLK_SEG : in std_logic;
24     RESET : in std_logic;
25     DATA : in std_logic_vector(11 downto 0);
26     ANODES : out bit_vector(3 downto 0);
27     DB_PART : out std_logic_vector(3 downto 0)
28     );
29 end SEV_SEG_CTR;

31 architecture SEG_CTR of SEV_SEG_CTR is
32 signal AN : bit_vector(2 downto 0);

34 begin

36   ANODE_CNT : process(CLK_SEG,RESET)
37   variable counter : unsigned(24 downto 0);
```

```
38    constant MaxCnt : unsigned(24 downto 0) := (others => '1');
39    variable DataSave : std_logic_vector(11 downto 0);
40    alias segment_select : unsigned(1 downto 0) is counter(8 downto 7);
41    alias segment_write : unsigned(6 downto 0) is counter(6 downto 0);
42    constant segment_write_zero : unsigned(6 downto 0) := (others =>'0');

44    begin

46      if RESET = '1' then
47        DB_PART <= x"0" after 9 ns;
48        counter := (others => '0');
49        DataSave := (others => '0');
50      elsif CLK_SEG = '1' and CLK_SEG'event then
51        if segment_write = segment_write_zero then
52          case segment_select is
53            when "01" => DB_PART <= DataSave(11 downto 8) after 9 ns;
54                         AN <= "011";
55            when "10" => DB_PART <= DataSave(7 downto 4) after 9 ns;
56                         AN <= "101";
57            when "11" => DB_PART <= DataSave(3 downto 0) after 9 ns;
58                         AN <= "110";
59            when others => DB_PART <= x"F" after 9 ns;
60                           AN <= "111";
61          end case;
62        end if;

64        if counter = MaxCnt then --"11111111"
65          DataSave := DATA;
66          counter := (others => '0');
67        else
68          counter := counter + 1;
69        end if;
70      end if;
71    end process ANODE_CNT;

73    ANODES <= '1' & AN after 9 ns;

75  end SEG_CTR;
```

C.1.7 7-Segment-Anzeige: Anzeige

```
1   ------------------------------------------------------------
2   --  seven-segment for testing in- and output of db
3   --
4   --  Author: A. Arvidsson
5   ------------------------------------------------------------

7   -- <makepackage>
8   --      <component = SEVEN_SEGMENT>
9   --      <name = arvidsson>
10  --      <project = ada_conv_modsys>
11  --       </component>
12  -- </makepackage>

14  library IEEE;
15    use IEEE.STD_LOGIC_1164.ALL;
16    use ieee.numeric_std.all;

18  library work;
```

```
19   use work.global_pkg.all;

21 entity SEVEN_SEGMENT is
22   port(
23     DB_PART : in std_logic_vector(3 downto 0);
24     SEG : out bit_vector(6 downto 0);
25     dp : out bit
26     );
27 end SEVEN_SEGMENT;

29 architecture sev_seg of SEVEN_SEGMENT is
30 begin
31   with DB_PART select
32     SEG <=  "1000000" when x"0" ,
33             "1111001" when x"1" ,
34             "0100100" when x"2" ,
35             "0110000" when x"3" ,
36             "0011001" when x"4" ,
37             "0010010" when x"5" ,
38             "0000010" when x"6" ,
39             "1111000" when x"7" ,
40             "0000000" when x"8" ,
41             "0010000" when x"9" ,
42             "0001000" when x"A" ,
43             "0000011" when x"B" ,
44             "1000110" when x"C" ,
45             "0100001" when x"D" ,
46             "0000110" when x"E" ,
47             "0001110" when x"F" ,
48             "0000000" when others;
49   with DB_PART select
50     dp <= '1' when others;

52 end sev_seg;
```

C.1.8 7-Segment-Anzeige: Multiplexer

```
1  --#######################################################
2  --##   Project name : ada_conv_modsys
3  --##   File name    : seven_seg_mux.vhdl
4  --##   Author       : Jan-Heiner Dreschhoff
5  --##   Re-Author    : Andreas Arvidsson

7  --<makepackage>
8  --   <component = seven_seg_mux>
9  --     <name = arvidsson>
10 --     <project = ada_conv_modsys>
11 --   </component>
12 --</makepackage>

14 library ieee;
15   use ieee.std_logic_1164.all;
16   use ieee.numeric_std.all;

18 library work;
19   use work.global_pkg.all;

21 entity seven_seg_mux is
22   port (
```

```
23      SWITCHES          : in bit_vector(2 downto 0);
24      --
25      CHANNEL_0         : in std_logic_vector(11 downto 0);
26      CHANNEL_1         : in std_logic_vector(11 downto 0);
27      CHANNEL_2         : in std_logic_vector(11 downto 0);
28      CHANNEL_3         : in std_logic_vector(11 downto 0);
29      CHANNEL_4         : in std_logic_vector(11 downto 0);
30      CHANNEL_5         : in std_logic_vector(11 downto 0);
31      CHANNEL_6         : in std_logic_vector(11 downto 0);
32      CHANNEL_7         : in std_logic_vector(11 downto 0);
33      --
34      CHANNEL_OUT        : out std_logic_vector(11 downto 0)
35    );
36    end entity;

38    architecture mux of seven_seg_mux is

40    begin

42    process(SWITCHES, CHANNEL_0
43            , CHANNEL_1
44            , CHANNEL_2
45            , CHANNEL_3
46            , CHANNEL_4
47            , CHANNEL_5
48            , CHANNEL_6
49            , CHANNEL_7
50            )
51    begin
52      case SWITCHES is
53        when "000"  => CHANNEL_OUT <= CHANNEL_0;
54        when "001"  => CHANNEL_OUT <= CHANNEL_1;
55        when "010"  => CHANNEL_OUT <= CHANNEL_2;
56        when "011"  => CHANNEL_OUT <= CHANNEL_3;
57        when "100"  => CHANNEL_OUT <= CHANNEL_4;
58        when "101"  => CHANNEL_OUT <= CHANNEL_5;
59        when "110"  => CHANNEL_OUT <= CHANNEL_6;
60        when "111"  => CHANNEL_OUT <= CHANNEL_7;
61      end case;
62    end process;

64    end mux;
```

C.1.9 MiniUART

```
1   ----------------------------------------------------------------------
2   -- Title      : UART
3   -- Project    : UART
4   ----------------------------------------------------------------------
5   -- File       : MiniUart.vhd
6   -- Author     : Philippe CARTON
7   --              (philippe.carton2@libertysurf.fr)
8   -- Organization:
9   -- Created    : 15/12/2001
10  -- Last update : 8/1/2003
11  -- Platform   : Foundation 3.1i
12  -- Simulators : ModelSim 5.5b
13  -- Synthesizers: Xilinx Synthesis
14  -- Targets    : Xilinx Spartan
```

```
15 -- Dependency  : IEEE std_logic_1164, Rxunit.vhd, Txunit.vhd, utils.vhd
16 ----------------------------------------------------------------------------
17 -- Description: Uart (Universal Asynchronous Receiver Transmitter) for SoC.
18 --    Wishbone compatable.
19 ----------------------------------------------------------------------------
20 -- Copyright (c) notice
21 --    This core adheres to the GNU public license
22 --
23 ----------------------------------------------------------------------------
24 -- Revisions        :
25 -- Revision Number :
26 -- Version          :
27 -- Date     :
28 -- Modifier         : name <email>
29 -- Description      :
30 --
31 ----------------------------------------------------------------------------
32 --
33 ----------------------------------------------------------------------------
34 ----------------------------------------------------------------------------
35 --  MiniUART for Serial-Matlab-Interface
36 --
37 --  Re-Author: A. Arvidsson
38 ----------------------------------------------------------------------------
39 --
40 -- <makepackage>
41 --      <component = MiniUART>
42 --      <name = arvidsson>
43 --      <project = ada_conv_modsys>
44 --       </component>
45 -- </makepackage>
46 ----------------------------------------------------------------------------

48 library ieee;
49    use ieee.std_logic_1164.all;

51 entity MiniUART is
52   generic(BRDIVISOR: INTEGER range 0 to 65535 := 163); -- Baud rate divisor  -- 103 -- 27
53   port (
54 -- Wishbone signals
55      WB_CLK_I : in  std_logic;  -- clock
56      WB_RST_I : in  std_logic;  -- Reset input
57      WB_ADR_I : in  std_logic_vector(1 downto 0); -- Adress bus
58      WB_DAT_I : in  std_logic_vector(7 downto 0); -- DataIn Bus
59      WB_DAT_O : out std_logic_vector(7 downto 0); -- DataOut Bus
60      WB_WE_I  : in  std_logic;  -- Write Enable
61      WB_STB_I : in  std_logic;  -- Strobe
62      WB_ACK_O : out std_logic;  -- Acknowledge
63 -- process signals
64      IntTx_O  : out std_logic;  -- Transmit interrupt: indicate waiting for Byte
65      IntRx_O  : out std_logic;  -- Receive interrupt: indicate Byte received
66      BR_Clk_I : in  std_logic;  -- Clock used for Transmit/Receive
67      TxD_PAD_O: out std_logic;  -- Tx RS232 Line
68      RxD_PAD_I: in  std_logic);  -- Rx RS232 Line
69 end MiniUART;

71 -- Architecture for UART for synthesis
72 architecture Behaviour of MiniUART is

74   component Counter
75   generic(COUNT: INTEGER range 0 to 65535); -- Count revolution
76   port (
77      Clk       : in  std_logic;  -- Clock
```

```
78      Reset     : in  std_logic;  -- Reset input
79      CE        : in  std_logic;  -- Chip Enable
80      O         : out std_logic); -- Output
81    end component;

83    component RxUnit
84    port (
85      Clk     : in  std_logic;  -- system clock signal
86      Reset   : in  std_logic;  -- Reset input
87      Enable  : in  std_logic;  -- Enable input
88      ReadA   : in  Std_logic;  -- Async Read Received Byte
89      RxD     : in  std_logic;  -- RS-232 data input
90      RxAv    : out std_logic;  -- Byte available
91      DataO   : out std_logic_vector(7 downto 0)); -- Byte received
92    end component;

94    component TxUnit
95    port (
96      Clk     : in  std_logic;  -- Clock signal
97      Reset   : in  std_logic;  -- Reset input
98      Enable  : in  std_logic;  -- Enable input
99      LoadA   : in  std_logic;  -- Asynchronous Load
100     TxD     : out std_logic;  -- RS-232 data output
101     Busy    : out std_logic;  -- Tx Busy
102     DataI   : in  std_logic_vector(7 downto 0)); -- Byte to transmit
103   end component;

105   signal RxData : std_logic_vector(7 downto 0); -- Last Byte received
106   signal TxData : std_logic_vector(7 downto 0); -- Last bytes transmitted
107   signal SReg   : std_logic_vector(7 downto 0); -- Status register
108   signal EnabRx : std_logic;  -- Enable RX unit
109   signal EnabTx : std_logic;  -- Enable TX unit
110   signal RxAv   : std_logic;  -- Data Received
111   signal TxBusy : std_logic;  -- Transmiter Busy
112   signal ReadA  : std_logic;  -- Async Read receive buffer
113   signal LoadA  : std_logic;  -- Async Load transmit buffer
114   signal Sig0   : std_logic;  -- gnd signal
115   signal Sig1   : std_logic;  -- vcc signal

117   begin
118     sig0 <= '0';
119     sig1 <= '1';

121     Uart_Rxrate : Counter -- Baud Rate adjust
122       generic map (COUNT => BRDIVISOR)
123       port map (BR_CLK_I, sig0, sig1, EnabRx);

125     Uart_Txrate : Counter -- 4 Divider for Tx
126       generic map (COUNT => 4)
127       port map (BR_CLK_I, Sig0, EnabRx, EnabTx);

129     Uart_TxUnit : TxUnit port map (BR_CLK_I, WB_RST_I, EnabTX, LoadA, TxD_PAD_O, TxBusy,
            TxData);
130     Uart_RxUnit : RxUnit port map (BR_CLK_I, WB_RST_I, EnabRX, ReadA, RxD_PAD_I, RxAv,
            RxData);

132     IntTx_O <= not TxBusy;
133     IntRx_O <= RxAv;
134     SReg(0) <= not TxBusy;
135     SReg(1) <= RxAv;
136     SReg(7 downto 2) <= "000000";

138     -- Implements WishBone data exchange.
```

```
139     -- Clocked on rising edge. Synchronous Reset RST_I
140     WBctrl : process(WB_CLK_I, WB_RST_I, WB_STB_I, WB_WE_I, WB_ADR_I)
141     variable StatM : std_logic_vector(4 downto 0);
142     begin
143       if Rising_Edge(WB_CLK_I) then
144         if (WB_RST_I = '1') then
145           ReadA <= '0';
146           LoadA <= '0';
147         else
148           if (WB_STB_I = '1' and WB_WE_I = '1' and WB_ADR_I = "00") then -- Write Byte to
                Tx
149             TxData <= WB_DAT_I;
150             LoadA <= '1';   -- Load signal
151           else
152             LoadA <= '0';
153           end if;
154           if (WB_STB_I = '1' and WB_WE_I = '0' and WB_ADR_I = "00") then -- Read Byte from
                Rx
155             ReadA <= '1';   -- Read signal
156           else
157             ReadA <= '0';
158           end if;
159         end if;
160       end if;
161     end process;

163     WB_ACK_O <= WB_STB_I;
164     WB_DAT_O <=
165       RxData when WB_ADR_I = "00" else  -- Read Byte from Rx
166         SReg when WB_ADR_I = "01" else    -- Read Status Reg
167           "00000000";
168 end Behaviour;
```

C.1.10 Rxunit

```
1  ----------------------------------------------------------------------
2  -- Title      : UART
3  -- Project    : UART
4  ----------------------------------------------------------------------
5  -- File       : Txunit.vhd
6  -- Author     : Philippe CARTON
7  --              (philippe.carton2@libertysurf.fr)
8  -- Organization:
9  -- Created    : 15/12/2001
10 -- Last update : 8/1/2003
11 -- Platform   : Foundation 3.1i
12 -- Simulators : ModelSim 5.5b
13 -- Synthesizers: Xilinx Synthesis
14 -- Targets    : Xilinx Spartan
15 -- Dependency : IEEE std_logic_1164
16 ----------------------------------------------------------------------
17 -- Description: Txunit is a parallel to serial unit transmitter.
18 ----------------------------------------------------------------------
19 -- Copyright (c) notice
20 --    This core adheres to the GNU public license
21 --
22 ----------------------------------------------------------------------
23 -- Revisions      :
24 -- Revision Number :
```

```
25 -- Version          :
26 -- Date     :
27 -- Modifier         : name <email>
28 -- Description      :
29 --
30 ----------------------------------------------------------------------------

32 library ieee;
33 use ieee.std_logic_1164.all;

35 entity TxUnit is
36   port (
37      Clk     : in  std_logic;  -- Clock signal
38      Reset   : in  std_logic;  -- Reset input
39      Enable  : in  std_logic;  -- Enable input
40      LoadA   : in  std_logic;  -- Asynchronous Load
41      TxD     : out std_logic;  -- RS-232 data output
42      Busy    : out std_logic;  -- Tx Busy
43      DataI   : in  std_logic_vector(7 downto 0)); -- Byte to transmit
44 end TxUnit;

46 architecture Behaviour of TxUnit is

48   component synchroniser
49   port (
50      C1 : in std_logic;  -- Asynchronous signal
51      C :  in std_logic;  -- Clock
52      O :  out Std_logic);-- Synchronised signal
53   end component;

55   signal TBuff     : std_logic_vector(7 downto 0); -- transmit buffer
56   signal TReg      : std_logic_vector(7 downto 0); -- transmit register
57   signal TBufL     : std_logic;  -- Buffer loaded
58   signal LoadS     : std_logic;  -- Synchronised load signal

60 begin
61   -- Synchronise Load on Clk
62   SyncLoad : Synchroniser port map (LoadA, Clk, LoadS);
63   Busy <= LoadS or TBufL;

65   -- Tx process
66   TxProc : process(Clk, Reset, Enable, DataI, TBuff, TReg, TBufL)
67   -- variable BitPos : INTEGER range 0 to 10; -- Bit position in the frame
68   variable BitPos : INTEGER range 0 to 11; -- Bit position in the frame
69   begin
70      if Reset = '1' then
71         TBufL <= '0';
72         BitPos := 0;
73         TxD <= '1';
74      elsif Rising_Edge(Clk) then
75         if LoadS = '1' then
76            TBuff <= DataI;
77            TBufL <= '1';
78         end if;
79         if Enable = '1' then
80            case BitPos is
81               when 0 => -- idle or stop bit
82                  TxD <= '1';
83                  if TBufL = '1' then -- start transmit. next is start bit
84                     TReg <= TBuff;
85                     TBufL <= '0';
86                     BitPos := 1;
87                  end if;
```

```
88                when 1 => -- Start bit
89                   TxD <= '0';
90                   BitPos := 2;
91                when 10 =>  -- new
92                   TxD <= '1';
93                   BitPos := BitPos + 1;
94                when others =>
95                   TxD <= TReg(BitPos-2); -- Serialisation of TReg
96                   BitPos := BitPos + 1;
97             end case;
98             -- if BitPos = 10 then -- bit8. next is stop bit
99             if BitPos = 11 then -- bit9. this and next are stop bits
100               BitPos := 0;
101            end if;
102         end if;
103      end if;
104   end process;
105 end Behaviour;
```

C.1.11 Txunit

```
1  ----------------------------------------------------------------------
2  -- Title      : UART
3  -- Project    : UART
4  ----------------------------------------------------------------------
5  -- File       : Rxunit.vhd
6  -- Author     : Philippe CARTON
7  --              (philippe.carton2@libertysurf.fr)
8  -- Organization:
9  -- Created    : 15/12/2001
10 -- Last update : 8/1/2003
11 -- Platform   : Foundation 3.1i
12 -- Simulators : ModelSim 5.5b
13 -- Synthesizers: Xilinx Synthesis
14 -- Targets    : Xilinx Spartan
15 -- Dependency : IEEE std_logic_1164
16 ----------------------------------------------------------------------
17 -- Description: RxUnit is a serial to parallel unit Receiver.
18 ----------------------------------------------------------------------
19 -- Copyright (c) notice
20 --    This core adheres to the GNU public license
21 --
22 ----------------------------------------------------------------------
23 -- Revisions        :
24 -- Revision Number :
25 -- Version          :
26 -- Date     :
27 -- Modifier         : name <email>
28 -- Description      :
29 --
30 ----------------------------------------------------------------------
31 library ieee;
32    use ieee.std_logic_1164.all;

34 entity RxUnit is
35   port (
36      Clk    : in  std_logic;  -- system clock signal
37      Reset  : in  std_logic;  -- Reset input
38      Enable : in  std_logic;  -- Enable input
```

```
39      ReadA   : in  Std_logic;  -- Async Read Received Byte
40      RxD     : in  std_logic;  -- RS-232 data input
41      RxAv    : out std_logic;  -- Byte available
42      DataO   : out std_logic_vector(7 downto 0)); -- Byte received
43  end RxUnit;

45  architecture Behaviour of RxUnit is
46    signal RReg     : std_logic_vector(7 downto 0); -- receive register
47    signal RRegL    : std_logic;                    -- Byte received
48  begin
49    -- RxAv process
50    RxAvProc : process(RRegL,Reset,ReadA)
51    begin
52       if ReadA = '1' or Reset = '1' then
53          RxAv <= '0';  -- Negate RxAv when RReg read
54       elsif Rising_Edge(RRegL) then
55          RxAv <= '1';  -- Assert RxAv when RReg written
56       end if;
57    end process;

59    -- Rx Process
60    RxProc : process(Clk,Reset,Enable,RxD,RReg)
61    -- variable BitPos : INTEGER range 0 to 10;   -- Position of the bit in the frame
62    variable BitPos : INTEGER range 0 to 11;   -- Position of the bit in the frame
63    variable SampleCnt : INTEGER range 0 to 3; -- Count from 0 to 3 in each bit
64    begin
65       if Reset = '1' then -- Reset
66          RRegL <= '0';
67          BitPos := 0;
68       elsif Rising_Edge(Clk) then
69          if Enable = '1' then
70             case BitPos is
71                when 0 => -- idle
72                   RRegL <= '0';
73                   if RxD = '0' then -- Start Bit
74                      SampleCnt := 0;
75                      BitPos := 1;
76                   end if;
77                -- when 10 => -- Stop Bit
78                when 10 => -- first Stop Bit  -- new
79                   if SampleCnt = 3 then -- Increment BitPos on 3
80                      BitPos := BitPos + 1;
81                   end if;
82                when 11 => -- 2 Stop Bits -- it was "when 10"
83                   BitPos := 0;     -- next is idle
84                   RRegL <= '1';    -- Indicate byte received
85                   DataO <= RReg;   -- Store received byte
86                when others =>
87                   if (SampleCnt = 1 and BitPos >= 2) then -- Sample RxD on 1
88                      RReg(BitPos-2) <= RxD; -- Deserialisation
89                      -- RReg(BitPos-3) <= RxD; -- Deserialisation
90                   end if;
91                   if SampleCnt = 3 then -- Increment BitPos on 3
92                      BitPos := BitPos + 1;
93                   end if;
94             end case;
95             if SampleCnt = 3 then
96                SampleCnt := 0;
97             else
98                sampleCnt := SampleCnt + 1;
99             end if;

101         end if;
```

```
102       end if;
103     end process;
104   end Behaviour;
```

C.1.12 utils

```
1   ----------------------------------------------------------------------
2   -- Title      : UART
3   -- Project    : UART
4   ----------------------------------------------------------------------
5   -- File       : utils.vhd
6   -- Author     : Philippe CARTON
7   --              (philippe.carton2@libertysurf.fr)
8   -- Organization:
9   -- Created    : 15/12/2001
10  -- Last update : 8/1/2003
11  -- Platform   : Foundation 3.1i
12  -- Simulators : ModelSim 5.5b
13  -- Synthesizers: Xilinx Synthesis
14  -- Targets    : Xilinx Spartan
15  -- Dependency : IEEE std_logic_1164
16  ----------------------------------------------------------------------
17  -- Description: VHDL utility file
18  ----------------------------------------------------------------------
19  -- Copyright (c) notice
20  --     This core adheres to the GNU public license
21  --
22  ----------------------------------------------------------------------
23  -- Revisions        :
24  -- Revision Number :
25  -- Version          :
26  -- Date     :
27  -- Modifier         : name <email>
28  -- Description      :
29  --
30  ----------------------------------------------------------------------



33  ----------------------------------------------------------------------
34  -- Revision list
35  -- Version   Author                 Date                      Changes
36  --
37  -- 1.0       Philippe CARTON  19 December 2001                New model
38  --       philippe.carton2@libertysurf.fr
39  ----------------------------------------------------------------------

41  ----------------------------------------------------------------------
42  -- Synchroniser:
43  --     Synchronize an input signal (C1) with an input clock (C).
44  --     The result is the O signal which is synchronous of C, and persist for
45  --     one C clock period.
46  ----------------------------------------------------------------------
47  library IEEE,STD;
48  use IEEE.std_logic_1164.all;

50  entity synchroniser is
51     port (
52        C1 : in std_logic;-- Asynchronous signal
53        C :  in std_logic;-- Clock
```

```
54      O :  out std_logic);-- Synchronised signal
55  end synchroniser;

57  architecture Behaviour of synchroniser is
58     signal C1A : std_logic;
59     signal C1S : std_logic;
60     signal R : std_logic;
61  begin
62     RiseC1A : process(C1,R)
63     begin
64        if Rising_Edge(C1) then
65           C1A <= '1';
66        end if;
67        if (R = '1') then
68           C1A <= '0';
69        end if;
70     end process;

72     SyncP : process(C,R)
73     begin
74        if Rising_Edge(C) then
75           if (C1A = '1') then
76              C1S <= '1';
77           else C1S <= '0';
78           end if;
79           if (C1S = '1') then
80              R <= '1';
81           else R <= '0';
82           end if;
83        end if;
84        if (R = '1') then
85           C1S <= '0';
86        end if;
87     end process;
88     O <= C1S;
89  end Behaviour;

91  ----------------------------------------------------------------------------
92  -- Counter
93  --    This counter is a parametrizable clock divider.
94  --    The count value is the generic parameter Count.
95  --    It is CE enabled. (it will count only if CE is high).
96  --    When it overflow, it will emit a pulse on O.
97  --    It can be reseted to 0.
98  ----------------------------------------------------------------------------
99  library IEEE,STD;
100 use IEEE.std_logic_1164.all;

102 entity Counter is
103   generic(Count: INTEGER range 0 to 65535); -- Count revolution
104   port (
105      Clk      : in  std_logic;  -- Clock
106      Reset    : in  std_logic;  -- Reset input
107      CE       : in  std_logic;  -- Chip Enable
108      O        : out std_logic); -- Output
109 end Counter;

111 architecture Behaviour of Counter is
112 begin
113   counter : process(Clk,Reset)
114      variable Cnt : INTEGER range 0 to Count-1;
115   begin
116      if Reset = '1' then
```

```
117         Cnt := Count - 1;
118         O <= '0';
119       elsif Rising_Edge(Clk) then
120         if CE = '1' then
121           if Cnt = 0 then
122             O <= '1';
123             Cnt := Count - 1;
124           else
125             O <= '0';
126             Cnt := Cnt - 1;
127           end if;
128         else O <= '0';
129         end if;
130       end if;
131   end process;
132 end Behaviour;
```

C.1.13 Steuerpfad für MiniUART

```
1  ------------------------------------------------------------
2  --  Tracer - interface between ADC-project and MiniUART
3  --
4  --  Author: A. Arvidsson
5  ------------------------------------------------------------

7  -- <makepackage>
8  --      <component = TRACER>
9  --      <name = arvidsson>
10 --      <project = ada_conv_modsys_master>
11 --       </component>
12 -- </makepackage>

14 library ieee;
15   use ieee.std_logic_1164.all;
16   use ieee.numeric_std.all;

18 library work;
19   use work.global_pkg.all;

21 entity TRACER is
22   port (
23     -- Wishbone signals
24     CLK_I : in  std_logic;  -- clock
25     RST_I : in  std_logic;  -- Reset input
26     ADR_O : out std_logic_vector(1 downto 0); -- Adress bus
27     DAT_I : in  std_logic_vector(7 downto 0); -- DataIn Bus
28     DAT_O : out std_logic_vector(7 downto 0); -- DataOut Bus
29     WE_O  : out std_logic;  -- Write Enable
30     STB_O : out std_logic;  -- Strobe
31     ACK_I : in  std_logic;  -- Acknowledge
32     --- process signals --- non-wishbone
33     IntTx_I  : in std_logic;  -- Transmit interrupt: indicate waiting for Byte
34     IntRx_I  : in std_logic;  -- Receive interrupt: indicate Byte received
35     --- extra ---
36     DataFull       : in bit; -- doch lieber mit DataReady (s.u.) ersetzen??
37     rsGetEnableOut  : out bit;
38     SendingDone    : in bit;
39     ---------- timeout -------
40     StartTimer   : out bit;
```

```
41      StopTimer    : out bit;
42      TimeOut      : in bit;
43      ----------
44      ProgReg   : out std_logic_vector(3 downto 0);
45      ProgInfo  : out std_logic_vector(11 downto 0);
46      ProgRegReset   : in bit;
47      newProgReg   : out bit;
48      DataWrite : in std_logic_vector(7 downto 0);
49      --
50      checkNewData : out bit;
51      DataCntCheck : out bit;
52      DataReady    : in bit
53    );
54  end TRACER;

56  architecture TRACE of TRACER is

58  signal DatReceive : std_logic_vector(7 downto 0);
59  signal count    : unsigned(1 downto 0);
60  signal waitCount : unsigned(1 downto 0);
61  signal ReceiveData : std_logic_vector(7 downto 0);
62  signal rsGetDone  : bit;

64  signal ProgRegX : std_logic_vector(3 downto 0);

66  signal idxRcv : integer range 0 to 4;
67  signal idxRcvX : integer range 0 to 4;
68  signal idxRcvAdd : bit;
69  signal idxRcvRST : bit;

71  signal Rcounter : unsigned(2 downto 0);

73  signal rsSendEnable   : bit;
74  signal rsGetEnable    : bit;
75  signal getData   : bit;
76  signal analyzed : bit;
77  signal checkEnable : bit;
78  signal dataCheckOK : bit;

80  type STATE_TYPE is (Decider,Decider2,
81                      WR0,WRcheck,WR1,WR2,WR3,
82                      R0,R1,R2,R3,R4
83                      );
84  signal STATE : STATE_TYPE;
85  type stateEvaluationType is (WaitForData,DataCheck,receiveAll,waitPart,allReceived,waitDone
        ,nextWork,waitDataFull,waitForSent,enableReceive);
86  signal stateEvaluation : stateEvaluationType;
87  signal firstRun : bit;

89  begin

91    CONTR : process(RST_I,CLK_I)
92    begin
93      if RST_I = '1' then
94        ADR_O <= "00" after 9 ns; -- address for transmit/receive buffer
95        DAT_O <= x"00" after 9 ns;
96        WE_O <= '0' after 9 ns;
97        STB_O <= '0' after 9 ns;

99        STATE <= R0 after 9 ns;
100       checkNewData <= '0' after 9 ns;
101       DataCntCheck <= '0' after 9 ns;
```

```
103         idxRcvAdd <= '0' after 9 ns;
104         idxRcvRST <= '0' after 9 ns;
105         getData <= '0' after 9 ns;
106         checkEnable <= '0' after 9 ns;

108         waitCount <= (others => '0') after 9 ns;
109         count <= (others => '0') after 9 ns;

111         StartTimer <= '0' after 9 ns;
112         StopTimer <= '0' after 9 ns;

114       elsif CLK_I = '1' and CLK_I'event then
115         ADR_O <= "00" after 9 ns;
116         DAT_O <= x"00" after 9 ns;
117         WE_O <= '0' after 9 ns;
118         STB_O <= '0' after 9 ns;

120         STATE <= R0 after 9 ns;
121         checkNewData <= '0' after 9 ns;
122         DataCntCheck <= '0' after 9 ns;

124         idxRcvAdd <= '0' after 9 ns;
125         idxRcvRST <= '0' after 9 ns;
126         getData <= '0' after 9 ns;
127         checkEnable <= '0' after 9 ns;

129         StartTimer <= '0' after 9 ns;
130         StopTimer <= '0' after 9 ns;

132         case STATE is
133         when Decider =>
134           if analyzed = '1' then
135             STATE <= Decider2 after 9 ns;
136           elsif waitCount = 2 then
137             waitCount <= (others => '0') after 9 ns;
138             STATE <= Decider2 after 9 ns;
139           else
140             waitCount <= waitCount + 1 after 9 ns;
141             STATE <= Decider after 9 ns;
142           end if;

144         when Decider2 =>
145           if rsGetEnable = '1' then
146             if firstRun = '0' then  -- StartTimer darf nur gesetzt werden, wenn es nicht der
                    erste Durchlauf ist, da sonst der TimeOut ausgelöst wird!!
147               StartTimer <= '1' after 9 ns;
148             end if;
149             STATE <= R0 after 9 ns;
150           elsif rsSendEnable = '1' then
151             STATE <= WR0 after 9 ns;
152           else
153             STATE <= Decider2 after 9 ns;
154           end if;

156         ---- Write to Matlab ----
157         when WR0 =>
158           checkNewData <= '1' after 9 ns;
159           count <= (others => '0') after 9 ns;
160           STATE <= WRcheck after 9 ns;
161         when WRcheck =>
162           if count = "10" then
163             STATE <= WR1 after 9 ns;
164           else
```

```
165           count <= count + 1 after 9 ns;
166           STATE <= WRcheck after 9 ns;
167         end if;
168       when WR1 =>
169         STATE <= WR1 after 9 ns;
170         if IntTx_I = '1' then
171           STATE <= WR2 after 9 ns;
172         end if;
173       when WR2 =>
174         WE_O <= '1' after 9 ns;
175         ADR_O <= "00" after 9 ns;
176         DAT_O <= DataWrite after 9 ns;
177         STB_O <= '1' after 9 ns;
178         STATE <= WR2 after 9 ns;
179         if ACK_I = '1' then
180           STATE <= WR3 after 9 ns;
181         end if;
182       when WR3 =>
183         DataCntCheck <= '1' after 9 ns;
184         STATE <= Decider after 9 ns;

186       ---- Read from Matlab ----
187       when R0 =>
188         if TimeOut = '1' then
189           STATE <= R0 after 9 ns;
190         elsif IntRx_I = '1' then
191           StopTimer <= '1' after 9 ns;
192           STATE <= R1 after 9 ns;
193         else
194           STATE <= R0 after 9 ns;
195         end if;
196       when R1 =>
197         WE_O <= '0' after 9 ns;
198         ADR_O <= "00" after 9 ns;
199         STB_O <= '1' after 9 ns;
200         STATE <= R1 after 9 ns;
201         if ACK_I = '1' then
202           DatReceive <= DAT_I after 9 ns;
203           getData <= '1' after 9 ns;
204           STATE <= R2 after 9 ns;
205         end if;
206       when R2 =>
207         Rcounter <= (others => '0') after 9 ns;
208         checkEnable <= '1' after 9 ns;
209         STATE <= R3 after 9 ns;
210       when R3 =>
211         if Rcounter = 1 then
212           STATE <= R4 after 9 ns;
213         else
214           Rcounter <= Rcounter + 1 after 9 ns;
215           STATE <= R3 after 9 ns;
216         end if;
217       when R4 =>
218         if dataCheckOK = '1' then
219           if idxRcv = 4 then
220             idxRcvRST <= '1' after 9 ns;
221           else
222             idxRcvAdd <= '1' after 9 ns;
223           end if;
224         else
225           idxRcvRST <= '1' after 9 ns;
226         end if;
227         STATE <= Decider after 9 ns;
```

```
228         ------------------------------------------
229       end case;

231     end if; -- RST_I and CLK_I and CLK_I'event
232   end process CONTR;

234   idxCounter : process(RST_I, CLK_I)
235   begin
236     if RST_I = '1' then
237       idxRcvX <= 0 after 9 ns;
238     elsif CLK_I = '1' and CLK_I'event then
239       if idxRcvRST = '1' then
240         idxRcvX <= 0 after 9 ns;
241       elsif idxRcvAdd = '1' then
242         if idxRcv = 4 then
243           idxRcvX <= 0 after 9 ns;
244         else
245           idxRcvX <= idxRcvX + 1 after 9 ns;
246         end if;
247       end if;
248     end if;
249   end process idxCounter;

251   idxRcv <= idxRcvX;

253   DatAuswertung : process(CLK_I, RST_I)
254   begin
255     if RST_I = '1' then
256       rsGetDone <= '0' after 9 ns;
257       rsGetEnable <= '1' after 9 ns;
258       rsSendEnable <= '0' after 9 ns;

260       stateEvaluation <= WaitForData after 9 ns;
261       firstRun <= '1' after 9 ns;

263     elsif CLK_I = '1' and CLK_I'event then
264       rsGetDone <= '0' after 9 ns;
265       if TimeOut = '1' then
266         stateEvaluation <= WaitForData after 9 ns;
267       else
268         case stateEvaluation is
269           when WaitForData =>
270             firstRun <= '1' after 9 ns;
271             if checkEnable = '1' then
272               stateEvaluation <= DataCheck after 9 ns;
273             else
274               stateEvaluation <= WaitForData after 9 ns;
275             end if;
276           when DataCheck =>
277             if DatReceive = x"23" then
278               dataCheckOK <= '1' after 9 ns;
279               stateEvaluation <= receiveAll after 9 ns;
280             else
281               stateEvaluation <= DataCheck after 9 ns;
282             end if;
283           when receiveAll =>
284             firstRun <= '0' after 9 ns;
285             if idxRcv = 4 then
286               stateEvaluation <= waitPart after 9 ns;
287             else
288               stateEvaluation <= receiveAll after 9 ns;
289             end if;
290           when waitPart =>  -- to get received information written into DatReceive
```

```
291             if idxRcv = 0 then
292               stateEvaluation <= allReceived after 9 ns;
293             else
294               stateEvaluation <= waitPart after 9 ns;
295             end if;
296           when allReceived =>
297             dataCheckOK <= '0' after 9 ns;
298             rsGetDone <= '1' after 9 ns;
299             rsGetEnable <= '0' after 9 ns;
300             stateEvaluation <= waitDone after 9 ns;
301           when waitDone =>
302             if analyzed = '1' then
303               stateEvaluation <= nextWork after 9 ns;
304             else
305               stateEvaluation <= waitDone after 9 ns;
306             end if;
307           when nextWork => -- to let information written into ReceiveArray
308             if ProgRegX = x"F" or ProgRegX = x"C" or ProgRegX = x"D" then
309               stateEvaluation <= waitDataFull after 9 ns;
310             else
311               stateEvaluation <= enableReceive after 9 ns;
312             end if;
313           when waitDataFull =>
314             rsGetEnable <= '0' after 9 ns;
315             if DataFull = '1' or DataReady = '1' then
316               stateEvaluation <= waitForSent after 9 ns;
317             else
318               stateEvaluation <= waitDataFull after 9 ns;
319             end if;
320           when waitForSent => -- wait for full data array
321             rsSendEnable <= '1' after 9 ns;
322             rsGetEnable <= '0' after 9 ns;
323             stateEvaluation <= waitForSent after 9 ns;
324             if SendingDone = '1' then
325               stateEvaluation <= enableReceive after 9 ns;
326             end if;
327           when enableReceive =>
328             rsGetEnable <= '1' after 9 ns;
329             rsSendEnable <= '0' after 9 ns;
330             stateEvaluation <= WaitForData after 9 ns;
331           when others => null;
332         end case;
333       end if; -- TimeOut
334     end if; -- CLK_I and RST_I
335   end process DatAuswertung;

337   ProgReg          <= ProgRegX;
338   rsGetEnableOut   <= rsGetEnable;

340   DataProg : process(RST_I,CLK_I)
341   begin
342     if RST_I = '1' then
343       ProgRegX <= (others => '0') after 9 ns;
344       ProgInfo <= (others => '0') after 9 ns;
345       analyzed <= '1' after 9 ns;
346       ReceiveData <= (others => '0') after 9 ns;
347     elsif CLK_I = '1' and CLK_I'event then
348       if SendingDone = '1' or ProgRegReset = '1' then
349         ProgRegX <= (others => '0') after 9 ns;
350       end if;
351       if rsGetDone = '1' then
352         ProgRegX  <= ReceiveData(7 downto 4) after 9 ns;
353         ProgInfo  <= ReceiveData(3 downto 0) & DatReceive after 9 ns;
```

```
354         analyzed <= '1' after 9 ns;
355       else
356         analyzed <= '0' after 9 ns;
357       end if;
358       if getData = '1' then
359         if idxRcv = 3 then
360           ReceiveData <= DatReceive after 9 ns;
361         end if;
362       end if;
363     end if;
364   end process DataProg;

366   newProgReg <= analyzed;

368 end TRACE;
```

C.1.14 Datenpfad für MiniUART

```
1  ------------------------------------------------------------------
2  --  Tracer - interface between ADC-project and MiniUART
3  --
4  --  Author: A.Arvidsson
5  ------------------------------------------------------------------

7  -- <makepackage>
8  --      <component = TracerData>
9  --      <name = arvidsson>
10 --      <project = ada_conv_modsys_master>
11 --       </component>
12 -- </makepackage>

14 library ieee;
15   use ieee.std_logic_1164.all;
16   use ieee.numeric_std.all;

18 library work;
19   use work.global_pkg.all;

21 entity TracerData is
22   port (
23     CLK           : in  std_logic;
24     RESET         : in  std_logic;
25     ---
26     ProgReg       : in std_logic_vector(3 downto 0);
27     ProgInfoPart  : in std_logic_vector(3 downto 0);
28     --- extra ---
29     ArrayUdiff    : in DataArray;
30     ArrayHB1      : in DataArray;
31     ArrayHB2      : in DataArray;
32     SampleIntervall : in unsigned((SampleIntervallMax - 1) downto 0);
33       -- DAC --
34     ref_diff1     : in std_logic_vector(11 downto 0);
35     ref_diff2     : in std_logic_vector(11 downto 0);
36     ref_hb1       : in std_logic_vector(11 downto 0);
37     ref_hb2       : in std_logic_vector(11 downto 0);
38     u_broff1      : in std_logic_vector(11 downto 0);
39     u_broff2      : in std_logic_vector(11 downto 0);
40     u_off         : in std_logic_vector(11 downto 0);
41     u_opt         : in std_logic_vector(11 downto 0);
```

```
42        -- ADC --
43      ADCrefOut      : in std_logic_vector(11 downto 0);
44      ADCsequence    : in std_logic_vector(11 downto 0);
45      AMPgain        : in std_logic_vector( 2 downto 0);
46      ---
47      DataCntCheck   : in bit;
48      checkNewData   : in bit;
49      DataWrite      : out std_logic_vector(7 downto 0);
50      MatlabDone     : out bit;
51      SendingDone    : out bit;
52      ---
53      DataCnt_RST    : out bit;
54      DataCnt_End    : out bit;
55      DataCntBack    : out bit;
56      DataCnt_Add    : out bit;
57      DataCntLoop    : out bit;
58      DataCnt        : in  unsigned(3 downto 0)  -- es werden 5 Bytes entgegengenommen bzw.
            verschickt
59    );
60  end TracerData;

62  architecture behaviour of TracerData is
63  signal Byte1, Byte3, Byte5 : std_logic_vector(3 downto 0);
64  signal Byte2, Byte4, Byte6 : std_logic_vector(7 downto 0);
65  signal P12_EN : bit;
66  signal count : unsigned(1 downto 0);
67  signal index : integer range 0 to ArrayMax;

69  begin

71    RegData : process(RESET,CLK)
72    begin
73      if RESET = '1' then
74        Byte1 <= x"0" after 9 ns;
75        Byte2 <= x"00" after 9 ns;
76        Byte3 <= x"0" after 9 ns;
77        Byte4 <= x"00" after 9 ns;
78        Byte5 <= x"0" after 9 ns;
79        Byte6 <= x"00" after 9 ns;
80        count <= "00" after 9 ns;
81      elsif CLK = '1' and CLK'event then
82        if P12_EN = '1' then
83          Byte1 <= x"0" after 9 ns;
84          Byte2 <= x"00" after 9 ns;
85          Byte3 <= x"0" after 9 ns;
86          Byte4 <= x"00" after 9 ns;
87          Byte5 <= x"0" after 9 ns;
88          Byte6 <= x"00" after 9 ns;
89          case ProgReg is
90            when x"C" =>
91              case count is
92                when "00" =>
93                  Byte1 <= ref_diff1(11 downto 8) after 9 ns;
94                  Byte2 <= ref_diff1(7 downto 0) after 9 ns;
95                  Byte3 <= ref_diff2(11 downto 8) after 9 ns;
96                  Byte4 <= ref_diff2(7 downto 0) after 9 ns;
97                  Byte5 <= ref_hb1(11 downto 8) after 9 ns;
98                  Byte6 <= ref_hb1(7 downto 0) after 9 ns;
99                  count <= "01" after 9 ns;
100               when "01" =>
101                 Byte1 <= ref_hb2(11 downto 8) after 9 ns;
102                 Byte2 <= ref_hb2(7 downto 0) after 9 ns;
103                 Byte3 <= u_broff1(11 downto 8) after 9 ns;
```

```
104                 Byte4 <= u_broff1(7 downto 0) after 9 ns;
105                 Byte5 <= u_broff2(11 downto 8) after 9 ns;
106                 Byte6 <= u_broff2(7 downto 0) after 9 ns;
107                 count <= "10" after 9 ns;
108               when "10" =>
109                 Byte1 <= u_off(11 downto 8) after 9 ns;
110                 Byte2 <= u_off(7 downto 0) after 9 ns;
111                 Byte3 <= u_opt(11 downto 8) after 9 ns;
112                 Byte4 <= u_opt(7 downto 0) after 9 ns;
113                 Byte5 <= x"0" after 9 ns;
114                 Byte6 <= "00000" & AMPgain(2 downto 0) after 9 ns;
115                 count <= "11" after 9 ns;
116               when "11" =>
117                 Byte1 <= ADCrefOut(11 downto 8) after 9 ns;
118                 Byte2 <= ADCrefOut(7 downto 0) after 9 ns;
119                 Byte3 <= ADCsequence(11 downto 8) after 9 ns;
120                 Byte4 <= ADCsequence(7 downto 0) after 9 ns;
121                 count <= "00" after 9 ns;
122               when others => null;
123             end case;
124           when x"D" =>
125             case ProgInfoPart is
126               when x"1" => Byte1 <= ref_diff1(11 downto 8) after 9 ns;
127                            Byte2 <= ref_diff1(7 downto 0) after 9 ns;
128               when x"2" => Byte1 <= ref_diff2(11 downto 8) after 9 ns;
129                            Byte2 <= ref_diff2(7 downto 0) after 9 ns;
130               when x"3" => Byte1 <= ref_hb1(11 downto 8) after 9 ns;
131                            Byte2 <= ref_hb1(7 downto 0) after 9 ns;
132               when x"4" => Byte1 <= ref_hb2(11 downto 8) after 9 ns;
133                            Byte2 <= ref_hb2(7 downto 0) after 9 ns;
134               when x"5" => Byte1 <= u_broff1(11 downto 8) after 9 ns;
135                            Byte2 <= u_broff1(7 downto 0) after 9 ns;
136               when x"6" => Byte1 <= u_broff2(11 downto 8) after 9 ns;
137                            Byte2 <= u_broff2(7 downto 0) after 9 ns;
138               when x"7" => Byte1 <= u_off(11 downto 8) after 9 ns;
139                            Byte2 <= u_off(7 downto 0) after 9 ns;
140               when x"8" => Byte1 <= u_opt(11 downto 8) after 9 ns;
141                            Byte2 <= u_opt(7 downto 0) after 9 ns;
142               when x"9" => Byte1 <= x"0" after 9 ns;
143                            Byte2 <= "00000" & AMPgain(2 downto 0) after 9 ns;
144               when x"A" => Byte1 <= ADCrefOut(11 downto 8) after 9 ns;
145                            Byte2 <= ADCrefOut(7 downto 0) after 9 ns;
146               when x"B" => Byte1 <= ADCsequence(11 downto 8) after 9 ns;
147                            Byte2 <= ADCsequence(7 downto 0) after 9 ns;
148               when others => null;
149             end case;
150           when x"F" =>
151             Byte1 <= ArrayUdiff(index)(11 downto 8) after 9 ns;  -- Part1
152             Byte2 <= ArrayUdiff(index)(7 downto 0) after 9 ns;          -- Part2
153             Byte3 <= ArrayHB1(index)(11 downto 8) after 9 ns;
154             Byte4 <= ArrayHB1(index)(7 downto 0) after 9 ns;
155             Byte5 <= ArrayHB2(index)(11 downto 8) after 9 ns;
156             Byte6 <= ArrayHB2(index)(7 downto 0) after 9 ns;
157           when others => null;
158         end case;
159       end if;
160     end if;
161   end process RegData;

163   DatAllocation : process(RESET,CLK)
164   begin
165     if RESET = '1' then
166       DataWrite <= (others => '0') after 9 ns;
```

```
167     elsif CLK = '1' and CLK'event then
168       case ProgReg is
169         when x"C" =>
170           case DataCnt is
171             when x"0" => DataWrite <= x"23" after 9 ns;
172             when x"1" | x"2" | x"3" => DataWrite <= x"32" after 9 ns;
173             when x"4" => DataWrite <= x"0" & Byte1 after 9 ns;
174             when x"5" => DataWrite <= Byte2 after 9 ns;
175             when x"6" => DataWrite <= x"0" & Byte3 after 9 ns;
176             when x"7" => DataWrite <= Byte4 after 9 ns;
177             when x"8" => DataWrite <= x"0" & Byte5 after 9 ns;
178             when x"9" => DataWrite <= Byte6 after 9 ns;
179             when x"A" => DataWrite <= x"0A" after 9 ns;  -- line-feed
180             when others => DataWrite <= x"00" after 9 ns;
181           end case; -- DataCnt
182         when x"D" =>
183           case DataCnt is
184             when x"0" => DataWrite <= x"23" after 9 ns;
185             when x"1" => DataWrite <= x"31" after 9 ns;
186             when x"2" => DataWrite <= x"32" after 9 ns;
187             when x"3" => DataWrite <= x"0" & Byte1 after 9 ns;
188             when x"4" => DataWrite <= Byte2 after 9 ns;
189             when x"5" => DataWrite <= x"0A" after 9 ns;
190             when others => DataWrite <= x"00" after 9 ns;
191           end case; -- DataCnt
192         when x"F" =>
193           case DataCnt is
194             when x"0" => DataWrite <= x"23" after 9 ns;
195             when x"1" => DataWrite <= x"33" after 9 ns;
196             when x"2" => DataWrite <= x"33" after 9 ns;
197             when x"3" => DataWrite <= x"38" after 9 ns;
198             when x"4" => DataWrite <= x"37" after 9 ns;
199             when x"5" => DataWrite <= x"0" & Byte1 after 9 ns;
200             when x"6" => DataWrite <= Byte2 after 9 ns;
201             when x"7" => DataWrite <= x"0" & Byte3 after 9 ns;
202             when x"8" => DataWrite <= Byte4 after 9 ns;
203             when x"9" => DataWrite <= x"0" & Byte5 after 9 ns;
204             when x"A" => DataWrite <= Byte6 after 9 ns;
205             when x"B" => DataWrite <= "000000" & std_logic_vector(SampleIntervall((
                    SampleIntervallMax - 1) downto 16)) after 9 ns;
206             when x"C" => DataWrite <= std_logic_vector(SampleIntervall(15 downto 8)) after
                    9 ns;
207             when x"D" => DataWrite <= std_logic_vector(SampleIntervall( 7 downto 0)) after
                    9 ns;
208             when x"E" => DataWrite <= x"0A" after 9 ns;
209             when others => DataWrite <= x"00" after 9 ns;
210           end case; -- DataCnt
211         when others => null;
212       end case; -- ProgReg
213     end if; -- RESET and CLK
214   end process DatAllocation;

216   SendCtrl : process(RESET,CLK)
217   begin
218     if RESET = '1' then
219       index <= 0 after 9 ns;
220       DataCnt_Add <= '0' after 9 ns;
221       DataCnt_RST <= '1' after 9 ns;
222       DataCnt_End <= '0' after 9 ns;
223       DataCntBack <= '0' after 9 ns;
224       DataCntLoop <= '0' after 9 ns;
225       MatlabDone <= '0' after 9 ns;
226       SendingDone <= '0' after 9 ns;
```

```
227       P12_EN <= '0' after 9 ns;
228     elsif CLK = '1' and CLK'event then
229       DataCnt_Add <= '0' after 9 ns;
230       DataCnt_RST <= '0' after 9 ns;
231       DataCnt_End <= '0' after 9 ns;
232       DataCntBack <= '0' after 9 ns;
233       DataCntLoop <= '0' after 9 ns;
234       MatlabDone <= '0' after 9 ns;
235       SendingDone <= '0' after 9 ns;
236       P12_EN <= '0' after 9 ns;

238       if checkNewData = '1' then
239         case ProgReg is
240           when x"C" =>
241             if DataCnt = x"4" then
242               P12_EN <= '1' after 9 ns;
243             end if;
244           when x"D" =>
245             if DataCnt = x"3" then
246               P12_EN <= '1' after 9 ns;
247             end if;
248           when x"F" =>
249             if DataCnt = x"5" then
250               P12_EN <= '1' after 9 ns;
251             end if;
252           when others => null;
253         end case;
254       elsif DataCntCheck = '1' then
255         case ProgReg is
256           when x"C" =>
257                 if DataCnt = x"7" and index = 3 then
258                   DataCnt_End <= '1' after 9 ns;
259                 elsif DataCnt = x"9" and index < 3 then
260                     index <= index + 1 after 9 ns;
261                     DataCntBack <= '1' after 9 ns;
262                 elsif DataCnt = x"A" then
263                   SendingDone <= '1' after 9 ns;
264                   index <= 0 after 9 ns;
265                   DataCnt_RST <= '1' after 9 ns;
266                 else
267                   DataCnt_Add <= '1' after 9 ns;
268                 end if;
269           when x"D" =>
270                 if DataCnt = x"5" then
271                   SendingDone <= '1' after 9 ns;
272                   DataCnt_RST <= '1' after 9 ns;
273                 else
274                   DataCnt_Add <= '1' after 9 ns;
275                 end if;
276           when x"F" =>
277                 if DataCnt = x"A" then
278                   if index < ArrayMax then
279                     index <= index + 1 after 9 ns;
280                     DataCntLoop <= '1' after 9 ns;
281                   else
282                     DataCnt_Add <= '1' after 9 ns;
283                   end if;
284                 elsif DataCnt = x"E" then
285                   DataCnt_RST <= '1' after 9 ns;
286                   index <= 0 after 9 ns;
287                   MatlabDone <= '1' after 9 ns;
288                   SendingDone <= '1' after 9 ns;
289                 else
```

```
290                   DataCnt_Add <= '1' after 9 ns;
291                 end if;
292               when others => null;
293             end case;
294           end if; -- DataCntCheck
295         end if; -- RESET and CLK
296       end process SendCtrl;

298   end behaviour;
```

C.1.15 Zähler für MiniUART

```
1   ------------------------------------------------------------------
2   --  Tracer - interface between ADC-project and MiniUART
3   --
4   --  Author: A. Arvidsson
5   ------------------------------------------------------------------

7   -- <makepackage>
8   --      <component = DataCount>
9   --      <name = arvidsson>
10  --      <project = ada_conv_modsys_master>
11  --       </component>
12  -- </makepackage>

14  library ieee;
15    use ieee.std_logic_1164.all;
16    use ieee.numeric_std.all;

18  library work;
19    use work.global_pkg.all;

21  entity DataCount is
22    port(
23      CLK          : in std_logic;
24      RESET        : in std_logic;
25      DataCnt_RST  : in bit;
26      DataCnt_End  : in bit;
27      DataCntBack  : in bit;
28      DataCnt_Add  : in bit;
29      DataCntLoop  : in bit;
30      DataCnt      : out unsigned(3 downto 0)
31      );
32  end DataCount;

34  architecture behaviour of DataCount is
35  signal X_DataCnt : unsigned(3 downto 0);

37  begin
38    COUNT : process(CLK,RESET,DataCnt_RST)
39    begin
40      if RESET = '1' or DataCnt_RST = '1' then
41        X_DataCnt <= x"0" after 9 ns; -- zur Synchronisierung
42      elsif CLK = '1' and CLK'event then
43        if DataCnt_Add = '1' then
44          X_DataCnt <= X_DataCnt + 1 after 9 ns;
45        elsif DataCntLoop = '1' then
46          X_DataCnt <= x"5" after 9 ns;
47        elsif DataCnt_End = '1' then
```

```
48          X_DataCnt <= x"A" after 9 ns;
49        elsif DataCntBack = '1' then
50          X_DataCnt <= x"4" after 9 ns;
51        end if;
52      end if;
53    end process COUNT;

55    DataCnt <= X_DataCnt;

57  end behaviour;
```

C.1.16 Speicherung der drei Signale in Arrays

```
1   ----------------------------------------------------------------------
2   --  ADC-Array
3   --
4   --  Author: A. Arvidsson
5   ----------------------------------------------------------------------

7   -- <makepackage>
8   --   <global>
9   --     constant ArrayMax : integer := 64 - 1; -- 64
10  --     type DataArray is array(ArrayMax downto 0) of std_logic_vector(11 downto 0);
11  --   </global>
12  --     <component = adc_array>
13  --     <name = arvidsson>
14  --     <project = ada_conv_modsys_master>
15  --      </component>
16  -- </makepackage>

18  library ieee;
19    use ieee.std_logic_1164.all;
20    use ieee.numeric_std.all;

22  library work;
23    use work.global_pkg.all;

25  entity adc_array is
26    port (
27      CLK    : in std_logic;
28      RESET  : in std_logic;
29      --
30      DB_UDIFF  : in  std_logic_vector(11 downto 0);  -- to be sent to Matlab
31      DB_HB1    : in  std_logic_vector(11 downto 0);
32      DB_HB2    : in  std_logic_vector(11 downto 0);
33      --
34      ArrayUdiff  : out DataArray;
35      ArrayHB1    : out DataArray;
36      ArrayHB2    : out DataArray;
37      DataFull    : out bit;
38      saveInArray : in bit;
39      MatlabDone  : in  bit;
40      ProgReg     : in  std_logic_vector(3 downto 0);
41      --
42      --- from periodCalc to always get one period
43      SampleIntervall : in unsigned ((SampleIntervallMax - 1) downto 0)
44    );
45  end adc_array;
```

```
47 architecture behaviour of adc_array is
48 signal index : integer range 0 to ArrayMax;
49 signal XDataFull : bit;
50 signal counter : unsigned ((SampleIntervallMax - 1) downto 0);

52 begin

54   ValueSave : process(RESET,CLK)
55   variable i : integer range 0 to ArrayMax;
56   begin
57     if RESET = '1' then
58       index <= 0 after 9 ns;
59       XDataFull <= '0' after 9 ns;
60       for i in 0 to ArrayMax loop
61         ArrayUdiff(i) <= (others => '0') after 9 ns;
62         ArrayHB1(i)   <= (others => '0') after 9 ns;
63         ArrayHB2(i)   <= (others => '0') after 9 ns;
64       end loop;
65       counter <= (others => '0') after 9 ns;
66     elsif CLK = '1' and CLK'event then
67       if MatlabDone = '1' then
68         index <= 0 after 9 ns;
69         XDataFull <= '0' after 9 ns;
70         counter <= (others => '0') after 9 ns;
71       else
72         if ProgReg = x"F" then
73           if XDataFull = '0' and saveInArray = '1' then
74             if counter = SampleIntervall then
75               ArrayUdiff(index) <= DB_UDIFF after 9 ns;
76               ArrayHB1(index)   <= DB_HB1 after 9 ns;
77               ArrayHB2(index)   <= DB_HB2 after 9 ns;
78               counter <= (others => '0') after 9 ns;
79               if XDataFull = '0' and index < ArrayMax then
80                 index <= index + 1 after 9 ns;
81               end if;
82               if index = ArrayMax then
83                 XDataFull <= '1' after 9 ns;
84               end if; -- index = ArrayMax
85             else
86               counter <= counter + 1 after 9 ns;
87             end if; -- counter = SampleIntervall
88           end if; -- XDataFull = 0 and counter = SampleIntervall
89         end if; -- ProgReg
90       end if; -- MatlabDone = 1
91     end if; -- RESET and CLK
92   end process ValueSave;

94   DataFull    <= XDataFull;

96 end behaviour;
```

C.1.17 Ausgabe der drei Arrays

```
1 ------------------------------------------------------------
2 --  ArrayTest to check information in array
3 --
4 --  Author: A. Arvidsson
5 ------------------------------------------------------------
```

```
7  -- <makepackage>
8  --      <component = ArrayTest>
9  --      <name = arvidsson>
10 --      <project = ada_conv_modsys>
11 --       </component>
12 -- </makepackage>

14 library ieee;
15   use ieee.std_logic_1164.all;
16   use ieee.numeric_std.all;

18 library work;
19   use work.global_pkg.all;

21 entity ArrayTest is
22   port (
23     CLK    : in std_logic;
24     RESET : in std_logic;

26     BtnNextArray  : in bit;
27     BtnBackValue  : in bit;
28     BtnNextValue  : in bit;

30     ArrayUdiff : in DataArray;
31     ArrayHB1   : in DataArray;
32     ArrayHB2   : in DataArray;

34     ArrayValue : out std_logic_vector(11 downto 0)
35   );
36 end ArrayTest;

38 architecture behaviour of ArrayTest is
39 signal index : integer range 0 to 63;
40 signal ActualArrayNum : bit_vector(1 downto 0);
41 signal count : bit;
42 signal cntNextVal : bit;
43 signal cntBackVal : bit;

45 begin

47   array_jump : process(RESET,CLK)
48   begin
49     if RESET = '1' then
50       ActualArrayNum <= "00" after 9 ns;
51     elsif CLK = '1' and CLK'event then
52       if BtnNextArray = '1' and count = '0' then
53         count <= '1' after 9 ns;
54         case ActualArrayNum is
55           when "00" => ActualArrayNum <= "01" after 9 ns;
56           when "01" => ActualArrayNum <= "10" after 9 ns;
57           when "10" => ActualArrayNum <= "00" after 9 ns;
58           when others => null;
59         end case;
60       elsif BtnNextArray = '0' then
61         count <= '0' after 9 ns;
62       end if;
63     end if; -- RESET and CLK
64   end process array_jump;

66   value_jump : process(RESET,CLK)
67   begin
68     if RESET = '1' then
69       cntNextVal <= '0' after 9 ns;
```

```
70        cntBackVal <= '0' after 9 ns;
71        index <= 0 after 9 ns;
72      elsif CLK = '1' and CLK'event then
73        if BtnNextValue = '1' and cntNextVal = '0' then
74          cntNextVal <= '1' after 9 ns;
75          cntBackVal <= '0' after 9 ns;
76          if index = 63 then
77            index <= 0 after 9 ns;
78          else
79            index <= index + 1 after 9 ns;
80          end if;
81        elsif BtnBackValue = '1' and cntBackVal = '0' then
82          cntNextVal <= '0' after 9 ns;
83          cntBackVal <= '1' after 9 ns;
84          if index = 0 then
85            index <= 63 after 9 ns;
86          else
87            index <= index - 1 after 9 ns;
88          end if;
89        end if;
90        if BtnNextValue = '0' then
91          cntNextVal <= '0' after 9 ns;
92        end if;
93        if BtnBackValue = '0' then
94          cntBackVal <= '0' after 9 ns;
95        end if;
96      end if; -- RESET and CLK
97    end process value_jump;

99    valueOut : process(RESET,CLK)
100   begin
101     if RESET = '1' then
102       ArrayValue <= x"000" after 9 ns;
103     elsif CLK = '1' and CLK'event then
104       case ActualArrayNum is
105         when "00" => ArrayValue <= ArrayUdiff(index) after 9 ns;
106         when "01" => ArrayValue <= ArrayHB1(index) after 9 ns;
107         when "10" => ArrayValue <= ArrayHB2(index) after 9 ns;
108         when others => null;
109       end case;
110     end if;
111   end process valueOut;

113 end behaviour;
```

C.1.18 Timeout für MiniUART

```
1  ----------------------------------------------------------------
2  --  Timeout for RS232-Interface
3  --
4  --  Author: A. Arvidsson
5  ----------------------------------------------------------------

7  -- <makepackage>
8  --    <global>
9  --      constant timerMax : unsigned(16 downto 0) := "11110100001001000";    ---- 10 ms =^
       125000
10 --    </global>
11 --     <component = TimeoutCnt>
```

```
12 --      <name = arvidsson>
13 --      <project = ada_conv_modsys_master>
14 --      </component>
15 -- </makepackage>

17 library ieee;
18   use ieee.std_logic_1164.all;
19   use ieee.numeric_std.all;

21 library work;
22   use work.global_pkg.all;

24 entity TimeoutCnt is
25   port (
26     CLK    : in std_logic;
27     RESET : in std_logic;
28     --
29     rsGetEnable    : in bit;
30     StartTimer   : in bit;
31     StopTimer    : in bit;
32     TimeOut      : out bit
33   );
34 end TimeoutCnt;

36 architecture behaviour of TimeoutCnt is
37 signal Xtimer : unsigned(16 downto 0);  -- 17 bit for 125000
38 signal doRun : bit;
39 signal TimeOutX : bit;

41 begin

43   timerCtrl : process(CLK,RESET)
44   begin
45     if RESET = '1' then
46       doRun <= '0' after 9 ns;
47     elsif CLK = '1' and CLK'event then
48       if TimeOutX = '1' then
49         doRun <= '0' after 9 ns;
50       elsif rsGetEnable = '1' then
51         if StartTimer = '1' then
52           doRun <= '1' after 9 ns;
53         elsif StopTimer = '1' then
54           doRun <= '0' after 9 ns;
55         end if; -- Start-/StopTimer
56       end if; -- rsGetEnable
57     end if; -- RESET and CLK
58   end process timerCtrl;

60   timer : process(RESET,CLK)
61   begin
62     if RESET = '1' then
63       Xtimer <= (others => '0') after 9 ns;
64       TimeOutX <= '0' after 9 ns;
65     elsif CLK = '1' and CLK'event then
66       if doRun = '1' then
67         if Xtimer = timerMax then
68           Xtimer <= (others => '0') after 9 ns;
69           TimeOutX <= '1' after 9 ns;
70         else
71           Xtimer <= Xtimer + 1 after 9 ns;
72           TimeOutX <= '0' after 9 ns;
73         end if;
74       else
```

```
75          Xtimer <= (others => '0') after 9 ns;
76          TimeOutX <= '0' after 9 ns;
77        end if; -- doRun
78      end if; -- RESET and CLK
79    end process timer;

81    TimeOut <= TimeOutX;

83  end behaviour;
```

C.1.19 Berechnung der Periodendauer und Verstärkung für das Vorverstärker-Modul

```
1   ------------------------------------------------------------
2   --  Period-calculation
3   --
4   --  Author: A. Arvidsson
5   ------------------------------------------------------------

7   -- <makepackage>
8   --   <global>
9   --     constant SampleIntervallValMax : unsigned := "11" & x"FFFF"; --## 18bit
10  --     constant SampleIntervallMax : integer := 18; --## 18bit
11  --    constant Offset  : unsigned(11 downto 0) := x"400"; --# 1024dec
12  --    constant GainMax : unsigned(11 downto 0) := x"303"; --# 771dec = x"303"
13  --    constant GainMin : unsigned(11 downto 0) := x"0F6"; --# 246dec = x"0F6"
14  --   </global>
15  --     <component = period_calc>
16  --     <name = arvidsson>
17  --     <project = ada_conv_modsys_master>
18  --      </component>
19  -- </makepackage>

21  library ieee;
22  use ieee.std_logic_1164.all;
23  use ieee.numeric_std.all;

25  library work;
26    use work.global_pkg.all;

28  entity period_calc is
29    port (
30      CLK    : in std_logic;
31      RESET : in std_logic;
32      --
33      sigCompUdiff1 : in bit; -- signal from comparator 1
34      sigCompUdiff2 : in bit; -- signal from comparator 2
35      SampleIntervall : out unsigned((SampleIntervallMax - 1) downto 0);
36      -- CalcGain
37      Udiff : in std_logic_vector(11 downto 0);
38      AMPgain : out std_logic_vector(2 downto 0);
39      AMP_CS : out bit
40    );
41  end period_calc;

43  architecture behaviour of period_calc is
44  -- PeriodCalc
45  type StateType is (Idle,CompHigh,HalfPeriodHigh,FullPeriod,CalcDevider,GetPeriod);
```

```
46  signal STATE : StateType;
47  signal TimeStart,TimeStop : bit;
48  signal Timer  : unsigned(23 downto 0);  -- 24bit
49  signal doStart : bit;
50  -- CalcGain
51  signal UdiffMax : std_logic_vector(11 downto 0);
52  signal UdiffMin : std_logic_vector(11 downto 0);
53  signal GainInt  : unsigned( 2 downto 0);
54  constant barrier1 : std_logic_vector(11 downto 0) := std_logic_vector(Offset + GainMax);
55  constant barrier2 : std_logic_vector(11 downto 0) := std_logic_vector(Offset + GainMin);
56  constant barrier3 : std_logic_vector(11 downto 0) := std_logic_vector(Offset - GainMin);
57  constant barrier4 : std_logic_vector(11 downto 0) := std_logic_vector(Offset - GainMax);
58  signal sigLow : bit;
59  signal sigHigh : bit;
60  signal clrMin : bit;
61  signal clrMax : bit;

63  begin
64    PeriodCalc : process(RESET,CLK)
65    begin
66      if RESET = '1' then
67        STATE <= Idle after 9 ns;
68        TimeStart <= '0' after 9 ns;
69        TimeStop <= '0' after 9 ns;
70        -- CalcGain
71        UdiffMax <= (others => '0') after 9 ns;
72        UdiffMin <= (others => '1') after 9 ns;
73        AMP_CS   <= '1' after 9 ns;
74        sigLow <= '0' after 9 ns;
75        sigHigh <= '0' after 9 ns;
76      elsif CLK = '1' and CLK'event then
77        TimeStart <= '0' after 9 ns;
78        TimeStop <= '0' after 9 ns;
79        STATE <= Idle after 9 ns;
80        -- CalcGain
81        AMP_CS <= '0' after 9 ns;
82        sigLow <= '0' after 9 ns;
83        sigHigh <= '0' after 9 ns;

85        if clrMin = '1' then
86          UdiffMin <= (others => '1') after 9 ns;
87        elsif clrMax = '1' then
88          UdiffMax <= (others => '0') after 9 ns;
89        end if;
90        if UdiffMax < Udiff then
91          UdiffMax <= Udiff after 9 ns;
92        elsif UdiffMin > Udiff then
93          UdiffMin <= Udiff after 9 ns;
94        end if;

96        case STATE is
97          when Idle =>
98            if sigCompUdiff1 = '0' and sigCompUdiff2 = '0' then
99              STATE <= CompHigh after 9 ns;
100           else
101             STATE <= Idle after 9 ns;
102           end if;
103         when CompHigh =>
104           STATE <= CompHigh after 9 ns;
105           if sigCompUdiff1 = '1' and sigCompUdiff2 = '1' then
106             TimeStart <= '1' after 9 ns;
107             STATE <= HalfPeriodHigh after 9 ns;
108           end if;
```

```
109         when HalfPeriodHigh =>
110           STATE <= HalfPeriodHigh after 9 ns;
111           if sigCompUdiff1 = '0' and sigCompUdiff2 = '0' then
112             sigLow <= '1' after 9 ns;
113             STATE <= FullPeriod after 9 ns;
114           end if;
115         when FullPeriod =>
116           STATE <= FullPeriod after 9 ns;
117           if sigCompUdiff1 = '1' and sigCompUdiff2 = '1' then
118             sigHigh <= '1' after 9 ns;
119             TimeStop <= '1' after 9 ns;
120             STATE <= CalcDevider after 9 ns;
121           end if;
122         when CalcDevider => -- Wartetakt, damit SampleIntervall berechnet werden kann
123           TimeStart <= '1' after 9 ns;
124           STATE <= GetPeriod after 9 ns;
125         when GetPeriod =>
126           STATE <= HalfPeriodHigh after 9 ns;
127       end case;
128     end if; -- RESET
129   end process PeriodCalc;

131   TimerCtr : process(RESET,CLK)
132   begin
133     if RESET = '1' then
134       SampleIntervall <= SampleIntervallValMax after 9 ns;
135       GainInt  <= (others => '1') after 9 ns;
136     elsif CLK = '1' and CLK'event then
137       doStart <= '0' after 9 ns;
138       -- CalcGain
139       clrMin <= '0' after 9 ns;
140       clrMax <= '0' after 9 ns;
141       if TimeStart = '1' then
142         doStart <= '1' after 9 ns;
143       elsif TimeStop = '1' then
144         SampleIntervall <= Timer(23 downto 6) after 9 ns; -- 12.5millionen / 64
145       end if;
146       if sigLow = '1' then
147         -- CalcGain
148         clrMin <= '1' after 9 ns;
149         -- <debug>  -- beim Debuggen auskommentieren
150         if (UdiffMin < barrier4) then
151           if GainInt > "000" then
152             GainInt <= GainInt - 1 after 9 ns;
153           end if;
154         elsif (UdiffMin > barrier3) then
155           if GainInt < "111" then
156             GainInt <= GainInt + 1 after 9 ns;
157           end if;
158         end if;
159         -- </debug>
160       elsif sigHigh = '1' then
161         -- CalcGain
162         clrMax <= '1' after 9 ns;
163         if (UdiffMax > barrier1) then
164           if GainInt > "000" then
165             GainInt <= GainInt - 1 after 9 ns;
166           end if;
167         elsif (UdiffMax < barrier2) then
168           if GainInt < "111" then
169             GainInt <= GainInt + 1 after 9 ns;
170           end if;
171         end if;
```

```
172       end if;
173     end if; -- RESET - CLK
174   end process TimerCtr;

176   AMPgain <= std_logic_vector(GainInt);

178   TimerAdding : process(RESET,CLK)
179   begin
180     if RESET = '1' then
181       Timer <= (others => '0') after 9 ns;
182     elsif CLK = '1' and CLK'event then
183       if doStart = '1' then
184         Timer(23 downto 2) <= (others => '0') after 9 ns;
185         Timer(1 downto 0) <= (others => '0') after 9 ns;
186       else
187         Timer <= Timer + 1 after 9 ns;
188       end if;
189     end if;
190   end process TimerAdding;

192 end behaviour;
```

C.1.20 UCF-Datei für das NEXYS2-Board

```
1  #CONN4
2  # #
3  NET RxD LOC = "L15"; # Bank = 1, Pin name = IO_L09N_1/A11, Type = DUAL, Sch name = JA1
4  NET TxD LOC = "K12"; # Bank = 1, Pin name = IO_L11N_1/A9/RHCLK1, Type = RHCLK/DUAL, Sch
       name = JA2
5  #NET "JA<2>" LOC = "L17"; # Bank = 1, Pin name = IO_L10N_1/VREF_1, Type = VREF, Sch name =
       JA3
6  #NET "JA<3>" LOC = "M15"; # Bank = 1, Pin name = IO_L07P_1, Type = I/O, Sch name = JA4
7  #
8  #NET "JA<4>" LOC = "M16"; # Bank = 1, Pin name = IO_L07N_1, Type = I/O, Sch name = JA10
9  #NET "JA<5>" LOC = "M14"; # Bank = 1, Pin name = IO_L05P_1, Type = I/O, Sch name = JA9
10 #NET "JA<6>" LOC = "L16"; # Bank = 1, Pin name = IO_L06P_1, Type = I/O, Sch name = JB7
11 #NET "JA<7>" LOC = "K13"; # Bank = 1, Pin name = IO_L03N_1/VREF_1, Type = VREF, Sch name =
       JB8
12 # #
13 NET statePart<0> LOC = "M13"; # Bank = 1, Pin name = IO_L05N_1/VREF_1, Type = VREF, Sch
       name = JB1
14 NET statePart<1> LOC = "R18"; # Bank = 1, Pin name = IO_L02P_1/A14, Type = DUAL, Sch name =
        JB2
15 NET statePart<2> LOC = "R15"; # Bank = 1, Pin name = IO_L03P_1, Type = I/O, Sch name = JB3
16 NET statePart<3> LOC = "T17"; # Bank = 1, Pin name = IO_L01N_1/A15, Type = DUAL, Sch name =
        JB4

18 NET statePart<4> LOC = "P17"; # Bank = 1, Pin name = IO_L06P_1, Type = I/O, Sch name = JB7
19 NET statePart<5> LOC = "R16"; # Bank = 1, Pin name = IO_L03N_1/VREF_1, Type = VREF, Sch
       name = JB8
20 #NET "JB<6>" LOC = "T18"; # Bank = 1, Pin name = IO_L02N_1/A13, Type = DUAL, Sch name = JB9
21 #NET "JB<7>" LOC = "U18"; # Bank = 1, Pin name = IO_L01P_1/A16, Type = DUAL, Sch name =
       JB10
22 # #
23 # NET "JC<0>" LOC = "G15"; # Bank = 1, Pin name = IO_L18P_1, Type = I/O, Sch name = JC1
24 # NET "JC<1>" LOC = "J16"; # Bank = 1, Pin name = IO_L13N_1/A5/RHCLK5, Type = RHCLK/DUAL,
       Sch name = JC2
25 # NET "JC<2>" LOC = "G13"; # Bank = 1, Pin name = IO_L20N_1, Type = I/O, Sch name = JC3
26 # NET "JC<3>" LOC = "H16"; # Bank = 1, Pin name = IO_L16P_1, Type = I/O, Sch name = JC4
```

```
27 # NET "JC<4>" LOC = "H15"; # Bank = 1, Pin name = IO_L17N_1, Type = I/O, Sch name = JC7
28 # NET "JC<5>" LOC = "F14"; # Bank = 1, Pin name = IO_L21N_1, Type = I/O, Sch name = JC8
29 # NET "JC<6>" LOC = "G16"; # Bank = 1, Pin name = IO_L18N_1, Type = I/O, Sch name = JC9
30 # NET "JC<7>" LOC = "J12"; # Bank = 1, Pin name = IO_L15P_1/A2, Type = DUAL, Sch name =
       JC10
31 # #
32 # NET "JD<0>" LOC = "J13"; # Bank = 1, Pin name = IO_L15N_1/A1, Type = DUAL, Sch name = JD1
33 # NET "JD<1>" LOC = "M18"; # Bank = 1, Pin name = IO_L08N_1, Type = I/O, Sch name = JD2
34 # NET "JD<2>" LOC = "N18"; # Bank = 1, Pin name = IO_L08P_1, Type = I/O, Sch name = JD3
35 # NET "JD<3>" LOC = "P18"; # Bank = 1, Pin name = IO_L06N_1, Type = I/O, Sch name = JD4
36 # NET "JD<4>"  LOC = "J14"; # Bank = 1, Pin name = IO_L14N_1/A3/RHCLK7, Type = RHCLK/DUAL,
       Sch name = JD10/LD0
37 # NET "JD<5>"  LOC = "J15"; # Bank = 1, Pin name = IO_L14P_1/A4/RHCLK6, Type = RHCLK/DUAL,
       Sch name = JD9/LD1
38 # NET "JD<6>"  LOC = "K15"; # Bank = 1, Pin name = IO_L12P_1/A8/RHCLK2, Type = RHCLK/DUAL,
       Sch name = JD8/LD2
39 # NET "JD<7>"  LOC = "K14"; # Bank = 1, Pin name = IO_L12N_1/A7/RHCLK3/TRDY1, Type = RHCLK/
       DUAL, Sch name = JD7/LD3

41 # ----------------------------------------------------------------------

43 # CONN3
44 # # Output
45 # NET "PIO<0>" LOC = "B4";
46 # NET "PIO<1>" LOC = "A4";
47 # NET "PIO<2>" LOC = "C3";
48 # NET "PIO<3>" LOC = "C4";
49 NET "AMP_CS" LOC = "B6";       # Pin 36
50 NET "AMPgain_O<0>" LOC = "D5";
51 NET "AMPgain_O<1>" LOC = "C5";
52 NET "AMPgain_O<2>" LOC = "F7";
53 # NET sigCompHB2 LOC = "E7";
54 # NET sigCompHB1 LOC = "A6";
55 NET sigCompUdiff2 LOC = "C7";
56 NET sigCompUdiff1 LOC = "F8";
57 # NET "PIO<12>" LOC = "D7";
58 # NET "PIO<13>" LOC = "E8";
59 NET DAC_WRn LOC = "E9";
60 NET DAC_RDn LOC = "C9";

62 NET DAC_CSn LOC = "A8";
63 NET DAC_GAIN LOC = "G9";
64 NET DAC_BUF LOC = "F9";
65 NET DAC_A<0> LOC = "D10";

67 #
68 # # Input
69 NET DAC_A<1> LOC = "A10";
70 NET DAC_A<2> LOC = "B10";
71 NET ADC_BUSY LOC = "A11";      # PIN 18
72 NET ADC_WRn LOC = "D11";
73 NET ADC_RDn LOC = "E10";
74 NET ADC_CSn LOC = "B11";
75 NET ADC_CLK LOC = "C11";
76 NET ADC_CONVSTn LOC = "E11";  # PIN 13
77 NET DB<0> LOC = "F11";
78 NET DB<1> LOC = "E12";
79 NET DB<2> LOC = "F12";
80 NET DB<3> LOC = "A13";
81 NET DB<4> LOC = "B13";
82 NET DB<5> LOC = "E13";
83 NET DB<6> LOC = "A14";
84 NET DB<7> LOC = "C14";
```

```
85  NET DB<8> LOC = "D14";
86  NET DB<9> LOC = "B14";
87  NET DB<10> LOC = "A16";      # PIN 2
88  NET DB<11> LOC = "B16";      # PIN 1

90  NET "SYS_CLK_S"   LOC = "B8"; # Bank = 0, Pin name = IP_L13P_0/GCLK8, Type = GCLK, Sch name
        = GCLK0

92  # buttons
93  NET "Btn0" LOC = "H13"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = BTN3
94  NET "Btn1" LOC = "E18"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = BTN2
95  NET "Btn2" LOC = "D18"; # Bank = 1, Pin name = IP/VREF_1, Type = VREF, Sch name = BTN1
96  NET "RESET" LOC = "B18" ; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = BTN0
97  NET "RESET" CLOCK_DEDICATED_ROUTE = FALSE; # LOC = "B18" ; # Bank = 1, Pin name = IP, Type
        = INPUT, Sch name = BTN0

99  # Switches
100 NET "sw<0>" LOC = "G18"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW0
101 NET "sw<1>" LOC = "H18"; # Bank = 1, Pin name = IP/VREF_1, Type = VREF, Sch name = SW1
102 NET "sw<2>" LOC = "K18"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW2
103 NET "sw<3>" LOC = "K17"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW3
104 # NET "sw<4>" LOC = "L14"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW4
105 # NET "sw<5>" LOC = "L13"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW5
106 # NET "sw<6>" LOC = "N17"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW6
107 # NET "sw<7>" LOC = "R17"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW7
108 #

110 # RS232 connector
111 NET "RxD_PAD_I" LOC = "U6"; # Bank = 2, Pin name = IP, Type = INPUT, Sch name = RS-RX
112 # NET "RxD_PAD_I" CLOCK_DEDICATED_ROUTE = FALSE;
113 NET "TxD_PAD_O" LOC = "P9"; # Bank = 2, Pin name = IO, Type = I/O, Sch name = RS-TX

115 # 7 segment display
116 NET "seg<0>" LOC = "L18"; # Bank = 1, Pin name = IO_L10P_1, Type = I/O, Sch name = CA
117 NET "seg<1>" LOC = "F18"; # Bank = 1, Pin name = IO_L19P_1, Type = I/O, Sch name = CB
118 NET "seg<2>" LOC = "D17"; # Bank = 1, Pin name = IO_L23P_1/HDC, Type = DUAL, Sch name = CC
119 NET "seg<3>" LOC = "D16"; # Bank = 1, Pin name = IO_L23N_1/LDC0, Type = DUAL, Sch name = CD
120 NET "seg<4>" LOC = "G14"; # Bank = 1, Pin name = IO_L20P_1, Type = I/O, Sch name = CE
121 NET "seg<5>" LOC = "J17"; # Bank = 1, Pin name = IO_L13P_1/A6/RHCLK4/IRDY1, Type = RHCLK/
        DUAL, Sch name = CF
122 NET "seg<6>" LOC = "H14"; # Bank = 1, Pin name = IO_L17P_1, Type = I/O, Sch name = CG
123 NET "dp"      LOC = "C17"; # Bank = 1, Pin name = IO_L24N_1/LDC2, Type = DUAL, Sch name = DP

125 NET "an<0>" LOC = "F17"; # Bank = 1, Pin name = IO_L19N_1, Type = I/O, Sch name = AN0
126 NET "an<1>" LOC = "H17"; # Bank = 1, Pin name = IO_L16N_1/A0, Type = DUAL, Sch name = AN1
127 NET "an<2>" LOC = "C18"; # Bank = 1, Pin name = IO_L24P_1/LDC1, Type = DUAL, Sch name = AN2
128 NET "an<3>" LOC = "F15"; # Bank = 1, Pin name = IO_L21P_1, Type = I/O, Sch name = AN3
```

C.1.21 ADC-Testbench

```
1  ----------------------------------------------------------------
2  -- Pseudocode for adc (ads7865)
3  --
4  -- Author: A. Arvidsson
5  ----------------------------------------------------------------

7  -- pseudocode of ads7865
8  -- <makepackage>
9  --   <component = TBadc_ads7865>
```

```
10 --      <name = arvidsson>
11 --      <project = testbench>
12 --    </component>
13 -- </makepackage>

15 library ieee;
16 use ieee.std_logic_1164.all;
17 use ieee.numeric_std.all;

19 entity TBadc_ads7865 is
20   port (
21     RESET    : in std_logic;
22     CLK      : in std_logic;
23     -- digital ports
24     DB       : inout std_logic_vector(11 downto 0);
25     nCS      : in bit;
26     nWR      : in bit;
27     nRD      : in bit;
28     BUSY     : out bit;
29     nCONVST  : in std_logic;
30     -- analog ports
31     CHA0p    : in std_logic_vector(11 downto 0);
32     CHA0m    : in std_logic_vector(11 downto 0);
33     CHA1p    : in std_logic_vector(11 downto 0);
34     CHA1m    : in std_logic_vector(11 downto 0);
35     CHB0p    : in std_logic_vector(11 downto 0);
36     CHB0m    : in std_logic_vector(11 downto 0);
37     CHB1p    : in std_logic_vector(11 downto 0);
38     CHB1m    : in std_logic_vector(11 downto 0);
39     refIn    : in  std_logic_vector(9 downto 0);
40     refOut   : out std_logic_vector(9 downto 0)
41   );
42 end TBadc_ads7865;

44 architecture behaviour of TBadc_ads7865 is
45 signal outgoingData : std_logic_vector(11 downto 0);
46 signal setBusy   : bit;
47 signal regConfig : std_logic_vector(11 downto 0);
48 signal regDAC : std_logic_vector(11 downto 0);
49 signal regSeq : unsigned(1 downto 0);
50 signal WriteNow : bit;
51 signal ReadNow : bit;
52 signal writeSeq : bit;
53 signal count  : unsigned(3 downto 0);
54 signal sequenceCounter : unsigned(1 downto 0);  -- := "00";
55 signal ReadFrom : std_logic_vector(11 downto 0);
56 signal WriteTo  : std_logic_vector(11 downto 0);
57 signal progDAC : bit;
58 signal progSeq : bit;
59 signal readDAC : bit;
60 signal readSeq : bit;
61 signal runningBusy : bit;
62 signal counter : bit;

64 -- analog signals
65 signal cha0p_sig : std_logic_vector(11 downto 0);
66 signal cha0m_sig : std_logic_vector(11 downto 0);
67 signal cha1p_sig : std_logic_vector(11 downto 0);
68 signal cha1m_sig : std_logic_vector(11 downto 0);
69 signal chb0p_sig : std_logic_vector(11 downto 0);
70 signal chb0m_sig : std_logic_vector(11 downto 0);
71 signal chb1p_sig : std_logic_vector(11 downto 0);
72 signal chb1m_sig : std_logic_vector(11 downto 0);
```

```
74  signal ChannelA0 : std_logic_vector(11 downto 0);
75  signal ChannelA1 : std_logic_vector(11 downto 0);
76  signal ChannelB0 : std_logic_vector(11 downto 0);
77  signal ChannelB1 : std_logic_vector(11 downto 0);

79  begin

81    sampleAndHold : process(RESET,nCONVST)
82    begin
83      if RESET = '1' then
84        cha0p_sig <= (others => '0') after 9 ns;
85        cha0m_sig <= (others => '0') after 9 ns;
86        chb0p_sig <= (others => '0') after 9 ns;
87        chb0m_sig <= (others => '0') after 9 ns;
88        cha1p_sig <= (others => '0') after 9 ns;
89        cha1m_sig <= (others => '0') after 9 ns;
90        chb1p_sig <= (others => '0') after 9 ns;
91        chb1m_sig <= (others => '0') after 9 ns;
92      elsif nCONVST = '0' and nCONVST'event then
93        -- cha0p_sig <= signed(CHA0p) after 9 ns;  -- CHA0
94        cha0p_sig <= CHA0p after 9 ns;  -- CHA0
95        cha0m_sig <= CHA0m after 9 ns;
96        chb0p_sig <= ChannelB0 after 9 ns;  -- CHB0
97        chb0m_sig <= CHB0m after 9 ns;
98        cha1p_sig <= ChannelA1 after 9 ns;  -- CHA1
99        cha1m_sig <= CHA1m after 9 ns;
100       chb1p_sig <= ChannelB1 after 9 ns;  -- CHB1
101       chb1m_sig <= CHB1m after 9 ns;
102     end if;
103   end process sampleAndHold;

105   InputData : process(RESET,CLK)
106   begin
107     if RESET = '1' then
108       -- ChannelA0 <= (others => '0');
109       ChannelA1 <= x"400";
110       ChannelB0 <= x"800";
111       ChannelB1 <= x"C00";
112     elsif CLK = '1' and CLK'event then
113       if nCONVST = '0' then
114         -- ChannelA0 <= std_logic_vector(signed(ChannelA0) + 1);
115         ChannelA1 <= std_logic_vector(unsigned(ChannelA1) + 1);
116         ChannelB0 <= std_logic_vector(unsigned(ChannelB0) + 1);
117         ChannelB1 <= std_logic_vector(unsigned(ChannelB1) + 1);
118       end if;
119     end if;
120   end process InputData;

122   control : process(RESET,CLK)
123   begin
124     if RESET = '1' then
125       setBusy <= '0' after 9 ns;
126       WriteTo <= (others => '0') after 9 ns;
127       WriteNow <= '0' after 9 ns;
128       regConfig <= (others => '0') after 9 ns;
129       regDAC <= (others => '0') after 9 ns;
130       regSeq <= "00" after 9 ns;
131       writeSeq <= '0' after 9 ns;
132     elsif CLK = '1' and CLK'event then
133       setBusy <= '0' after 9 ns;
134       WriteNow <= '0' after 9 ns;
135       ReadNow <= '0' after 9 ns;
```

```
136       writeSeq <= '0' after 9 ns;
137       if nCONVST = '0' then
138         setBusy <= '1' after 9 ns;  -- 13 clks high and 3 clks low
139       else
140         if nRD = '0' then -- reading from outside means writing here
141           WriteNow <= '1' after 9 ns;
142           if WriteNow <= '0' then
143             if readDAC = '1' then
144               readDAC <= '0' after 9 ns;
145               WriteTo <= regDAC after 9 ns;
146             elsif readSeq = '1' then
147               readSeq <= '0' after 9 ns;
148               if regSeq = "11" then
149                 regSeq <= "00" after 9 ns;
150               else
151                 regSeq <= regSeq + 1 after 9 ns;
152               end if;
153               WriteTo <= x"00" & "00" & std_logic_vector(regSeq) after 9 ns;
154             else
155                 writeSeq <= '1' after 9 ns;
156                 WriteTo <= outgoingData after 9 ns;
157             end if;
158           end if;
159         elsif nWR = '0' then  -- writing into ADC-register means reading here
160           ReadNow <= '1' after 9 ns;
161           if progDAC = '1' then
162             progDAC <= '0' after 9 ns;
163             regDAC <= ReadFrom after 9 ns;
164           elsif progSeq = '1' then
165             progSeq <= '0' after 9 ns;
166           else
167             case ReadFrom is
168               when x"101" => progDAC <= '1' after 9 ns;
169               when x"104" => progSeq <= '1' after 9 ns;
170               when x"103" => readDAC <= '1' after 9 ns;
171               when x"106" => readSeq <= '1' after 9 ns;
172               when others => regConfig <= ReadFrom after 9 ns;
173             end case;
174           end if;
175         end if; -- nRD,nWR
176       end if; -- nCONVST
177     end if; -- RESET
178   end process control;

180   OUTPUT : process (WriteNow, DB, WriteTo)     -- Behavioral representation
181   begin                                        -- of tri-states.
182     if WriteNow = '0' then
183       DB <= (others => 'Z') after 9 ns;
184       ReadFrom <= DB after 9 ns;  -- only invert MSB for real data therefore this has to be
              done REG_DATA-process!!
185     elsif WriteNow = '1' then
186       DB <= WriteTo after 9 ns;
187       ReadFrom <= DB after 9 ns;
188     end if;
189   end process OUTPUT;

191   sequenceCount : process(RESET,CLK)
192   begin
193     if RESET = '1' then
194       sequenceCounter <= (others => '0') after 9 ns;
195       counter <= '0' after 9 ns;
196     elsif CLK = '1' and CLK'event then
197       if writeSeq = '1' then
```

```
198         if counter = '0' then
199           counter <= '1' after 9 ns;
200           if sequenceCounter = "11" then
201             sequenceCounter <= "00" after 9 ns;
202           else
203             sequenceCounter <= sequenceCounter + 1 after 9 ns;
204           end if;
205         end if;
206       else
207         counter <= '0' after 9 ns;
208       end if;
209     end if;
210   end process sequenceCount;

212   DataOut : process(RESET,runningBusy,sequenceCounter,cha0p_sig,cha0m_sig,chb0p_sig,
          chb0m_sig,cha1p_sig,cha1m_sig,chb1p_sig,chb1m_sig)
213   begin
214     if RESET = '1' then
215       outgoingData <= (others => '0') after 9 ns;
216     else
217       if runningBusy = '1' then
218         case sequenceCounter is
219           when "00" => outgoingData <= cha0p_sig after 9 ns;  -- channelA0
220           when "01" => outgoingData <= chb0p_sig after 9 ns;  -- channelB0
221           when "10" => outgoingData <= cha1p_sig after 9 ns;  -- channelA1
222           when "11" => outgoingData <= chb1p_sig after 9 ns;  -- channelB1
223           when others => null;
224         end case;
225       end if; -- runningBusy
226     end if; -- RESET and CLK
227   end process DataOut;

229   doBusy : process(RESET,CLK) -- make BUSY-signal
230   begin
231     if RESET = '1' then
232       BUSY <= '0' after 9 ns;
233       runningBusy <= '0' after 9 ns;
234     elsif CLK = '1' and CLK'event then
235       if setBusy = '1' then
236         runningBusy <= '1' after 9 ns;
237         BUSY <= '1' after 9 ns;
238       elsif runningBusy = '1' then
239         if count < 13 then
240           BUSY <= '1' after 9 ns;
241         else
242           BUSY <= '0' after 9 ns;
243           runningBusy <= '0' after 9 ns;
244         end if;
245       end if;
246     end if;
247   end process doBusy;

249   CounterProc : process(RESET,CLK)
250   begin
251     if RESET = '1' then
252       count <= (others => '0') after 9 ns;
253     elsif CLK = '1' and CLK'event then
254       if setBusy = '1' then
255         count <= x"1" after 9 ns;
256       elsif count < 13 then
257         count <= count + 1 after 9 ns;
258       else
259         count <= (others => '0') after 9 ns;
```

```
260       end if;
261     end if;
262   end process CounterProc;

264   referenceOut : process(regDAC)
265   begin
266     refOut <= regDAC(9 downto 0) after 9 ns;
267   end process referenceOut;

269 end behaviour;
```

C.1.22 DAC-Testbench

```
1  ----------------------------------------------------------------
2  -- Pseudocode for dac (ad5348)
3  --
4  -- Author: A. Arvidsson
5  ----------------------------------------------------------------

7  -- pseudocode of ad5348
8  -- <makepackage>
9  --   <component = TBdac_ad5348>
10 --     <name = arvidsson>
11 --     <project = testbench>
12 --   </component>
13 -- </makepackage>

15 library ieee;
16 use ieee.std_logic_1164.all;
17 use ieee.numeric_std.all;

19 entity TBdac_ad5348 is
20   port (
21     RESET    : in std_logic;
22     CLK      : in std_logic;
23     -- digital ports
24     DB       : inout std_logic_vector(11 downto 0);
25     nCSd     : in bit;
26     nWRd     : in bit;
27     nLDACd   : in bit;
28     nRDd     : in bit;
29     BUFd     : in bit;
30     GAINd    : in bit;
31     Ad       : in bit_vector(2 downto 0);
32     -- analog ports
33     -- VrefAB  : in -- open
34     -- VrefCD  : in
35     -- VrefEF  : in
36     -- VrefGH  : in
37     VoutA    : out std_logic_vector(11 downto 0);
38     VoutB    : out std_logic_vector(11 downto 0);
39     VoutC    : out std_logic_vector(11 downto 0);
40     VoutD    : out std_logic_vector(11 downto 0);
41     VoutE    : out std_logic_vector(11 downto 0);
42     VoutF    : out std_logic_vector(11 downto 0);
43     VoutG    : out std_logic_vector(11 downto 0);
44     VoutH    : out std_logic_vector(11 downto 0)
45   );
46 end TBdac_ad5348;
```

```
48  architecture behaviour of TBdac_ad5348 is
49  signal WriteNow : bit;
50  signal ReadFrom : std_logic_vector(11 downto 0);
51  signal WriteTo  : std_logic_vector(11 downto 0);
52  signal inVoutA  : std_logic_vector(11 downto 0);
53  signal inVoutB  : std_logic_vector(11 downto 0);
54  signal inVoutC  : std_logic_vector(11 downto 0);
55  signal inVoutD  : std_logic_vector(11 downto 0);
56  signal inVoutE  : std_logic_vector(11 downto 0);
57  signal inVoutF  : std_logic_vector(11 downto 0);
58  signal inVoutG  : std_logic_vector(11 downto 0);
59  signal inVoutH  : std_logic_vector(11 downto 0);

61  begin

63    Ctrl : process(RESET,CLK)
64    begin
65      if RESET = '1' then
66        inVoutA <= (others => '0') after 9 ns;
67        inVoutB <= (others => '0') after 9 ns;
68        inVoutC <= (others => '0') after 9 ns;
69        inVoutD <= (others => '0') after 9 ns;
70        inVoutE <= (others => '0') after 9 ns;
71        inVoutF <= (others => '0') after 9 ns;
72        inVoutG <= (others => '0') after 9 ns;
73        inVoutH <= (others => '0') after 9 ns;
74        WriteNow <= '0' after 9 ns;
75      elsif CLK = '1' and CLK'event then
76        WriteNow <= '0' after 9 ns;
77        if nCSd = '0' then
78          if nWRd = '0' then
79            case Ad is
80              when "000" => inVoutA <= ReadFrom after 9 ns;
81              when "001" => inVoutB <= ReadFrom after 9 ns;
82              when "010" => inVoutC <= ReadFrom after 9 ns;
83              when "011" => inVoutD <= ReadFrom after 9 ns;
84              when "100" => inVoutE <= ReadFrom after 9 ns;
85              when "101" => inVoutF <= ReadFrom after 9 ns;
86              when "110" => inVoutG <= ReadFrom after 9 ns;
87              when "111" => inVoutH <= ReadFrom after 9 ns;
88            end case;
89          elsif nRDd = '0' then
90            WriteNow <= '1' after 9 ns;
91            case Ad is
92              when "000" => WriteTo <= inVoutA after 9 ns;
93              when "001" => WriteTo <= inVoutB after 9 ns;
94              when "010" => WriteTo <= inVoutC after 9 ns;
95              when "011" => WriteTo <= inVoutD after 9 ns;
96              when "100" => WriteTo <= inVoutE after 9 ns;
97              when "101" => WriteTo <= inVoutF after 9 ns;
98              when "110" => WriteTo <= inVoutG after 9 ns;
99              when "111" => WriteTo <= inVoutH after 9 ns;
100           end case;
101         end if;
102       end if;
103     end if;
104   end process Ctrl;

106   VoutA <= inVoutA;
107   VoutB <= inVoutB;
108   VoutC <= inVoutC;
109   VoutD <= inVoutD;
```

```
110   VoutE <= inVoutE;
111   VoutF <= inVoutF;
112   VoutG <= inVoutG;
113   VoutH <= inVoutH;

115   OUTPUT : process (WriteNow, DB, WriteTo)     -- Behavioral representation
116   begin                                        -- of tri-states.
117     if WriteNow = '0' then
118       DB <= (others => 'Z') after 9 ns;
119       ReadFrom <= DB after 9 ns;
120     elsif WriteNow = '1' then
121       DB <= WriteTo after 9 ns;
122       ReadFrom <= DB after 9 ns;
123     end if;
124   end process OUTPUT;

126 end behaviour;
```

C.1.23 Signalerzeugung für Simulation

```
1  --#####################################################
2  --##   Project name : adc_sample_calc_thd
3  --##   File name    : TL_adc_sample_calc_thd.vhd
4  --##   Author       : Jan-Heiner Dreschhoff

6  --<makepackage>
7  --  <component = tb_source>
8  --    <name = dreschhoff>
9  --    <project = thd>
10 --  </component>
11 --</makepackage>

13 library ieee;
14   use ieee.std_logic_1164.all;
15   use ieee.numeric_std.all;

17 entity tb_source is
18   generic(
19     STEPS_PER_PERIOD   : natural := 128;
20     STEPS_PHASE_OFFSET : natural := 0;
21     TIME_PER_STEP      : time    := 5100 ns;
22     HARMONIC_NUMBER    : natural := 1;
23     VALUE_WIDTH        : natural := 12;
24     MIN_VALUE          : integer := 0;
25     MAX_VALUE          : integer := 4095
26     );
27   port(
28     VALUE: out std_logic_vector((VALUE_WIDTH-1) downto 0)
29     );
30 end tb_source;

32 architecture behavior of tb_source is

34   constant offset    : real := real(MAX_VALUE+MIN_VALUE)/real(2);
35   constant amplitude : real := real(MAX_VALUE-MIN_VALUE)/real(2);
36   constant omega     : real := 2*3.14159265/real(STEPS_PER_PERIOD);
37   constant phase     : real := real(STEPS_PHASE_OFFSET);

39 begin
```

```
41    process
42      variable step : natural := 0;
43      begin

45        if step = STEPS_PER_PERIOD then
46          step := 0;
47        end if;
48  --((MAX_VALUE-MIN_VALUE)/2)+(((MAX_VALUE-MIN_VALUE)/2)*sin(2pi*step/STEPS_PER_PERIOD));
49        VALUE <= std_logic_vector(to_unsigned(integer((offset + amplitude
50              * ieee.math_real.sin((omega*real(step))+phase))), VALUE'length));

52        step := step + 1;
53        wait for TIME_PER_STEP/HARMONIC_NUMBER;
54      end process;

56  end behavior;
```

C.1.24 Toplevel-Testbench

```
1   ------------------------------------------------------------
2   --  Toplevel for testing ada_conv_modsys-board
3   --
4   --  Author: A.Arvidsson
5   ------------------------------------------------------------

7   --<makepackage>
8   --  <component = TB_Toplevel_TL>
9   --    <name = dreschhoff>
10  --    <project = ada_conv_modsys>
11  --  </component>
12  --</makepackage>

14  library unisim;
15    use unisim.vcomponents.all;

17  library ieee;
18    use ieee.std_logic_1164.all;
19    use ieee.numeric_std.all;

21  library work;
22    use work.global_pkg.all;
23    use work.ada_conv_modsys_TL_pkg.all;
24    use work.MiniUART_TL_pkg.all;
25    use work.CLK_DIV_pkg.all;
26    use work.TBadc_ads7865_pkg.all;
27    use work.TBdac_ad5348_pkg.all;
28    use work.tb_source_pkg.all;
29    use work.SinGain_pkg.all;

31  entity TB_Toplevel_TL is
32    generic (
33      INTCLK  : time := 10 ns;
34      maxTime : time := 40000 ns
35    );
36  end TB_Toplevel_TL;


39  architecture behaviour of TB_Toplevel_TL is
```

```
40 ------ testbench signals --------
41 signal SYS_CLK_S      : std_logic; -- Systemtakt
42 signal RESET          : std_logic; -- allgemeiner Reset
43 signal Btn0, Btn1, Btn2 : std_logic;

45 signal sw  : bit_vector(2 downto 0);

47 signal sigCompUdiff1 : bit;
48 signal sigCompUdiff2 : bit;

50     --- ADC ---
51 signal ADC_BUSY                : bit;
52 signal DB                      : std_logic_vector(11 downto 0);
53 signal ADC_CSn, ADC_WRn, ADC_RDn   : bit;
54 signal ADC_CONVSTn              : std_logic;
55 signal NEW_DATA_O               : bit_vector(1 downto 0);
56 signal DB_UDIFF_O               : std_logic_vector(11 downto 0);
57 signal DB_HB1_O, DB_HB2_O : std_logic_vector(11 downto 0);
58 signal CLK_O                   : std_logic;
59 signal ADC_CLK                 : std_logic;
60     --------
61     --- DAC ---
62 signal DAC_CSn        : bit;
63 signal DAC_WRn        : bit;
64 signal DAC_LDACn      : bit;
65 signal DAC_RDn        : bit;
66 signal DAC_BUF        : bit;
67 signal DAC_GAIN           : bit;
68 signal DAC_A          : bit_vector(2 downto 0);
69     ----------
70     ---- MiniUART ----
71 signal TxD_PAD_O : std_logic;  -- Tx RS232 Line
72 signal RxD_PAD_I : std_logic;  -- Rx RS232 Line
73 signal TxD        : std_logic;
74 signal RxD        : std_logic;
75     ----------------
76     ---- 7-Segment ----
77 signal seg : bit_vector(6 downto 0);
78 signal an  : bit_vector(3 downto 0);
79 signal dp  : bit;
80     --------------------
81 ------------------------------------
82 signal CLK        : std_logic;
83 signal BtnNextArray : bit;
84 signal BtnBackValue : bit;
85 signal BtnNextValue : bit;
86 signal ArrayValue : std_logic_vector(11 downto 0);

88 -- MiniUART
89 signal NEW_DATA   : bit_vector(1 downto 0);
90 signal DB_UDIFF   : std_logic_vector(11 downto 0);
91 signal DB_HB1     : std_logic_vector(11 downto 0);
92 signal DB_HB2     : std_logic_vector(11 downto 0);
93 signal ProgReg    : std_logic_vector(3 downto 0);
94 signal ProgInfo   : std_logic_vector(11 downto 0);
95 signal newProgReg : bit;
96 signal ProgRegReset : bit;

98 signal DataFull : bit;
99 signal saveInArray : bit;
100 signal ADCsequence : std_logic_vector(11 downto 0);
101 signal ADCsequence_O : std_logic_vector(11 downto 0);
```

```
103 signal ref_diff1 : std_logic_vector(11 downto 0);
104 signal ref_diff2 : std_logic_vector(11 downto 0);
105 signal ref_hb1 : std_logic_vector(11 downto 0);
106 signal ref_hb2 : std_logic_vector(11 downto 0);
107 signal u_broff1 : std_logic_vector(11 downto 0);
108 signal u_broff2 :std_logic_vector(11 downto 0);
109 signal u_off : std_logic_vector(11 downto 0);
110 signal u_opt : std_logic_vector(11 downto 0);

112 signal ADCrefOut : std_logic_vector(11 downto 0);
113 signal AMPgain_O : std_logic_vector(2 downto 0);
114 signal AMPgain : std_logic_vector(2 downto 0);
115 signal AMP_CS : bit;
116 signal DataReady : bit;
117 signal SendingDone : bit;

119 signal TxD_PAD : std_logic;
120 signal readout : std_logic;
121 signal statePart : bit_vector(5 downto 0);
122 type SendArray is array(110 downto 0) of std_logic;

124 -- 2 Stop Bits
125 -- A0FF F0FF:
126   -- signal RxD_PAD_I_sendArray : SendArray :=
        "01100010011010001100110010011001100000010111011111111111011000100110100011001100100110011000001111
         -- with 2 stop bits
127                                          -- -- |           |           |           |
                                                    |           ||          |           |
                                                     |          |           |
128 -- A0FF D00A:
129   -- signal RxD_PAD_I_sendArray : SendArray :=
        "01100010011010001100110010011001100000010111011111111111011000100110100011001100100110011000001011
         -- with 2 stop bits
130                                          -- |           |           |           |           |
                                                    ||          |           |
                                                 |'D0'       |'0A'       |
131 -- A0FF C000:
132   -- signal RxD_PAD_I_sendArray : SendArray :=
        "01100010011010001100110010011001100000010111011111111111011000100110100011001100100110011000000011
         -- with 2 stop bits
133                                          -- |           |           |           |           |
                                                    ||          |           |
                                                 |'C0'       |'00'       |
134 -- A0FF D001:
135   -- signal RxD_PAD_I_sendArray : SendArray :=
        "01100010011010001100110010011001100000010111011111111111011000100110100011001100100110011000001011
         -- with 2 stop bits
136                                          -- |           |           |           |           |
                                                    ||          |           |
                                                 |'D0'       |'0A'       |
137 -- D00A C000:
138   signal RxD_PAD_I_sendArray : SendArray := "
        01100010011010001100110010011001100000010111100101000011101100010011010001100110010011001100000001111
        "; -- with 2 stop bits
139                                       -- |           |           |           |'D0'       |'0A
                                              '       ||          |           |           |'C0'
                                                    |'00'       |

141 signal index : integer range 0 to 110 := 0; -- 10byte + Start- & 2 Stopbit  => 10*8+10*2(
      Stopbit) + 10*1 (Startbit) = 110
142 signal ready : bit;
143 signal StartSending : bit;
144 signal doWait : bit;
```

```
146 -- adc-part --
147 signal nCS      : bit;
148 signal nWR      : bit;
149 signal nRD      : bit;
150 signal nCONVST : std_logic;
151     -- analog ports
152 signal CHA0p   : std_logic_vector(11 downto 0);
153 signal CHA0m   : std_logic_vector(11 downto 0);
154 signal CHA1p   : std_logic_vector(11 downto 0);
155 signal CHA1m   : std_logic_vector(11 downto 0);
156 signal CHB0p   : std_logic_vector(11 downto 0);
157 signal CHB0m   : std_logic_vector(11 downto 0);
158 signal CHB1p   : std_logic_vector(11 downto 0);
159 signal CHB1m   : std_logic_vector(11 downto 0);
160 signal refIn   : std_logic_vector(9 downto 0);
161 signal refOut  : std_logic_vector(9 downto 0);
162 signal refOut_i : std_logic_vector(9 downto 0);

164 signal sigVal : std_logic_vector(11 downto 0);

166 signal VoutA  : std_logic_vector(11 downto 0);
167 signal VoutB  : std_logic_vector(11 downto 0);
168 signal VoutC  : std_logic_vector(11 downto 0);
169 signal VoutD  : std_logic_vector(11 downto 0);
170 signal VoutE  : std_logic_vector(11 downto 0);
171 signal VoutF  : std_logic_vector(11 downto 0);
172 signal VoutG  : std_logic_vector(11 downto 0);
173 signal VoutH  : std_logic_vector(11 downto 0);

175 begin

177   CLOCK : process
178   begin
179     SYS_CLK_S <= '1';
180       wait for INTCLK;
181     SYS_CLK_S <= '0';
182       wait for INTCLK;
183   end process CLOCK;

185   I_DIV : CLK_DIV
186     port map (
187       SYS_CLK_S => SYS_CLK_S,
188       RESET     => RESET,
189       CLK_OUT   => CLK_O
190     );

192   BUFG_CLK_OUT : BUFG
193     port map (O => CLK,
194               I => CLK_O);

196   OBUF_ADC_CLK : OBUF
197     port map (O => ADC_CLK,
198               I => CLK);

200   init : process
201   begin
202     sw <= (others => '0');
203     RESET <= '1';
204       wait for 300 ns;
205     RESET <= '0';
206       wait;
207   end process init;
```

```
209   starting : process
210   begin
211     StartSending <= '0';
212       wait for 887240 ns;
213     StartSending <= '1';
214       wait for 10*573760 ns;
215       wait for 4000000 ns;
216     StartSending <= '0';
217       wait;
218   end process starting;

220   indexing : process
221   begin
222     if StartSending = '1' then
223       doWait <= '0';
224       ready <= '0';
225       if index >= 0 and index <= 110 then
226         if index = 110 then
227           index <= 0;
228           ready <= '1';
229         elsif index = 54 then
230           doWait <= '1';
231           index <= index + 1;
232             wait for 3500 us;
233         else
234           index <= index + 1;
235         end if;

237       end if;
238     else
239       index <= 0;
240     end if;
241         wait for 1*52160 ns;
242   end process indexing;

244   rs232 : process
245   begin
246     if StartSending = '1' then
247       if ready = '0' then
248         if doWait = '1' then
249           RxD_PAD_I <= '1';
250         else
251           RxD_PAD_I <= RxD_PAD_I_sendArray(110 - index);
252         end if;
253       elsif ready = '1' then
254         if doWait = '1' then
255           RxD_PAD_I <= '1';
256         else
257           RxD_PAD_I <= RxD_PAD_I_sendArray(110 - index);
258         end if;
259       else
260         RxD_PAD_I <= '1';
261       end if;
262     else
263       RxD_PAD_I <= '1';
264     end if;
265       wait for 1*52160 ns;
266   end process rs232;

268   -- adc-part
269   AnalogPorts : process
270   begin
```

```
271     if RESET = '1' then
272       -- CHA0p <= (others => '0');  -- Udiff
273       CHA0m <= (others => '0');
274       CHA1p <= (others => '0');  -- null  (neuerdings ebenfalls Udiff)
275       CHA1m <= (others => '0');
276       CHB0p <= (others => '0');  -- HB1
277       CHB0m <= (others => '0');
278       CHB1p <= (others => '0');  -- HB2
279       CHB1m <= (others => '0');
280     else
281       -- CHA0p <= std_logic_vector(ChannelA0);  -- Udiff
282       -- CHA0p <= x"111";  -- Udiff
283       CHA0m <= (others => '0');
284       -- CHA1p <= std_logic_vector(ChannelA1);  -- null
285       CHA1p <= x"555";  -- null
286       CHA1m <= (others => '0');
287       -- CHB0p <= std_logic_vector(ChannelB0);  -- HB1
288       CHB0p <= x"AAA";  -- HB1
289       CHB0m <= (others => '0');
290       -- CHB1p <= std_logic_vector(ChannelB1);  -- HB2
291       CHB1p <= x"FFF";  -- HB2
292       CHB1m <= (others => '0');
293     end if;
294       wait;
295   end process AnalogPorts;

297   init_signal : process
298   begin
299     if CHA0p > x"389" then  -- x"439"
300       sigCompUdiff1 <= '1';
301     else
302       sigCompUdiff1 <= '0';
303     end if;
304     if CHA0p > x"409" then  -- x"459"
305       sigCompUdiff2 <= '1';
306     else
307       sigCompUdiff2 <= '0';
308     end if;
309       wait for 40 ns;
310   end process init_signal;

312   ------------------------------------------------------------
313   RxD <= RxD_PAD_I;
314   TxD <= TxD_PAD;
315   TxD_PAD_O <= TxD_PAD;
316   DB_UDIFF_O <= DB_UDIFF;
317   DB_HB1_O   <= DB_HB1;
318   DB_HB2_O   <= DB_HB2;
319   NEW_DATA_O <= NEW_DATA;
320   AMPgain_O <= AMPgain;

322   -- adc-part
323   refIn <= refOut_i;
324   refOut <= refOut_i;

326   I_ada : ada_conv_modsys_TL
327     port map (
328       CLK => CLK,
329       RESET      => RESET,
330       ArrayValue       => ArrayValue,
331       sw  => sw,

333       ---- ADC ----
```

```
334         ADC_BUSY     => ADC_BUSY,
335         DB           => DB,
336         ADC_CSn      => ADC_CSn,
337         ADC_WRn      => ADC_WRn,
338         ADC_RDn      => ADC_RDn,
339         ADC_CONVSTn  => ADC_CONVSTn,
340         NEW_DATA_O   => NEW_DATA,
341         DB_UDIFF_O   => DB_UDIFF,
342         DB_HB1_O     => DB_HB1,
343         DB_HB2_O     => DB_HB2,
344         ---------
345         ---- DAC ----
346         DAC_CSn          => DAC_CSn,
347         DAC_WRn          => DAC_WRn,
348         DAC_LDACn        => DAC_LDACn,
349         DAC_RDn          => DAC_RDn,
350         DAC_BUF          => DAC_BUF,
351         DAC_GAIN           => DAC_GAIN,
352         DAC_A            => DAC_A,
353         ---------
354         ---- Outputs ----
355           -- DAC --
356         ref_diff1_O   => ref_diff1,
357         ref_diff2_O   => ref_diff2,
358         ref_hb1       => ref_hb1,
359         ref_hb2       => ref_hb2,
360         u_broff1      => u_broff1,
361         u_broff2      => u_broff2,
362         u_off         => u_off,
363         u_opt         => u_opt,
364           -- ADC --
365         ADCrefOut_O   => ADCrefOut,
366         ADCsequence_O => ADCsequence,

368         ProgReg    => ProgReg,
369         ProgInfo   => ProgInfo,
370         newProgReg => newProgReg,
371         ProgRegReset => ProgRegReset,

373         DataFull => DataFull,
374         saveInArray => saveInArray,
375         DataReady => DataReady,
376         SendingDone => SendingDone,

378         ------ 7-Segment ------
379         seg => seg,
380         an  => an,
381         dp  => dp,
382         ----------------------
383         statePart => statePart
384       );

386     I_UART : MiniUART_TL
387       port map (
388         CLK         => CLK,
389         RESET       => RESET,

391         BtnNextArray  => BtnNextArray,
392         BtnBackVAlue  => BtnBackValue,
393         BtnNextValue  => BtnNextValue,
394         ArrayValue    => ArrayValue,

396         DB_UDIFF   => DB_UDIFF,
```

```
397         DB_HB1      => DB_HB1,
398         DB_HB2      => DB_HB2,

400         sigCompUdiff1 => sigCompUdiff1,
401         sigCompUdiff2 => sigCompUdiff2,

403         ---- Inputs ----
404           -- DAC --
405         ref_diff1         => ref_diff1,
406         ref_diff2         => ref_diff2,
407         ref_hb1           => ref_hb1,
408         ref_hb2           => ref_hb2,
409         u_broff1          => u_broff1,
410         u_broff2          => u_broff2,
411         u_off             => u_off,
412         u_opt             => u_opt,
413           -- ADC --
414         ADCrefOut         => ADCrefOut,
415         ADCsequence       => ADCsequence,
416         AMPgain_O         => AMPgain_O,
417         AMP_CS            => AMP_CS,

419         ProgReg     => ProgReg,
420         ProgInfo_O  => ProgInfo,
421         newProgReg => newProgReg,

423         ProgRegReset => ProgRegReset,

425         DataFull_O => DataFull,
426         saveInArray => saveInArray,
427         DataReady => DataReady,
428         SendingDone_O => SendingDone,

430         ---- MiniUART ----
431         TxD_PAD_O => TxD_PAD,
432         RxD_PAD_I => RxD_PAD_I
433       );

435     I_TBadc : TBadc_ads7865
436       port map (
437         RESET    => RESET,
438         CLK      => CLK,
439         -- digital ports
440         DB       => DB,
441         nCS      => ADC_CSn,
442         nWR      => ADC_WRn,
443         nRD      => ADC_RDn,
444         BUSY     => ADC_BUSY,
445         nCONVST  => ADC_CONVSTn,
446         -- analog ports
447         CHA0p    => CHA0p,
448         CHA0m    => CHA0m,
449         CHA1p    => CHA1p,
450         CHA1m    => CHA1m,
451         CHB0p    => CHB0p,
452         CHB0m    => CHB0m,
453         CHB1p    => CHB1p,
454         CHB1m    => CHB1m,
455         refIn    => refIn,
456         refOut   => refOut_i
457       );
458     I_TBdac : TBdac_ad5348
459       port map (
```

```
460       RESET    => RESET,
461       CLK      => CLK,
462       -- digital
463       DB       => DB,
464       nCSd     => DAC_CSn,
465       nWRd     => DAC_WRn,
466       nLDACd   => DAC_LDACn,
467       nRDd     => DAC_RDn,
468       BUFd     => DAC_BUF,
469       GAINd    => DAC_GAIN,
470       Ad       => DAC_A,
471       -- analog ports
472       -- VrefAB  => VrefAB, -- open because hardwired
473       -- VrefCD  => VrefCD,
474       -- VrefEF  => VrefEF,
475       -- VrefGH  => VrefGH,
476       VoutA    => VoutA,
477       VoutB    => VoutB,
478       VoutC    => VoutC,
479       VoutD    => VoutD,
480       VoutE    => VoutE,
481       VoutF    => VoutF,
482       VoutG    => VoutG,
483       VoutH    => VoutH
484     );

486   I_source : tb_source
487     port map (
488       -- VALUE => CHA0p
489       VALUE => sigVal
490     );
491   I_gainVal : SinGain
492     port map (
493       CLK        => CLK,
494       RESET      => RESET,
495       AMPgain    => AMPgain,
496       sigVal     => sigVal,
497       newSigVal => CHA0p
498     );

500 end behaviour;
```

C.1.25 Signal in Abhängigkeit von der Verstärkungsstufe

```
1  ------------------------------------------------------------
2  --  sinGain - die Verstärkung auf das Signal legen
3  --
4  --  Author: A.Arvidsson
5  ------------------------------------------------------------

7  -- <makepackage>
8  --      <component = SinGain>
9  --      <name = arvidsson>
10 --      <project = ada_conv_modsys>
11 --       </component>
12 -- </makepackage>

14 library ieee;
15   use ieee.std_logic_1164.all;
```

```
16   use ieee.numeric_std.all;

18 library work;
19   use work.global_pkg.all;

21 entity SinGain is
22   port (
23     CLK       : in std_logic;
24     RESET     : in std_logic;

26     AMPgain   : in std_logic_vector(2 downto 0);
27     sigVal    : in std_logic_vector(11 downto 0);
28     newSigVal : out std_logic_vector(11 downto 0)
29   );
30 end SinGain;

32 architecture behaviour of SinGain is
33 signal tempVal : integer range 0 to 4095;

35 begin

37   recalc : process(sigVal,AMPgain)
38   begin
39     case AMPgain is
40       when "000" => tempVal <= (to_integer(unsigned(sigVal)) / 15) after 9 ns;
41       when "001" => tempVal <= (to_integer(unsigned(sigVal)) / 10) after 9 ns;
42       when "010" => tempVal <= (to_integer(unsigned(sigVal)) / 8) after 9 ns;
43       when "011" => tempVal <= (to_integer(unsigned(sigVal)) / 6) after 9 ns;
44       when "100" => tempVal <= (to_integer(unsigned(sigVal)) / 3) after 9 ns;
45       when "101" => tempVal <= (to_integer(unsigned(sigVal)) / 17 * 10) after 9 ns;
46       when "110" => tempVal <= (to_integer(unsigned(sigVal)) / 13 * 10) after 9 ns;
47       when "111" => tempVal <= to_integer(unsigned(sigVal)) after 9 ns;
48       when others => tempVal <= to_integer(unsigned(sigVal)) after 9 ns;
49     end case;
50   end process recalc;

52   FF : process(RESET,CLK)
53   begin
54     if RESET = '1' then
55       newSigVal <= sigVal;
56     elsif CLK = '1' and CLK'event then
57       newSigVal <= std_logic_vector(to_unsigned(tempVal,12));
58     end if;
59   end process FF;

61 end behaviour;
```

C.1.26 .do-Datei für die Simulation

```
1 vsim work.tb_toplevel_tl
2 restart
3 radix hex

5 add wave \
6 -group toplevel \
7 -label sys_clk_s {sim:/tb_toplevel_tl/sys_clk_s } \
8 -label clk {sim:/tb_toplevel_tl/clk } \
9 -label reset {sim:/tb_toplevel_tl/reset }
```

```
11 add wave \
12 -group ada \
13 -label adc_busy {sim:/tb_toplevel_tl/i_ada/adc_busy } \
14 -label wr_n_en {sim:/tb_toplevel_tl/i_ada/wr_n_en } \
15 -label db {sim:/tb_toplevel_tl/i_ada/db } \
16 -label adc_csn {sim:/tb_toplevel_tl/i_ada/adc_csn } \
17 -label adc_wrn {sim:/tb_toplevel_tl/i_ada/adc_wrn } \
18 -label adc_rdn {sim:/tb_toplevel_tl/i_ada/adc_rdn } \
19 -label adc_convstn {sim:/tb_toplevel_tl/i_ada/adc_convstn } \
20 -label writeto {sim:/tb_toplevel_tl/i_ada/i_data/writeto } \
21 -label state {sim:/tb_toplevel_tl/i_ada/i_fsm/state } \
22 -label dac_csn {sim:/tb_toplevel_tl/i_ada/i_fsm/dac_csn } \
23 -label dac_wrn {sim:/tb_toplevel_tl/i_ada/i_fsm/dac_wrn } \
24 -label dac_ldacn {sim:/tb_toplevel_tl/i_ada/i_fsm/dac_ldacn } \
25 -label dac_rdn {sim:/tb_toplevel_tl/i_ada/i_fsm/dac_rdn } \
26 -label dac_buf {sim:/tb_toplevel_tl/i_ada/i_fsm/dac_buf } \
27 -label dac_gain {sim:/tb_toplevel_tl/i_ada/i_fsm/dac_gain } \
28 -label dac_a {sim:/tb_toplevel_tl/i_ada/i_fsm/dac_a }

30 add wave \
31 -group ada \
32 -label statepart {sim:/tb_toplevel_tl/i_ada/i_fsm/statepart } \
33 -label checkseq {sim:/tb_toplevel_tl/i_ada/i_fsm/checkseq } \
34 -label firstrun {sim:/tb_toplevel_tl/i_ada/i_fsm/firstrun } \
35 -label seqok {sim:/tb_toplevel_tl/i_ada/i_fsm/seqok } \
36 -label saveinarray {sim:/tb_toplevel_tl/i_ada/i_fsm/saveinarray }

38 add wave \
39 -group ada \
40 -label poweronreset {sim:/tb_toplevel_tl/i_ada/i_fsm/poweronreset } \
41 -label adcgotodacreg {sim:/tb_toplevel_tl/i_ada/i_fsm/adcgotodacreg } \
42 -label adcwrdacreg {sim:/tb_toplevel_tl/i_ada/i_fsm/adcwrdacreg } \
43 -label adcgotodacread {sim:/tb_toplevel_tl/i_ada/i_fsm/adcgotodacread } \
44 -label adcrddacreg {sim:/tb_toplevel_tl/i_ada/i_fsm/adcrddacreg } \
45 -label adcgotoseqreg {sim:/tb_toplevel_tl/i_ada/i_fsm/adcgotoseqreg } \
46 -label adcwrseqreg {sim:/tb_toplevel_tl/i_ada/i_fsm/adcwrseqreg } \
47 -label adcgotoseqread {sim:/tb_toplevel_tl/i_ada/i_fsm/adcgotoseqread } \
48 -label adcrdseqreg {sim:/tb_toplevel_tl/i_ada/i_fsm/adcrdseqreg } \
49 -label adcwrconf {sim:/tb_toplevel_tl/i_ada/i_fsm/adcwrconf } \
50 -label checkseq {sim:/tb_toplevel_tl/i_ada/i_fsm/checkseq } \
51 -label newtestseq {sim:/tb_toplevel_tl/i_ada/i_fsm/newtestseq } \
52 -label dacwrdacreg {sim:/tb_toplevel_tl/i_ada/i_fsm/dacwrdacreg } \
53 -label dacrddacreg {sim:/tb_toplevel_tl/i_ada/i_fsm/dacrddacreg }

55 add wave \
56 -group ada \
57 -label new_data_o {sim:/tb_toplevel_tl/i_ada/new_data_o } \
58 -label db_udiff {sim:/tb_toplevel_tl/db_udiff } \
59 -label db_udiff_o {sim:/tb_toplevel_tl/i_ada/db_udiff_o } \
60 -label datafull {sim:/tb_toplevel_tl/i_uart/datafull } \
61 -label arrayudiff {sim:/tb_toplevel_tl/i_uart/i_adc_array/arrayudiff } \
62 -label arrayhb1 {sim:/tb_toplevel_tl/i_uart/i_adc_array/arrayhb1 } \
63 -label arrayhb2 {sim:/tb_toplevel_tl/i_uart/i_adc_array/arrayhb2 } \
64 -label db_hb1 {sim:/tb_toplevel_tl/db_hb1 } \
65 -label db_hb1_o {sim:/tb_toplevel_tl/i_ada/db_hb1_o } \
66 -label db_hb2 {sim:/tb_toplevel_tl/db_hb2 } \
67 -label db_hb2_o {sim:/tb_toplevel_tl/i_ada/db_hb2_o } \
68 -label readnow {sim:/tb_toplevel_tl/i_ada/readnow } \
69 -label part {sim:/tb_toplevel_tl/i_ada/part } \
70 -label conv_ch {sim:/tb_toplevel_tl/i_ada/i_fsm/conv_channel }

72 add wave \
73 -group ada \
```

```
74 -label proginfopart {sim:/tb_toplevel_tl/i_ada/i_fsm/proginfopart } \
75 -label progregreset {sim:/tb_toplevel_tl/i_ada/i_fsm/progregreset } \
76 -label dataready {sim:/tb_toplevel_tl/i_ada/i_fsm/dataready } \
77 -label reading {sim:/tb_toplevel_tl/i_ada/i_fsm/reading } \
78 -label readallregister {sim:/tb_toplevel_tl/i_ada/i_fsm/readallregister } \
79 -label readoneregister {sim:/tb_toplevel_tl/i_ada/i_fsm/readoneregister } \
80 -label callregister {sim:/tb_toplevel_tl/i_ada/i_fsm/callregister }

82 add wave \
83 -group tb_adc \
84 -label ncs {sim:/tb_toplevel_tl/i_tbadc/ncs } \
85 -label nwr {sim:/tb_toplevel_tl/i_tbadc/nwr } \
86 -label nrd {sim:/tb_toplevel_tl/i_tbadc/nrd } \
87 -label busy {sim:/tb_toplevel_tl/i_tbadc/busy } \
88 -label nconvst {sim:/tb_toplevel_tl/i_tbadc/nconvst } \
89 -label refin {sim:/tb_toplevel_tl/i_tbadc/refin } \
90 -label refout {sim:/tb_toplevel_tl/i_tbadc/refout } \
91 -label outgoingdata {sim:/tb_toplevel_tl/i_tbadc/outgoingdata } \
92 -label setbusy {sim:/tb_toplevel_tl/i_tbadc/setbusy } \
93 -label regconfig {sim:/tb_toplevel_tl/i_tbadc/regconfig } \
94 -label regdac {sim:/tb_toplevel_tl/i_tbadc/regdac } \
95 -label regseq {sim:/tb_toplevel_tl/i_tbadc/regseq } \
96 -label writenow {sim:/tb_toplevel_tl/i_tbadc/writenow } \
97 -label readnow {sim:/tb_toplevel_tl/i_tbadc/readnow } \
98 -label writeseq {sim:/tb_toplevel_tl/i_tbadc/writeseq } \
99 -label count {sim:/tb_toplevel_tl/i_tbadc/count } \
100 -label sequencecounter {sim:/tb_toplevel_tl/i_tbadc/sequencecounter } \
101 -label readfrom {sim:/tb_toplevel_tl/i_tbadc/readfrom } \
102 -label writeto {sim:/tb_toplevel_tl/i_tbadc/writeto } \
103 -label progdac {sim:/tb_toplevel_tl/i_tbadc/progdac } \
104 -label progseq {sim:/tb_toplevel_tl/i_tbadc/progseq } \
105 -label readdac {sim:/tb_toplevel_tl/i_tbadc/readdac } \
106 -label readseq {sim:/tb_toplevel_tl/i_tbadc/readseq } \
107 -label runningbusy {sim:/tb_toplevel_tl/i_tbadc/runningbusy } \
108 -label counter {sim:/tb_toplevel_tl/i_tbadc/counter }

110 add wave \
111 -group uart \
112 -label dat_o {sim:/tb_toplevel_tl/i_uart/dat_o } \
113 -label we_o {sim:/tb_toplevel_tl/i_uart/we_o } \
114 -label stb_o {sim:/tb_toplevel_tl/i_uart/stb_o } \
115 -label ack_i {sim:/tb_toplevel_tl/i_uart/ack_i }

117 add wave \
118 -group uart \
119 -label txd_pad_o {sim:/tb_toplevel_tl/i_uart/txd_pad_o } \
120 -label rxd_pad_i {sim:/tb_toplevel_tl/i_uart/rxd_pad_i } \
121 -label state {sim:/tb_toplevel_tl/i_uart/i_trac/state } \
122 -label index_adcarray {sim:/tb_toplevel_tl/i_uart/i_adc_array/index } \
123 -label matlabdone {sim:/tb_toplevel_tl/i_uart/matlabdone } \
124 -label rssendenable {sim:/tb_toplevel_tl/i_uart/i_trac/rssendenable } \
125 -label progreg {sim:/tb_toplevel_tl/i_uart/progreg } \
126 -label proginfo {sim:/tb_toplevel_tl/i_uart/proginfo } \
127 -label receivedata {sim:/tb_toplevel_tl/i_uart/i_trac/receivedata } \
128 -label datreceive {sim:/tb_toplevel_tl/i_uart/i_trac/datreceive }

130 add wave \
131 -group uart \
132 -label rsgetdone {sim:/tb_toplevel_tl/i_uart/i_trac/rsgetdone } \
133 -label idxrcv {sim:/tb_toplevel_tl/i_uart/i_trac/idxrcv } \
134 -label rsgetenable {sim:/tb_toplevel_tl/i_uart/i_trac/rsgetenable } \
135 -label idxrcvadd {sim:/tb_toplevel_tl/i_uart/i_trac/idxrcvadd } \
136 -label intrx_i {sim:/tb_toplevel_tl/i_uart/i_trac/intrx_i }
```

```
138 add wave \
139 -group uart \
140 -label getdata {sim:/tb_toplevel_tl/i_uart/i_trac/getdata } \
141 -label stateevaluation {sim:/tb_toplevel_tl/i_uart/i_trac/stateevaluation } \
142 -label analyzed {sim:/tb_toplevel_tl/i_uart/i_trac/analyzed } \
143 -label checkenable {sim:/tb_toplevel_tl/i_uart/i_trac/checkenable } \
144 -label datacheckok {sim:/tb_toplevel_tl/i_uart/i_trac/datacheckok }

146 add wave \
147 -group tb_toplevel \
148 -label rxd_pad_i_sendarray {sim:/tb_toplevel_tl/rxd_pad_i_sendarray } \
149 -label index {sim:/tb_toplevel_tl/index } \
150 -label ready {sim:/tb_toplevel_tl/ready } \
151 -label startsending {sim:/tb_toplevel_tl/startsending } \
152 -label dowait {sim:/tb_toplevel_tl/dowait }

154 add wave \
155 -group trace_data \
156 -label datacntcheck {sim:/tb_toplevel_tl/i_uart/i_tracedata/datacntcheck } \
157 -label checknewdata {sim:/tb_toplevel_tl/i_uart/i_tracedata/checknewdata } \
158 -label datawrite {sim:/tb_toplevel_tl/i_uart/i_tracedata/datawrite } \
159 -label matlabdone {sim:/tb_toplevel_tl/i_uart/i_tracedata/matlabdone } \
160 -label sendingdone {sim:/tb_toplevel_tl/i_uart/i_tracedata/sendingdone } \
161 -label datacnt_rst {sim:/tb_toplevel_tl/i_uart/i_tracedata/datacnt_rst } \
162 -label datacnt_end {sim:/tb_toplevel_tl/i_uart/i_tracedata/datacnt_end } \
163 -label datacnt_add {sim:/tb_toplevel_tl/i_uart/i_tracedata/datacnt_add } \
164 -label datacntloop {sim:/tb_toplevel_tl/i_uart/i_tracedata/datacntloop } \
165 -label datacnt {sim:/tb_toplevel_tl/i_uart/i_tracedata/datacnt } \
166 -label byte1 {sim:/tb_toplevel_tl/i_uart/i_tracedata/byte1 } \
167 -label byte2 {sim:/tb_toplevel_tl/i_uart/i_tracedata/byte2 } \
168 -label byte3 {sim:/tb_toplevel_tl/i_uart/i_tracedata/byte3 } \
169 -label byte4 {sim:/tb_toplevel_tl/i_uart/i_tracedata/byte4 } \
170 -label byte5 {sim:/tb_toplevel_tl/i_uart/i_tracedata/byte5 } \
171 -label byte6 {sim:/tb_toplevel_tl/i_uart/i_tracedata/byte6 } \
172 -label p12_en {sim:/tb_toplevel_tl/i_uart/i_tracedata/p12_en } \
173 -label count {sim:/tb_toplevel_tl/i_uart/i_tracedata/count } \
174 -label index {sim:/tb_toplevel_tl/i_uart/i_tracedata/index }

176 add wave \
177 -group CalcPeriod_Gain \
178 -label sigcompudiff1 {sim:/tb_toplevel_tl/i_uart/i_period/sigcompudiff1 } \
179 -label sigcompudiff2 {sim:/tb_toplevel_tl/i_uart/i_period/sigcompudiff2 } \
180 -label sampleintervall {sim:/tb_toplevel_tl/i_uart/i_period/sampleintervall } \
181 -label udiff {sim:/tb_toplevel_tl/i_uart/i_period/udiff } \
182 -label ampgain {sim:/tb_toplevel_tl/i_uart/i_period/ampgain } \
183 -label amp_cs {sim:/tb_toplevel_tl/i_uart/i_period/amp_cs } \
184 -label state {sim:/tb_toplevel_tl/i_uart/i_period/state } \
185 -label timestart {sim:/tb_toplevel_tl/i_uart/i_period/timestart } \
186 -label timestop {sim:/tb_toplevel_tl/i_uart/i_period/timestop } \
187 -label timer {sim:/tb_toplevel_tl/i_uart/i_period/timer } \
188 -label dostart {sim:/tb_toplevel_tl/i_uart/i_period/dostart } \
189 -label udiffmax {sim:/tb_toplevel_tl/i_uart/i_period/udiffmax } \
190 -label udiffmin {sim:/tb_toplevel_tl/i_uart/i_period/udiffmin } \
191 -label gainint {sim:/tb_toplevel_tl/i_uart/i_period/gainint }

193 run 30000 us
```

C.2 Matlab

In diesem Abschnitt wird das Matlab-Skript aufgelistet, das die Aufgaben für die Steuerung des FPGAs übernimmt.

C.2.1 Matlab-Skript

```
% init

close all;
clear all;

twoBytes      = uint32(0);
udiff_cmd     = uint8(0);
adc_value     = uint16(0);
udiff         = uint16(0);
bytesReceived = 0;
errorMessage  = 'none';

Vmax   = 5;   % in Volt
Offset = 2.5;   % in Volt
index  = 1;
runs   = 1;
max    = 64;
udiff_vec = zeros([max 2]);   % Zeile x Spalte (1:max,1:2)
hb1_vec   = zeros([max 2]);
hb2_vec   = zeros([max 2]);

xmin = 0;
xmax = 63;
ymin = 0;
ymax = 2.5;

%%%%
ref_diff1   = uint16(0);
ref_diff2   = uint16(0);
ref_hb1     = uint16(0);
ref_hb2     = uint16(0);
u_broff1    = uint16(0);
u_broff2    = uint16(0);
u_off       = uint16(0);
u_opt       = uint16(0);
gain        = uint16(0);
ADCrefOut   = uint16(0);
ADCsequence = uint16(0);
Scale       = uint16(63);

%% connect to rs232-interface
interfaceObj_ADC_Board_Control = instrfind('Name', 'Serial-COM1');

if isempty(interfaceObj_ADC_Board_Control)
    interfaceObj_ADC_Board_Control = serial('COM1');
else
    fclose(interfaceObj_ADC_Board_Control);
    interfaceObj_ADC_Board_Control = interfaceObj_ADC_Board_Control(1);
end
```

```
51 deviceObj_ADC_Board_Control = icdevice('ADC_Board_Control.mdd',
       interfaceObj_ADC_Board_Control);
52 connect(deviceObj_ADC_Board_Control);


55 %% get gain value
56 gainDone = get(deviceObj_ADC_Board_Control.Control,'gain');
57 gain      = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

59 %% set config register of ADC
60 set(deviceObj_ADC_Board_Control.Control,'ADCconfig',hex2dec('000')); % e.g. "000" =
       PowerOnReset of ADC
61 % Temp = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

63 %% do reset of all registers
64 set(deviceObj_ADC_Board_Control.Control,'regReset',hex2dec('000'));

66 %% set refOut of ADC
67 set(deviceObj_ADC_Board_Control.Control,'ADCrefOut',hex2dec('3FF'));      % min = 205 = x"0
       CD" = 0.5V; max = 1023 = x"3FF" = 2.5V
68 Temp = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

70 %% get refOut of ADC
71 ADCdone   = get(deviceObj_ADC_Board_Control.Control,'ADCrefOut');
72 ADCrefOut = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

74 %% set sequence register of ADC
75 set(deviceObj_ADC_Board_Control.Control,'ADCsequence',hex2dec('230'));
76 Temp = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

78 %% get sequence register of ADC
79 ADCseqDone  = get(deviceObj_ADC_Board_Control.Control,'ADCsequence');
80 ADCsequence = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

82 %% set Channel 1 (ref_diff1) of DAC
83 set(deviceObj_ADC_Board_Control.Control,'ref_diff1',hex2dec('390'));
84 Temp = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

86 %% set Channel 2 (ref_diff2) of DAC
87 set(deviceObj_ADC_Board_Control.Control,'ref_diff2',hex2dec('410'));
88 Temp = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

90 %% set Channel 3 (ref_hb1) of DAC
91 set(deviceObj_ADC_Board_Control.Control,'ref_hb1',hex2dec('000'));
92 Temp = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

94 %% set Channel 4 (ref_hb2) of DAC
95 set(deviceObj_ADC_Board_Control.Control,'ref_hb2',hex2dec('000'));
96 Temp = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

98 %% set Channel 5 (u_broff1) of DAC
99 set(deviceObj_ADC_Board_Control.Control,'u_broff1',hex2dec('000'));
100 Temp = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

102 %% set Channel 6 (u_broff2) of DAC
103 set(deviceObj_ADC_Board_Control.Control,'u_broff2',hex2dec('000'));
104 Temp = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

106 %% set Channel 7 (u_off) of DAC
107 set(deviceObj_ADC_Board_Control.Control,'u_off',hex2dec('400'));    % 4CC für 1,5V; 400 für
       1,25V
108 Temp = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');
```

```
110 %% set Channel 8 (u_opt) of DAC
111 set(deviceObj_ADC_Board_Control.Control,'u_opt',hex2dec('000'));
112 Temp = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

114 %%
115 %% get Channel 1 (ref_diff1) of DAC
116 DACch1done = get(deviceObj_ADC_Board_Control.Control,'ref_diff1');
117 ref_diff1  = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

119 %% get Channel 2 (ref_diff2) of DAC
120 DACch2done = get(deviceObj_ADC_Board_Control.Control,'ref_diff2');
121 ref_diff2  = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

123 %% get Channel 3 (ref_hb1) of DAC
124 DACch3done = get(deviceObj_ADC_Board_Control.Control,'ref_hb1');
125 ref_hb1    = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

127 %% get Channel 4 (ref_hb2) of DAC
128 DACch4done = get(deviceObj_ADC_Board_Control.Control,'ref_hb2');
129 ref_hb2    = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

131 %% get Channel 5 (u_broff1) of DAC
132 DACch5done = get(deviceObj_ADC_Board_Control.Control,'u_broff1');
133 u_broff1   = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

135 %% get Channel 6 (u_broff2) of DAC
136 DACch6done = get(deviceObj_ADC_Board_Control.Control,'u_broff2');
137 u_broff2   = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

139 %% get Channel 7 (u_off) of DAC
140 DACch7done = get(deviceObj_ADC_Board_Control.Control,'u_off');
141 u_off      = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

143 %% get Channel 8 (u_opt) of DAC
144 DACch8done = get(deviceObj_ADC_Board_Control.Control,'u_opt');
145 u_opt      = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

147 %%
148 %% read all register of ADC and DAC
149 readAlldone = get(deviceObj_ADC_Board_Control.Control,'readAll');
150 [temp]      = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');

152 ref_diff1   = temp(1);
153 ref_diff2   = temp(2);
154 ref_hb1     = temp(3);
155 ref_hb2     = temp(4);
156 u_broff1    = temp(5);
157 u_broff2    = temp(6);
158 u_off       = temp(7);
159 u_opt       = temp(8);
160 gain        = temp(9);
161 ADCrefOut   = temp(10);
162 ADCsequence = temp(11);


165 %% dec2hex-function
166 fprintf('\nADCrefOut:   %s',dec2hex(ADCrefOut));
167 fprintf('\nADCsequence: %s',dec2hex(ADCsequence));
168 fprintf('\nref_diff1:   %s',dec2hex(ref_diff1));
169 fprintf('\nref_diff2:   %s',dec2hex(ref_diff2));
170 fprintf('\nref_hb1:     %s',dec2hex(ref_hb1));
171 fprintf('\nref_hb2:     %s',dec2hex(ref_hb2));
172 fprintf('\nu_broff1:    %s',dec2hex(u_broff1));
```

```
173 fprintf('\nu_broff2:    %s',dec2hex(u_broff2));
174 fprintf('\nu_off:       %s',dec2hex(u_off));
175 fprintf('\nu_opt:       %s',dec2hex(u_opt));
176 fprintf('\ngain:        %s',dec2hex(gain));
177 fprintf('\nfrequence:   %7.3d Hz',1./Scale);
178 fprintf('\n');


181 %% get new values from ADC via the three arrays
182 twoBytesDone = get(deviceObj_ADC_Board_Control.Control,'NewValues');

184 index = 1;

186 [twoBytes] = invoke(deviceObj_ADC_Board_Control.Control,'read_channels');
187 disp(length(twoBytes))
188 for i=0:3:190
189     udiff = cast(twoBytes(i+1),'double');
190     hb1   = cast(twoBytes(i+2),'double');
191     hb2   = cast(twoBytes(i+3),'double');

193     udiff_vec(index,1) = udiff .* (Vmax./power(2,12));
194     hb1_vec(index,1)   = hb1   .* (Vmax./power(2,12));
195     hb2_vec(index,1)   = hb2   .* (Vmax./power(2,12));

197     udiff_vec(index,2) = udiff;
198     hb1_vec(index,2)   = hb1;
199     hb2_vec(index,2)   = hb2;

201     index = index + 1;
202 end

204 timeScale1 = bitshift(cast(twoBytes(193),'uint32'),8);
205 timeScale2 = cast(twoBytes(194),'uint32');
206 timeScale  = cast(bitor(timeScale1,timeScale2),'double');

208 fprintf('\ndisplay read data\n')

210 Scale = timeScale.*64.*(80e-9); % time for the whole sinus

212 t = 1:index-1;
213 figure(runs)  % to get an own figure for each run
214 subplot(3,1,1);
215 plot(((t-1).*Scale./63),udiff_vec(t))  % '-1' because index has to be higher than 0 but
        plot starts from 0
216 axis([xmin Scale ymin ymax]);
217 title('UDIFF')
218 xlabel('Zeit in sec');
219 ylabel('Spannung in Volt');
220 subplot(3,1,2);
221 plot(((t-1).*Scale./63),hb1_vec(t))
222 axis([xmin Scale ymin ymax]);
223 title('HB1')
224 xlabel('Zeit in sec');
225 ylabel('Spannung in Volt');
226 subplot(3,1,3);
227 plot(((t-1).*Scale./63),hb2_vec(t))
228 axis([xmin Scale ymin ymax]);
229 title('HB2')
230 xlabel('Zeit in seci');
231 ylabel('Spannung in Volt');


234 %% Daten auswerten und in Hex ausgeben
```

```
235 val = 1:64;
236 ch1hex = dec2hex(udiff_vec(val,2));
237 ch2hex = dec2hex(hb1_vec(val,2));
238 ch3hex = dec2hex(hb2_vec(val,2));

240 %% Interface object output
241 fprintf('\nInterface-Object:');
242 disp(interfaceObj_ADC_Board_Control);

244 %% Device object output
245 fprintf('\nDevice-Object:');
246 disp(deviceObj_ADC_Board_Control);

248 %% disconnect and close interface-object and device-object
249 disconnect(deviceObj_ADC_Board_Control);
250 delete(deviceObj_ADC_Board_Control);
251 delete(interfaceObj_ADC_Board_Control);
```

Abkürzungsverzeichnis

ABS Antiblockiersystem

ADC Analog Digital Converter

AMR Anisotrope Magnetoresistive

AMR anisotroper magnetoresistiver Effekt

DAC Digital Analog Converter

ESZ-ABS Experimentelle digitale Signalverarbeitung und Zustandserkennung für ABS-Sensoren

LSB Least Significant Bit

MSB Most Significant Bit

POR Power On Reset

RMP Radmessplatz

THD Total Harmonic Distortion

UART Universal Asynchronous Receiver Transmitter

VHDL Very-High Speed Integrated Circuit Hardware Description Language

Tabellenverzeichnis

Bildverzeichnis

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 28. Februar 2011
Ort, Datum

Unterschrift