HAW HAMBURG

Bachelorthesis

Jannes Helck

Digitale Signalverarbeitungs-Module für einen Chipentwurf für ein Sensor-Array

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Jannes Helck

Digitale Signalverarbeitungs-Module für einen Chipentwurf für ein Sensor-Array

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Elektrotechnik und 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. Pawel Buczek

Abgegeben am 23. August 2018

Jannes Helck

Thema der Bachelorthesis

Digitale Signalverarbeitungs-Module für einen Chipentwurf für ein Sensor-Array

Stichworte

Sensor-Array, RAM, CORDIC, Digitale Signalverarbeitung, FPGA, Mikrocontroller

Kurzzusammenfassung

In dieser Arbeit werden Verarbeitungsschritte der Signalauswertung eines magnetoresistiven Sensor-Arrays in VHDL entworfen und implementiert. Damit werden Module für ein experimentelles Chipdesign erarbeitet. Die Kommunikation zwischen den Modulen erfolgt über einen zentralen RAM. Mit einem Mikrocontroller werden Debugging-Funktionen und ein externer Zugriff auf den RAM-Inhalt realisiert. Weiterhin wird ein Modul zur Winkelberechnung entwickelt und gemeinsam mit dem RAM-Modul auf einem FPGA gestestet.

Jannes Helck

Title of the bachelor thesis

Digital signal processing modules for a chip design for a sensor array

Keywords

sensor array, RAM, CORDIC, digital signal processing, FPGA, microcontroller

Abstract

This thesis deals with the development and implementation of signal processing steps in VHDL for a magneto-resistive sensor array. Therefore modules for an experimental chip design are developed. The communication between the modules is realized via a central RAM. Debugging functions and external RAM access are realized with a microcontroller. Additionally a module for angle calculation is developed and tested together with the RAM on a FPGA.

Danksagung

An dieser Stelle möchte ich mich bei meinem betreuenden Prüfer Prof. Dr.-Ing. Karl-Ragmar Riemschneider, für die Möglichkeit, diese Bachelorarbeit im Rahmen des ISAR-Projektes zu erstellen, bedanken. Weiterhin bedanke ich mich bei meinem Zweitgutachter Prof. Dr. Pawel Buczek für sein Engagement.

Besonderer Dank gilt Herrn Thorben Schüthe für seinen fachlichen Rat und seine tatkräftige Unterstützung.

Des Weiteren bedanke ich mich bei Herrn Dipl.-Ing. Günter Müller für das Korrekturlesen der vorliegenden Arbeit und seinen fachlichen Rat.

Außerdem bedanke ich mich bei dem gesamten Forschungsteam der Projekte ISAR und BATSEN für das angenehme und kollegiale Arbeitsklima.

Vor allem aber möchte ich meiner Familie danken, ohne deren Unterstützung ich mein Studium niemals hätte absolvieren können.

Inhaltsverzeichnis

Abbildungsverzeichnis **VII**

Tabellenverzeichnis **IX**

Abkürzungen **X**

1 Einleitung **1**
1.1 Motivation 1
1.2 Überblick über das Projekt 1
1.3 Ziel dieser Arbeit 2
1.4 Mathematische Grundlagen des CORDIC Algorithmus 3

2 Aufbau **6**
2.1 Gesamtkonzept 6
2.2 Module auf dem FPGA 7
2.3 Verwendete Hardware 8
2.4 Definition der Schnittstellen 9

3 Konzeption des RAM-Moduls **12**
3.1 Darstellungsformen der Werte 14
3.2 Übergabe und Speicherung der Werte 15
3.3 Größenabschätzung 17

4 Umsetzung des RAM-Moduls **18**
4.1 VHDL-Beschreibung des RAM-Moduls 18
4.2 Definition von Zugriffsfunktionen und -abläufen 21
4.3 Ansteuerung der Signalverarbeitungsmodule 27
4.4 Test-Modul zur Ansteuerung des RAMs 28

5 Entwurf und Umsetzung der Winkelberechnung **31**
5.1 Verfahren zur Winkelberechnung 31
5.2 Konzeption 32
5.3 Praktische Implementierung der Winkelberechnung 32
5.4 Betrachtung der numerischen Genauigkeit 34
5.5 VHDL-Umsetzung 38

6 Funktionstest und Aufwandsabschätzung **41**
6.1 Entwicklung der Mikrocontroller-Steuerung für das RAM-Modul 41
6.2 Test der Gesamtfunktion auf dem FPGA 50
6.3 Test der Winkelberechnung auf dem FPGA 52
6.4 Abschätzung des Syntheseaufwands mit Cadence 54

7 Schlussfolgerungen **55**
7.1 Zusammenfassung 55
7.2 Ausblick 57

Literaturverzeichnis **58**

Anhang

A Diverses **61**

B Quelltexte **63**
B.1 VHDL-Quellcode 63
B.2 C-Quellcode 83
B.3 Matlab/Octave-Quellcode 100

C Cadence-Synthesereports **113**

D Kurzbeschreibung zur Nutzung der Xilinx-Umgebung im Zusammenspiel mit der Cadence-Umgebung **120**

E CD **122**

Abbildungsverzeichnis

2.1 Schematischer Aufbau des Testsystems 6
2.2 Schematische Übersicht der Module auf dem FPGA 7
2.3 Aufbau der Schnittstelle zwischen dem Mikrocontroller und dem FPGA . 9
2.4 Aufbau des Testsystems 11

3.1 Speicherung der Werte im RAM 15
3.2 Schematische Darstellung der Zugriffsverfahren zum Lesen und Schreiben des RAM-Speichers 16

4.1 Blockschaltbild des in VHDL beschriebenen RAM-Moduls 19
4.2 Programmablaufplan des Prozesses zur Modellierung des Verhaltens des RAM-Speichers 20
4.3 Schematische Übersicht des Anschlusses mehrerer Module an den RAM . 23
4.4 Blockschaltbild des Memory Control Blocks 24
4.5 Programmablaufplan zur Beschreibung der Funktion innerhalb des Prozesses des Memory Control Blocks 25
4.6 Blockschaltbild des Module Control Blocks 27
4.7 Zustandsautomat Module Control Block 28
4.8 Blockschaltbild des Test-Moduls 28
4.9 Zustandsautomat zur Realisierung der Funktion des Test-Moduls 30

5.1 Fallunterscheidungen für die praktische Implementierung des CORDIC-Algorithmus zur Winkelberechnung 33
5.2 Berechnetes Winkelergebnis der in Matlab implementierten `atan2_cordic`-Funktion 35
5.3 Betragsmäßiger absoluter Fehler der in Matlab implementierten `atan2_cordic`-Funktion für den im $s2Q9$-Format darstellbaren Bereich der Eingangswerte 36
5.4 Betragsmäßiger absoluter Fehler der in Matlab implementierten `atan2_cordic`-Funktion im Bereich um den Nullpunkt der Eingangswerte 37
5.5 Blockschaltbild des Winkelberechungsmoduls 38
5.6 Zustandsautomat des in VHDL implementierten **cordic-Algorithmus!** zur Winkelberechnung 39

6.1 Strukturierung der Software auf dem EK-TM4C1294XL als Header- und Sourcedatein 42
6.2 Programmablaufplan der parallel ablaufenden Tasks auf dem Mikrocontroller 44
6.3 Zustandsautomat zur Ansteuerung des RAMs über den Memory Control Block 47
6.4 Programmablauf des erstellten Skriptes zum Testen des Winkelberechungsmoduls auf dem FPGA 52

A.1 Ausschnitt eines Signal-Zeit-Diagramms zur Darstellung der Ein- und Ausgangssignale des Test-Moduls 62
A.2 Ausschnitt eines Signal-Zeit-Diagramms zur Darstellung der Ein- und Ausgangsignale des Winkelberechnungsmoduls 62

E.1 Ordnerstruktur der beigefügten CD 122

Tabellenverzeichnis

1.1 Gegenüberstellung magnetischer Sensoren basierend auf dem AMR- und TMR-Effekt 2

2.1 Gegenüberstellung des Entwicklungsboards EK-TM4C1294XL und des Arm Cortex A9 PS 9

3.1 Möglichkeiten der Parametrisierung des RAM-Moduls von AMS 12
3.2 Abschätzung des Speicherbedarfs der Signalverarbeitungsmodule im RAM 17

5.1 Gemittelter Betrag der Differenz der implementierten `atan2_cordic`-Funktion und der Matlab-`atan2`-Funktion für verschiedene Zahlenformate und Iterationszahlen 34

6.1 Festlegung der Interrupt-Priotitäten 42
6.2 Implementierte Debugging-Befehle zur Erstellung von Testskripten in GNU Octave 50
6.3 Berechnete Winkel auf dem FPGA 54
6.4 Übersicht der Ergebnisse der Cadence-Synthese 54

A.1 Pinbelegung der parallelen Schnittstelle zwischen Mikrocontroller und FPGA 61

Abkürzungen

ADC	Analog-Digital-Umsetzer
AMR	Anisotroper magnetoresistiver Effekt
ASIC	Anwendungsspezifische integrierte Schaltung (Application-Specific Integrated Circuit)
BMBF	Bundesministerium für Bildung und Forschung
CORDIC	COrdinate Rotation DIgital Computer
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input/Output
HAW	Hochschule für angewandte Wissenschaften
IO	Input/Output
ISAR	Signalverarbeitung für Integrated Sensor-Arrays basierend auf dem Tunnel-Magnetoresistiven Effekt für den Einsatz in der Automobilelektronik
IP	intellectual property
ISR	Interrupt Service Routine
LSB	Least Significant Bit
MC	Mikrocontroller
MSB	Most Significant Bit
PC	Personal Computer
PL	Programmable Logic
PS	Processing System
RAM	Random Access Memory
SoC	System on Chip
TMR	Tunnel-magnetoresistiver Effekt

UART Universal Asynchronous Receiver Transmitter

USB Universal Serial Bus

VHDL Very High Speed Integrated Circuit Hardware Description Language

1 Einleitung

1.1 Motivation

Magnetische Sensoren erlauben die Erfassung von mechatronischen Messgrößen, wie Winkeln und Winkelgeschwindigkeiten. Sie werden im Automobil in der Motorelektronik, im Bremssystem und anderen Anwendungen eingesetzt.

Die HAW Hamburg arbeitet in diesem Bereich im Forschungsprojekt "ISAR - Signalverarbeitung für Integrated Sensor-Arrays basierend auf dem Tunnel-Magnetoresistiven Effekt für den Einsatz in der Automobilelektronik". In diesem vom Bundesministerium für Bildung und Forschung (BMBF) geförderten Projekt sollen insbesondere die Signalverarbeitung und die Systemarchitektur entwickelt werden.

Mit dem Sensor-Array ist die Erfassung der Lage eines Permanentmagneten möglich. Die Winkelinformation ist die Nutzgröße. Sie wird von magnetischen Störungen und anderen Fehlerquellen beeinflusst. Mit dem Sensor-Array wird eine Fehlerkorrektur möglich.

1.2 Überblick über das Projekt

In technischen Anwendungen werden magnetische Sensoren basierend auf verschiedenen physikalischen Effekten eingesetzt. Als Beispiele lassen sich der anisotrope magnetoresistive Effekt (AMR), der Tunnel-magnetoresistive Effekt (TMR) sowie der Hall-Effekt nennen [1].

In vorangegangen Projekten an der HAW zum Thema magnetischer Sensorik wurde an magnetischen Differenzsensoren basierend auf dem AMR-Effekt geforscht. In diesem Zusammenhang befasst sich die Dissertation von Krey [2] unter anderem mit der Umsetzung eines Diagnosealgorithmus für magnetische ABS-Sensoren auf einem Chip.

Im Projekt ISAR wird der Forschungschwerpunkt auf TMR-Sensoren gelegt. Diese unterscheiden sich in einigen Eigenschaften deutlich von auf dem AMR-Effekt basierenden Sensoren.

Tabelle 1.1: Gegenüberstellung magnetischer Sensoren basierend auf dem AMR- und TMR-Effekt nach Slatter[3]

	AMR-Effekt	TMR-Effekt
$\frac{\Delta R}{R}$	o	++
Empfindlichkeit	+	+
Signal/Rauschen	++	o
Bandbreite	++	o
Leistungsaufnahme	o	++
Temperaturstabilität	+	++
Hysterese	++	o
Miniaturisierbarkeit	o	++

Im Hinblick auf den Einsatz in einem integrierten Sensor-Array bieten TMR-Sensoren klare Vorteile. Sie haben einen geringeren Flächenbedarf und benötigen keine aktiven Bauelemente (Verstärker), da ihre Aussteuerung $\frac{\Delta R}{R}$ größer ist als bei anderen Effekten. Zusätzlich haben sie eine geringere Leistungsaufnahme. Ein detaillierter Vergleich der beiden Sensor-Typen wurde im Rahmen des ISAR-Projekts von Airich [4] durchgeführt.

Das Konzept der Sensormatrix ermöglicht die Lageerfassung eines Permanentmagneten. Dabei ist es möglich Störfelder zu detektieren und zu unterdrücken. Durch die Signalverarbeitung ist eine Fehlerkorrektur möglich.
Zum Zeitpunkt der Erstellung dieser Thesis wurden in der Projektgruppe, als Vorstufe für ein integriertes Sensor-Array, bereits Arrays mit diskreten SMD-Bauelementen aufgebaut. Dabei handelt es sich um ein Array der Größe 8×8. Verwendet wurde der TMR-Sensor AAT 001-10E [5] der Firma NVE. Details zum Aufbau der Sensor-Arrays finden sich in der Abschlussarbeit von Begic [6].

1.3 Ziel dieser Arbeit

In dieser Arbeit sollen Verarbeitungsschritte der Signalauswertung des Sensor-Arrays in VHDL entworfen und implementiert werden. Damit werden Module für ein experimentelles Chipdesign erarbeitet.

Alle entstehenden Module sollen über einen zentralen RAM kommunizieren. Dort werden Eingangs-, Zwischen- und Ausgangswerte gespeichert. Der RAM soll zudem von einem externen Mikrocontroller geladen oder gelesen werden können. Damit können Sensorinformationen oder errechnete Werte des Verarbeitungsschrittes abgelegt und weitergegeben werden.

Werden die Signalverarbeitungsmodule im Verarbeitungsprozess angehalten, kann der RAM-Inhalt für Testzwecke ausgelesen und geprüft werden (Debugging). In dieser Thesis soll das zentrale RAM-Modul und eine geeignete Zugriffssteuerung entworfen werden. Außerdem soll ein Modul für die Winkelberechnung entwickelt und mit dem RAM-Modul gemeinsam getestet werden. Als Plattform stehen die Toolchain des Cadence-Systems und ein Field Programmable Gate Array (FPGA) zur Verfügung. Mit einem Mikrocontroller soll der externe Zugriff realisiert werden.

Weitere Module sollen an den RAM angeschlossen werden. Diese Module werden im Projektteam gemeinsam entwickelt. Abschließend soll zum Zweck einer ersten Aufwandsabschätzung des Chipdesigns eine Synthese mit der Cadence-Umgebung durchgeführt werden.

1.4 Mathematische Grundlagen des CORDIC Algorithmus

An dieser Stelle wird eine Einführung zum COrdinate Rotation DIgital Computer (CORDIC) Algorithmus gegeben, mit welchem in Kapitel 5 das Modul zur Winkelberechnung implementiert wird. Dabei handelt es sich um ein iteratives Verfahren der digitalen Signalverarbeitung zur Berechnung verschiedener trigonometrischer Funktionen, bei dem als Rechenoperationen ausschließlich Additionen und Multiplikationen mit Zweierpotenzen (Bitshift) benötigt werden. Diese lassen sich effizient in digitalen Schaltungen implementieren. Das Verfahren wurde ursprünglich von J. E. Voulder entwickelt (1959) und von J. S. Walter auf die heutige Form erweitert (1971) [7].

Der CORDIC-Algoritmus besitzt zwei Operationsmodi. Den Rotationsmodus, der die Drehung eines Vektors um einen definierten Winkel beschreibt und den Vektormodus, welcher der Berechnung des Ausgangswinkels der Funktion $atan2(y, x)$ entspricht. Im Vektormodus wird der Ausgangswinkel $\boldsymbol{v_0}$ immer so gedreht, das sich der Betrag seiner y-Komponente verringert. Der gesuchte Winkel ergibt sich aus der vorzeichenrichtigen Addition der Teilwinkel [7].

Die folgende Herleitung bezieht sich auf den Vektormodus. Ausgangspunkt ist der Vektor

$$\boldsymbol{v_n} = \begin{pmatrix} x_n \\ y_n \end{pmatrix}. \tag{1.1}$$

Mit der Rotationsmatrix $\boldsymbol{R}$ kann dieser um den Winkel Θ_n gedreht werden

$$\boldsymbol{R_n} = \begin{pmatrix} \cos\Theta_n & -\sin\Theta_n \\ \sin\Theta_n & \cos\Theta_n \end{pmatrix}. \tag{1.2}$$

Dazu wird die Rotationsmatrix $\boldsymbol{R_n}$ mit dem Vektor $\boldsymbol{v_n}$ multipliziert.

$$\boldsymbol{v_n} = \boldsymbol{R_n}\boldsymbol{v_{n-1}} \tag{1.3}$$

Unter Verwendung der beiden mathematischen Identitäten [8]

$$\cos\Theta_n = \frac{1}{\sqrt{1+\tan^2\Theta_n}} \tag{1.4}$$

sowie

$$\sin\Theta_n = \frac{\tan\Theta_n}{\sqrt{1+\tan^2\Theta_n}} \tag{1.5}$$

lässt sich die Rotationsmatrix $\boldsymbol{R_n}$ darstellen als

$$\boldsymbol{R_n} = \frac{1}{\sqrt{1+\tan^2\Theta_n}} \begin{pmatrix} 1 & -\tan\Theta_n \\ \tan\Theta_n & 1 \end{pmatrix}. \tag{1.6}$$

Setzt man nun Gl. 1.6 in Gl. 1.3 ein, ergibt sich der folgende Ausdruck:

$$\begin{pmatrix} x_n \\ y_n \end{pmatrix} = \frac{1}{\sqrt{1+\tan^2\Theta_n}} \begin{pmatrix} 1 & -\tan\Theta_n \\ \tan\Theta_n & 1 \end{pmatrix} \begin{pmatrix} x_{n-1} \\ y_{n-1} \end{pmatrix} \tag{1.7}$$

Nach Ausmultiplizieren des Vektors $\boldsymbol{v_n}$ mit der Rotationsmatrix ergibt sich

$$\begin{pmatrix} x_n \\ y_n \end{pmatrix} = \frac{1}{\sqrt{1+\tan^2\Theta_n}} \begin{pmatrix} x_{n-1} & -y_{n-1}\tan\Theta_n \\ x_{n-1}\tan\Theta_n & y_{n-1} \end{pmatrix}. \tag{1.8}$$

Beschränkt man nun den Winkel Θ auf Werte von $\pm 2^{-n}$, kann die Multiplikation mit der Tangens-Funktion durch eine Multiplikation mit dem Faktor 2^{-n} ersetzt werden. Diese kann effizient als Bitshift-Operation implementiert werden.

Der Faktor σ_n bestimmt die Richtung der Rotation. Wenn y_{n-1} negativ ist, nimmt σ_n den Wert -1 an, ansonsten 1.

$$\begin{pmatrix} x_n \\ y_n \end{pmatrix} = K_n \begin{pmatrix} x_{n-1} & \sigma_n 2^{-n} y_{n-1} \\ \sigma_n 2^{-n} x_{n-1} & y_{n-1} \end{pmatrix} \tag{1.9}$$

Für die Winkelberechnung kann der Skalierungsfaktor K_n vernachlässigt werden.

$$K_n = \frac{1}{\sqrt{1 + 2^{-2n}}} \tag{1.10}$$

Die Berechnung des gesuchten Winkels Θ erfolgt durch die Akkumulation mehrerer Teilwinkel.

$$\Theta_n = \Theta_{n-1} - \sigma_n \alpha_n \tag{1.11}$$

Die Teilwinkel α_n berechnen sich folgendermaßen:

$$\alpha_n = \arctan 2^{-n} \tag{1.12}$$

Entfernt man sich im Hinblick auf die Implementierung von der Matrixschreibweise ergeben sich für die Berechnung der Komponenten von $\boldsymbol{v_n}$ die folgenden Gleichungen:

$$x_n = x_{n-1} + \sigma_n 2^{-n} y_{n-1} \tag{1.13}$$

$$y_n = \sigma_n 2^{-n} x_{n-1} + y_{n-1} \tag{1.14}$$

Für weitere Informationen sei auf die Quellen [7] [9] [10] verwiesen.

2 Aufbau

2.1 Gesamtkonzept

In diesem Kapitel wird ein Überblick über den Aufbau des zu entwickelnden Testsystems gegeben. Das Gesamtsystem besteht aus einem FPGA, einem Mikrocontroller und einem Personal Computer (PC). Der schematische Aufbau des Testsystems ist in Abbildung 2.1 dargestellt.

Der FPGA dient als Prototyp für das spätere Design einer anwendungssspezifischen integrierten Schaltung (ASIC). Auf ihm werden die Signalverarbeitungsmodule implementiert. Der RAM als gemeinsame Kommunikationsschnittstelle zwischen den Signalverarbeitungsmodule befindet sich ebenfalls auf dem FPGA.

Der Mikrocontroller ermöglicht das externe Auslesen und Beschreiben des RAMs. Weiterhin dient er zum Steuern des Verarbeitungsprozesses der Signalverarbeitungsmodule. Auf ihm werden Debugging-Funktionalitäten zum Testen der Signalverarbeitungsmodule implementiert. Da auf dem späteren Chipdesign keine serielle Schnittstelle vorgesehen ist, wird zur Kommunikation zwischen Mikrocontroller und FPGA ein paralleles General Purpose Input/Output (GPIO)-Interface implementiert.

Um eine Eingabemöglichkeit für Testdaten und Debbuging-Befehle zu schaffen wird das System um einen PC erweitert. Als Verbindung zwischen PC und Mikrocontroller wird eine serielle Schnittstelle verwendet. Als Software auf dem PC kommt GNU Octave zum Einsatz. In Octave lassen sich sowohl Testdaten für das Sensor-Array bereitstellen, als auch Skripte mit sequentellen Abläufen von Debugging-Befehlen erstellen. Dies hat den Vorteil, dass verschiedene Testabläufe ausgeführt werden können, ohne den Programmcode auf dem Mikrocontroller anpassen zu müssen.

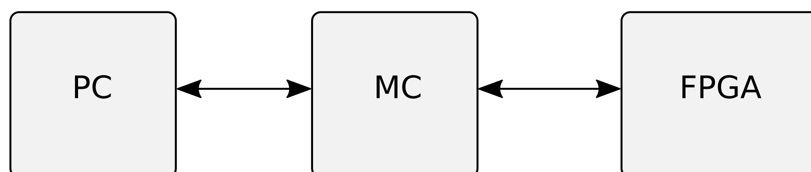


Abbildung 2.1: Schematischer Aufbau des Testsystems

2.2 Module auf dem FPGA

Im Folgenden wird die Modulstruktur auf dem FPGA näher erläutert, da diese wichtig für die Einordnung der folgenden Kapitel in das Gesamtsystem ist. Abbildung 2.2 zeigt alle auf dem FPGA zu implementierenden Module und deren Schnittstellen. Die Funktion der einzelnen Module wird im folgenden beschrieben:

- Der RAM dient als zentrale Schnittstelle zwischen den Signalverarbeitungsmodulen. Diese können direkt auf den RAM zugreifen und Werte auslesen und schreiben. Dies setzt voraus, dass die entsprechenden Speicheradressen in den Modulen hinterlegt sind.
- Der Memory Control Block dient dem externen RAM-Zugriff durch den Mikrocontroller. Da die Anzahl der zur Verfügung stehenden Ports auf dem FPGA begrenzt ist, ist es nicht möglich, jedes Daten- und Adressbit für den RAM parallel anzulegen bzw. auszulesen. Der Memory Control Block übernimmt daher die Aufgabe eines Multiplexers/Demultiplexers.
- Der Module Control Block dient der Aktivierung der Signalverarbeitungsmodule. In ihm ist die Ablaufreihenfolge der Module hinterlegt. Es ist jedoch möglich, die Ansteuerung von extern durch den Mikrocontroller zu übernehmen.
- Die Signalverarbeitungsmodule dienen zur Verarbeitung der Rohdaten des Sensor-Arrays. Ein Modul ist die Winkelberechnung. Alle weiteren Module werden im Projektteam entwickelt und sind daher nicht unmittelbar Teil dieser Arbeit.

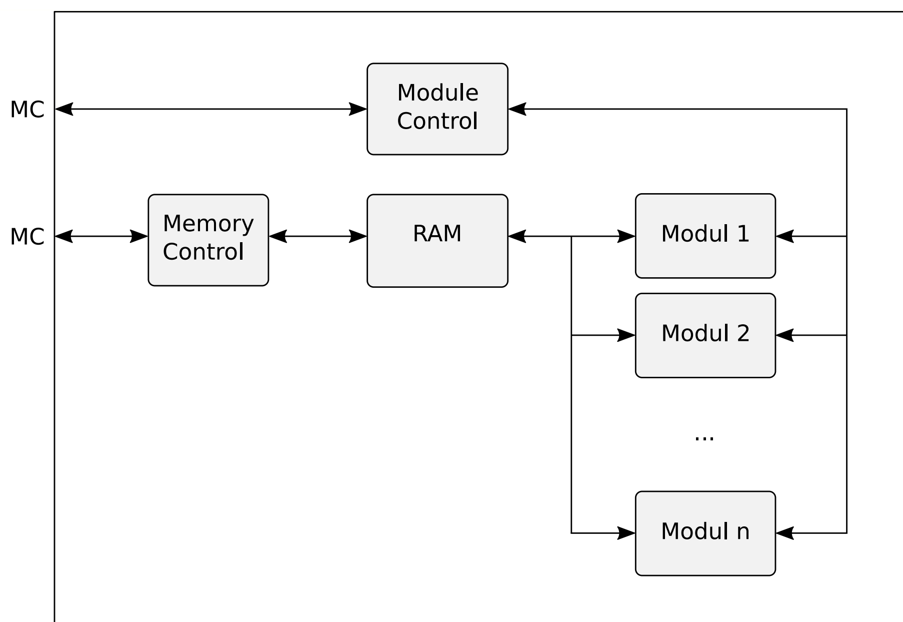


Abbildung 2.2: Schematische Übersicht der Module auf dem FPGA

2.3 Verwendete Hardware

Als FPGA-Entwicklungsplattform steht ein Zedboard mit einem Zynq-7000 All Programmable System on Chip (SoC) Z-7020 zur Verfügung. Das Zynq-7000 SoC lässt sich unterteilen in ein Processing System (PS) sowie eine Programmable Logic (PL). Die PL besteht aus einem Artix-7 FPGA mit folgenden Kenngrößen [11]:

- 85K programmierbare Logikzellen
- 53,200 Look-up Tables
- 106,400 Flip-Flops
- 4.9 Mb Block RAM (140 × 36 Kb Blöcke)
- 220 DSP Slices

Das PS besteht aus einem ARM-Cortex A9 Dualcore Prozessorsystem mit folgenden Eigenschaften [11] [12]:

- bis zu 667 Mhz Taktrate
- 32-bit Architektur
- 32 KB L1 Cache, 512 KB L2 Cache
- 256 KB On Chip Memory

Peripheriemodule für die Anbindung an serielle Schnittstellen und Bussysteme sind ebenfalls vorhanden.

Die Verwendung des Mikroprozessors auf dem Zedboard hat jedoch einige Nachteile. Die FPGA-Implementierung dient als Prototyp für ein Chipdesign. Durch Verwendung des PS auf dem Zedboard sind die Testmöglichkeiten auf dieses beschränkt. Mit einem separaten Entwicklungsboard kann das Testsystem auch zum Testen des fertigen Chips genutzt werden.

Das Connected LaunchPad Evalution Kit EK-TM4C1294XL der Firma Texas Instruments bietet sich für diese Aufgabe an. Es wurde im Projektteam bereits mehrfach verwendet. Auf dem Board befindet sich ein ARM-Cortex M4 Prozessor mit folgenden Eigenschaften [13]:

- bis zu 120 Mhz Taktrate
- 32-bit Architektur
- 1024 KB Flash Speicher, 256 KB SRAM, 6 KB EEPROM

Tabelle 2.1: Gegenüberstellung des Entwicklungsboards EK-TM4C1294XL und des Arm Cortex A9 PS

	EK-TM4C1294XL		Arm Cortex A9 PS
+	Know-how im Projektteam vorhanden	+	keine Verdrahtung notwendig
+	kann ebenfalls zum Testen des Chipdesigns verwendet werden	–	geringe Flexibiltät, Testprogramm auf ZYNQ-7000 beschränkt
–	Zusätzlicher Verdrahtungsaufwand	–	Einarbeitung notwendig

Das EK-TM4C1294XL verfügt ebenfalls über diverse serielle Kommunikationsschnittstellen. In Tabelle 3.1 werden die Vor- und Nachteile der Verwendung der beiden Prozessorsysteme gegenübergestellt. Aufgrund der höheren Flexibilität und des bereits vorhandenen Wissen im Umgang mit dem EK-TM4C1294XL Entwicklungsboard soll es in dieser Arbeit verwendet werden.

2.4 Definition der Schnittstellen

Die Verwendung des EK-TM4C1294XL Entwicklungsboard setzt die Implementierung einer parallelen Schnittstelle voraus. Auf der Mikrocontrollerseite bieten sich hierfür die GPIOs der Pinleisten X6 und X8 an, diese lassen sich kompakt mittels Flachbandkabel und Pfostensteckern mit dem Zedboard verbinden. Auf dem Zedboard stehen mit den Pmod-Adaptern 32 digitale IOs zur Verfügung [12]. Abbildung 2.3 zeigt den Aufbau der parallelen Schnittstelle mit den vorgesehenen Signalen. Eine vollständige Übersicht über die Pinbelegung ist im Anhang in Tabelle A.1 zu finden.

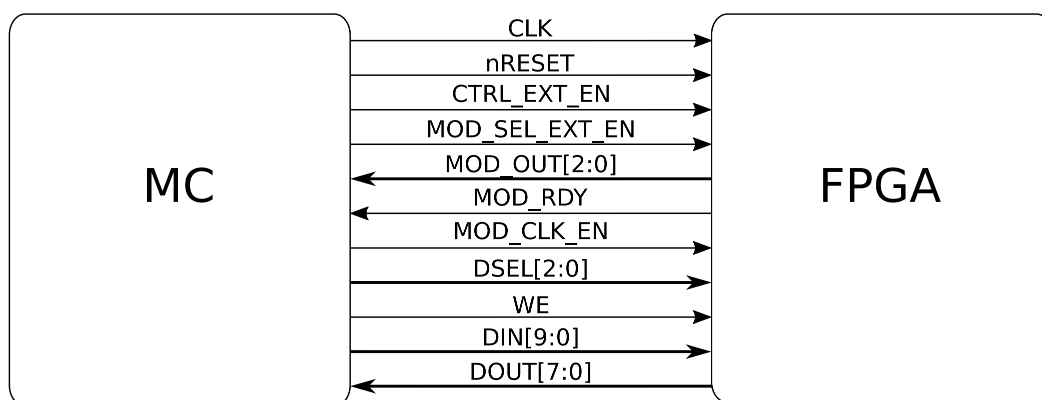


Abbildung 2.3: Aufbau der Schnittstelle zwischen dem Mikrocontroller und dem FPGA

Im Folgenden wird die Funktion der einzelnen Signale aus Abbildung 2.3 beschrieben:

- `CLK`: Der Takt zur Ansteuerung der digitalen Logik auf dem FPGA wird durch den Mikrocontroller generiert.
- `nRESET`: Das Reset-Signal wird separat an alle Module auf dem FPGA geführt. Es ist als Low-Aktiv und asynchron definiert.
- `CTRL_EXT_EN`: Mit dem Control Extern Enable-Signal lässt sich die Verbindung zwischen dem RAM und den Signalverarbeitungsmodulen unterbrechen. Dies ist erforderlich wenn das aktive Signalverarbeitungsmodul zum Zeitpunkt eines externen RAM-Zugriffs ebenfalls auf den Speicher zugreift.
- `MOD_SEL_EXT_EN`: Das Module Select Extern Enable-Signal ermöglicht die externe Aktivierung der Signalverarbeitungsmodule. Die jeweilige Modulnummer wird in diesem Modus über das Signal `DSEL` (Data Select) übertragen.
- `MOD_OUT`: Die zugeordnete Modulnummer des aktiven Signalverarbeitungsmoduls wird über das Module Out-Signal ausgegeben.
- `MOD_RDY`: Das Module Ready-Signal wird aktiviert, wenn das aktive Signalverarbeitungsmodul seinen Verarbeitungsprozess beendet hat.
- `MOD_CLK_EN`: Mithilfe des Module Clock Enable-Signals lässt sich der Takt für die Signalverarbeitungsmodule an- bzw. ausschalten.
- `DSEL`: Durch das Data Select-Signal wird die Position eines 8 Bit Blocks einer Speicheradresse definiert (z.B. ob es sich bei gegebenen Daten um die ersten oder zweiten 8 Bit im RAM handelt).
- `WE`: Wird das Write-Enable auf "1" gesetzt aktiviert es den Schreib-Modus des RAMs. Im Zustand "0" aktiviert es den Lese-Modus.
- `DIN`: Mit dem Data In-Signal können sequentiell 8 Bit, die durch das Memory Control Modul zusammengefügt werden, in den RAM geschrieben werden. Die Adresse zum jeweiligen Schreib/Lesezugriff wird ebenfalls über dieses Signal übertragen.
- `DOUT`: Das Data Out-Signal dient zum Auslesen der Daten aus dem RAM. Durch den Memory Control Block wird der Inhalt einer Speicheradresse in 8 Bit Blöcke zerlegt. Diese lassen sich sequentiell übertragen.

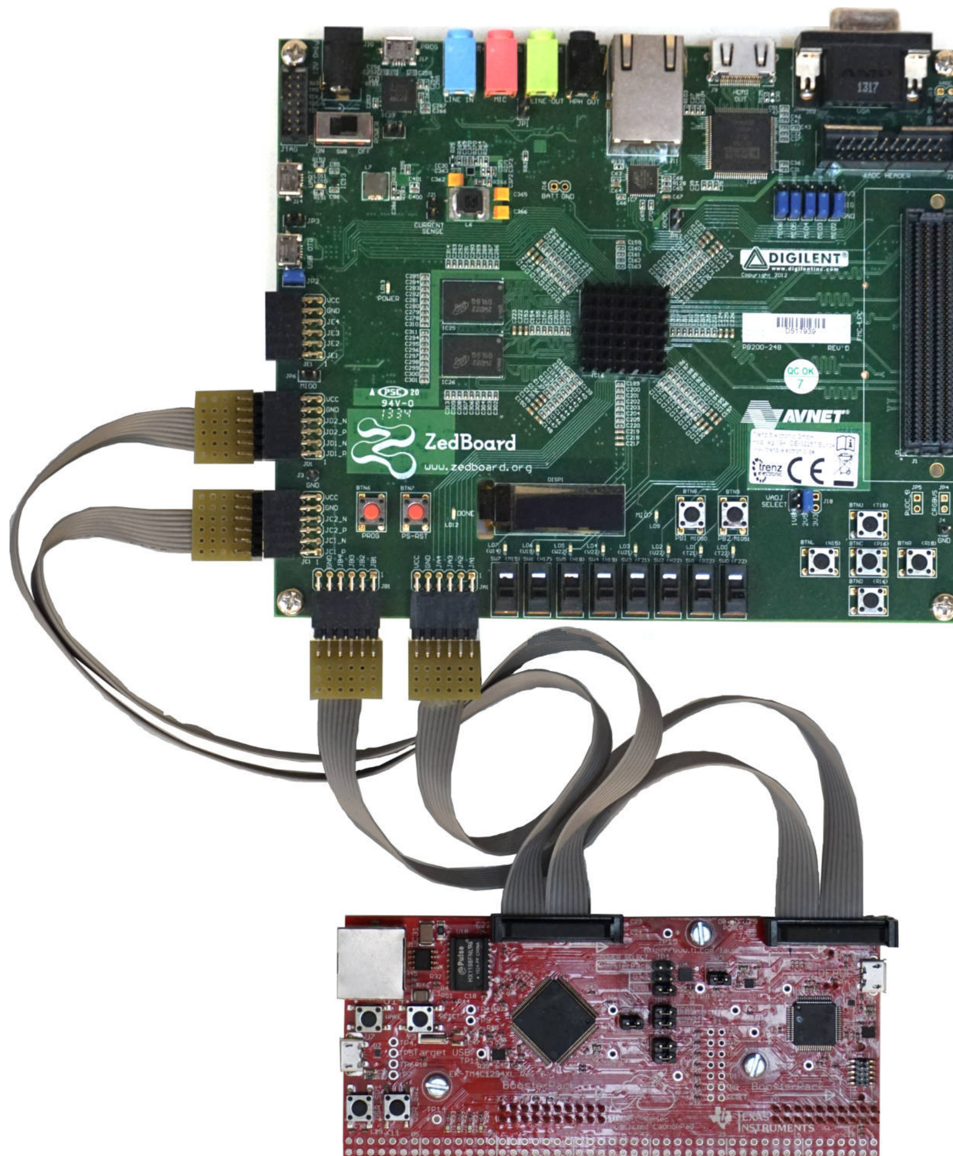


Abbildung 2.4: Aufbau des Testsystems

Abbildung 2.4 zeigt das Zedboard gemeinsam mit dem EK-TM4C1294XL Launchpad, welche auf einer Grundplatte montiert wurden. Zu sehen ist weiterhin die mit Flachbandkabeln realisierte Schnittstelle zwischen beiden Plattformen.

Für die serielle Verbindung zwischen PC und Mikrocontroller wird eine Verbindung als Universal Asynchronous Receiver Transmitter (UART) gewählt. Das UART-Protokoll wird von Octave durch Einbindung des "Instrument Package" unterstützt. Weiterhin spricht für die Wahl dieser Schnittstelle, dass sie in Verbindung mit Octave und dem EK-TM4C1294XL bereits in anderen Arbeiten im Projektteam genutzt wird. Die Verbindung lässt sich einfach über USB-Schnittstellen am Entwicklungsboard und PC realisieren.

3 Konzeption des RAM-Moduls

In diesem Kapitel wird die Konzeption des RAMs beschrieben. Basierend auf der Zielsetzung dieser Arbeit werden die Anforderungen an den Speicher ermittelt. Im folgenden werden die Vor- und Nachteile verschiedener Implementierungsvariaten verglichen. Darauf aufbauend werden dann die Designentscheidungen getroffen.

Für die spätere Implementierung auf dem ASIC bietet die Fa. AMS AG, bei der der Chip gefertigt werden soll, spezielle IP-Cores für Speicher-Module an. Diese haben gegenüber einem in VHDL modellierten Speicher den Vorteil, dass sie optimiert wurden in Bezug auf ihren Bedarf an Ressourcen, wie der benötigten Chipfläche oder den Zugriffszeiten. Eine Implementierung als IP-Core ist also vorzuziehen. Tabelle 3.1 zeigt die Möglichkeiten der Parametrisierung des IP-Speichermoduls von AMS.

Tabelle 3.1: Möglichkeiten der Parametrisierung des RAM-Moduls von AMS (Quelle AMS)

Model	Diffusion ROM	Dual Port RAM	Single Port RAM
Data Bus	Bidirectional	Separated I/O	
Number of Words	$128 \rightarrow 32768$	$2 \cdot 10^7 \rightarrow 2 \cdot 10^{15}$	
Number of Bits/Word	8	16	32
Process	C35	H35	S35
Metalization	Triple Metal		
Voltage	3,3 V		

Auf die Auswahl einiger Parameter kann an dieser Stelle bereits eingegangen werden:

- Model: Da der Speicher beschrieben werden soll, kommen nur die beiden RAM-Varianten in Frage. Die Dual Port Ausführung ermöglicht den gleichzeitigen Speicherzugriff durch zwei Systeme. Dies wäre denkbar für eine getrennte Realisierung des externen Speicherzugriffs und des Speicherzugriffs der Signalverarbeitungsmodule. Alternativ kann bei der Single Port Variante die Verbindung zu den Signalverarbeitungsmodulen getrennt werden, sobald ein externer Speicherzugriff erfolgt. Aufgrund einer erfolgreichen ersten Implementierung der Ansteuerung und der besseren Kontrolle über den Speicherzugriff wird die Single Port Variante gewählt.
- Data Bus: Es wird das Modul mit separaten Datenein- und -ausgängen gewählt. Dies hat den Vorteil einer einfacheren Ansteuerung des RAM-Moduls. Beim internen Zugriff durch die Signalverarbeitungsmodule ist die Anzahl der parallelen Leitungen unkritisch. Der externe Zugriff wird über das Memory Control Modul gesteuert, sodass die RAM-Ports nicht direkt nach außen geführt werden.
- Process: Der Chip soll mit einem 0,35 μm CMOS Prozess gefertigt werden (C35).

Die Bestimmung der Bits pro Wort und des Bedarfs an Wörtern erfolgt in den Abschnitten 3.2 und 3.3.

Der gewählte IP-Core von AMS steht nur für das Chipdesign zur Verfügung. Auf dem Zedboard stehen in der verwendeten Design-Umgebung Vivado ebenfalls IP-Cores für Speichermodule zur Verfügung. Obwohl diese über Parametrierungsmöglichkeiten verfügen, erreichen sie nicht den Flexibilitätsgrad einer Modellierung in VHDL. Da der FPGA ein Prototyp für den Chipentwurf ist, soll der Speicher dort sich nach außen hin genauso verhalten wie der auf dem Chip. Die Implementierung des RAMs auf dem Zedboard erfolgt daher in VHDL.

Für den Speicher-IP-Core steht ein VHDL-Modell zur Verhaltenssimulation zur Verfügung. Dies ermöglicht einen direkten Vergleich des externen Verhaltens der beiden Speicher.

3.1 Darstellungsformen der Werte

Die Berechnungen der Signalverarbeitungsmodule basieren auf den Ausgangsdaten des Sensor-Arrays. Dieses wird mit einer Spannung von 3,3 V angesteuert. Nach einer Offsetbereinigung variieren die Ausgangsspannungen zwischen -1,65 V und 1,65 V. Da somit ein definierter Wertebereich vorliegt, ist eine Darstellung der Werte in einem Festkomma-Zahlenformat sinnvoll. Berechnungen im Gleitkommaformat benötigen deutlich mehr Hardwareressourcen.

Dualzahlen, die sowohl einen ganzzahligen und einen gebrochenen Anteil besitzen, lassen sich im Q-Format darstellen. Eine Darstellung negativer Zahlen ist damit ebenfalls möglich. Die Darstellung dieses Formats erfolgt nach [14, S.82-84].
Das Format $smQn$ besteht aus m Vorkomma- und n Nachkommabits. Durch das zusätzliche Vorzeichenbit gilt für die Bitbreite k:

$$k = m + n + 1 \tag{3.1}$$

Die dezimale Interpretation des Q-Formats erfolgt nach Gleichung 3.2:

$$z = -b_m \cdot 2^m + \sum_{i=m-1}^{0} b_i \cdot 2^i + \sum_{i=1}^{n} b_i \cdot 2^{-i} \tag{3.2}$$

Zur Umsetzung der analogen Werte des diskret aufgebauten Sensor-Arrays wird ein 12-Bit ADC verwendet. Diese Auflösung soll auch weiterhin beibehalten werden. Durch die begrenze Auflösung ist es nicht sinnvoll, die Messwerte im RAM mit einer höheren Auflösung als 12 Bit zu speichern.

Für die Darstellung der Rohdaten des Arrays im RAM wird ein Zahlenformat mit Vorzeichenbit benötigt. Um den Wertebereich darstellen zu können, wird ein Vorkommabit benötigt. Damit wird der Wertebereich $[-2; 2)$ abgedeckt. Die restlichen 10 Bits dienen der Darstellung der Nachkommastellen. Dies ermöglicht eine Auflösung von 2^{-10}, also $9,765625 \cdot 10^{-4}$. Damit ergibt sich das Format $s1Q10$.
Die im Projektteam entwickelten Signalverarbeitungsmodule verwenden ebenfalls das $s1Q10$-Format.

Das Winkelberechnungsmodul soll jeden Ausgangswinkel darstellen können. Die Ausgangswinkel varieren im Bogenmaß von $-\pi$ bis π. Daher wird der berechnete Ausgangswinkel im $s2Q9$-Format im RAM abgelegt. Dieses Format umfasst den Wertebereich $[-4; 4)$ und hat eine Auflösung von 2^{-9} bzw. $1,953125 \cdot 10^{-3}$. Dies entspricht im Gradmaß einem Winkel von $0,1119°$.

3.2 Übergabe und Speicherung der Werte

Für die Anzahl der Bits pro Wort im Speicher werden nach den Überlegungen im vorherigen Abschnitt mindestens 12 Bit benötigt. Laut Tabelle 3.1 kommen als Wortbreiten also 16 oder 32 Bit in Frage.

Die Verwendung von 32 Bit Wörtern hat den Vorteil, dass pro Speicherzugriff 2 Werte gelesen oder geschrieben werden können. Damit wird die benötigte Anzahl an Zugriffen bei der gleichen Datenmenge halbiert. Ein weiterer Vorteil für die Signalverarbeitung ist, dass Imaginär- und Realteil der komplexen Sensordaten gemeinsam in einem RAM-Wort gespeichert werden können.

Abbildung 3.1 zeigt die Anordnung der Werte im RAM. Zur Speicherung der Werte werden die Bits 31-20 bzw. 16-4 verwendet. Die übrigen Bits werden nicht verwendet. Die Daten selbst werden so gespeichert, dass das Most Significant Bit (MSB), also das Vorzeichenbit, auch das höchste Bit im Speicherwort ist (innerhalb der jeweiligen 12 Bit).

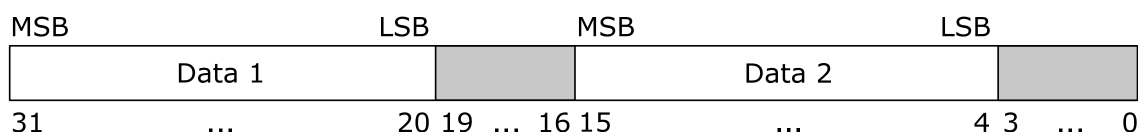
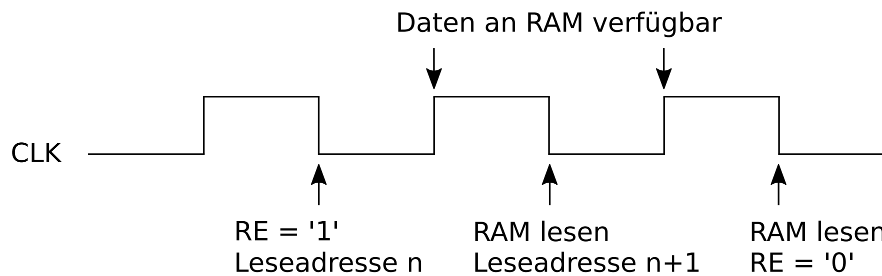
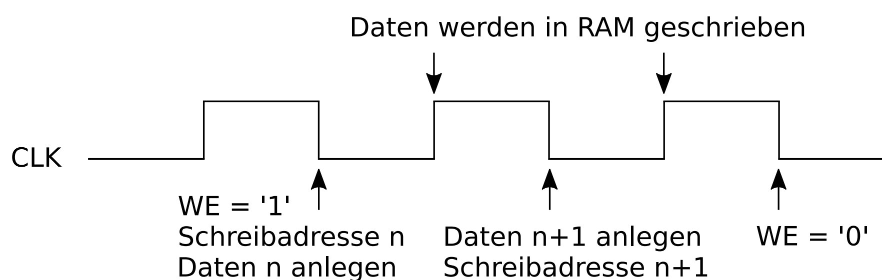


Abbildung 3.1: Speicherung der Werte im RAM



(a) Lese-Vorgang für zwei Wörter aus dem RAM



(b) Schreib-Vorgang für zwei Wörter in den RAM

Abbildung 3.2: Schematische Darstellung der Zugriffsverfahren zum Lesen und Schreiben des RAM-Speichers

Um einen Zugriff auf einen RAM-Speicher zu realisieren muss dieser in einer vorgegeben Sequenz angesteuert werden. Ein Beispiel für die Ansteuerung eines RAMs befindet sich in [15, S.190-196].

Abbildung 3.2 zeigt beispielhaft einen lesenden und einen schreibenden Zugriff für jeweils 2 Wörter. Es handelt sich hier um einen taktsynchronen RAM. Zu beachten ist, dass das Anlegen der Signale an den RAM stehts zur negativen Taktflanke erfolgt. Die Antwort des Speichers erfolgt jeweils an der darauffolgenden positiven Taktflanke. Die Sequenzen lässen sich auf beliebig viele Lese- oder Schreibzugriffe erweitern.

Die Nutzung der negativen Tanktflanke hat zwei wesentliche Vorteile:

- Die Anzahl der benötigten Takte für einen Speicherzugriff wird halbiert.
- Es wird vermieden, Signale an den RAM bei positiver Tankflanke anzulegen. Aufgrund der Taktverzögerung durch Signallaufzeiten wird ein angelegtes Signal mit hoher Warscheinlichkeit erst beim nächsten Takt verarbeitet. Dies erfordert zusätzliche Takte, was die benötigte Zeit für Speicherzugriffe unnötig erhöht.

3.3 Größenabschätzung

Zum Zweck einer Größenabschätzung wird der Speicherbedarf der einzelnen Signalverarbeitungsmodule erfasst. Bei der Auflistung handelt es sich um die zum Zeitpunkt der Bearbeitung der Thesis definierten Module. Weitere Module werden zu späterem Zeitpunkt hinzukommen.

Werden die Werte wie in Abschnitt 3.2 beschrieben im RAM gespeichert, ergibt sich ein Gesamtbedarf von 1134 Wörtern (siehe Tabelle 3.2). Prinzipiell lassen sich aber alle Werte nach Einlesen durch das Folgemodul wieder überschreiben. Außgenommen davon sind die Winkelinformation (das Endergebnis), die Twiddle-Matrix und die Filterkoeffizienten.
Daher wird ein Speicher mit 1024 Wörtern gewählt. Aus der Anzahl der Wörter ergibt sich die Breite der Adressleitung mit 10 Bit.

Tabelle 3.2: Abschätzung des Speicherbedarfs der Signalverarbeitungsmodule im RAM

Anwendung	Ausgangswerte	Wörter im RAM
Sensor-Rohdaten	$2 \times 64 \times 12$ Bit	64
Interpolation	$2 \times 225 \times 12$ Bit	225
2D-DFT	$2 \times 225 \times 12$ Bit	225
Filter-Daten	$2 \times 225 \times 12$ Bit	225
2D-IDFT	$2 \times 225 \times 12$ Bit	225
Winkelinformation	1×12 Bit	1
Twiddle-Matrix	$2 \times 49 \times 12$ Bit	49
Filterkoeffizienten	$2 \times 120 \times 12$ Bit	120

4 Umsetzung des RAM-Moduls

In diesem Kapitel wird die Implementierung des RAM-Moduls in VHDL beschrieben. Weiterhin werden die Module beschrieben, die nicht direkt der Signalverarbeitung zugeordnet sind, aber zum Ablauf notwendig sind. Der entwickelte VHDL-Code wird mit dem Cadence-Tool NC-Sim simuliert. Anschließend erfolgt eine Synthese in der Vivado-Umgebung der Firma Xilinx, um den Code auf dem Zedboard zu testen.Die letztendliche Synthese innerhalb der Cadence-Umgebung erfolgt in Kapitel 6.4. Für die Nutzung der Vivado Design Suite im Zusammenspiel mit der Cadence-Umgebung in dieser Arbeit befindet sich eine Kurzbeschreibung im Anhang D.

Bei der Implementierung eines RAM-Speichers auf einem FPGA sind zwei Varianten denkbar. Der Speicher kann durch digitale Logikressourcen modelliert werden, diese Form wird als Distributed-RAM bezeichnet. Die zweite Form ist die Implementierung als Block-RAM, hier werden dedizierte Speicherblöcke verwendet. Da bei dem Distributed-RAM auf die vergleichsweise teuren Logikressourcen zugegriffen wird, ist die Implementierung als Block-RAM vorzuziehen [14].

4.1 VHDL-Beschreibung des RAM-Moduls

Die vollständige VHDL-Beschreibung des Speichermoduls befindet sich in Listing B.2. Das Blockschaltbild des in VHDL beschriebenen RAM-Moduls ist in Abbildung 4.1 dargestellt. Ein Beispiel zur Beschreibung eines RAMs in VHDL befindet sich in [15, S.190-196]. Anhand dieses Beispiels wurde der hier beschriebene VHDL-Code entwickelt. Die Implementierung des RAM-Speicher erfolgt als Block-RAM.

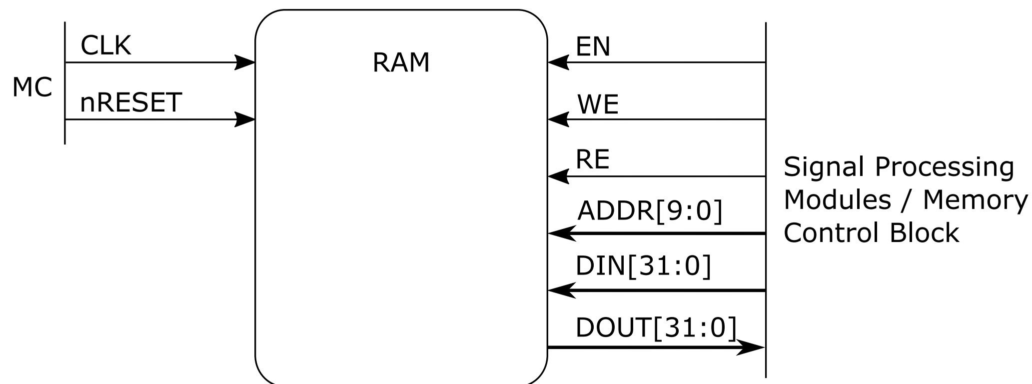


Abbildung 4.1: Blockschaltbild des in VHDL beschriebenen RAM-Moduls

Die Definition des Speichertyps erfolgt durch die Zeilen 64-65 im VHDL-Code. Die Speicherung der Wörter im RAM erfolgt durch das Array `BRAM`. Gemäß den Überlegungen des Kapitels 3 hat das Array eine Größe von 1024 Wörter mit einer Länge von 32 Bit.

Bei der Festlegung der Designentscheidung steht im Vordergrund, dass das beschriebene Speichermodul sich nach außen wie das IP-Core-Modul auf dem zu entwerfenden Chip verhält. So ist die Entscheidung, alle Ein- und Ausgänge als `std_logic` bzw. `std_logic_vector` zu definieren, in der Kompatibilität zum Chipentwurf begründet. Dies gilt ebenfalls für das Verhalten des RAMs, welches durch den Prozess `RAM_P` beschrieben wird. Das Verhalten des Prozesses wird in Abbildung 4.2 beschrieben.

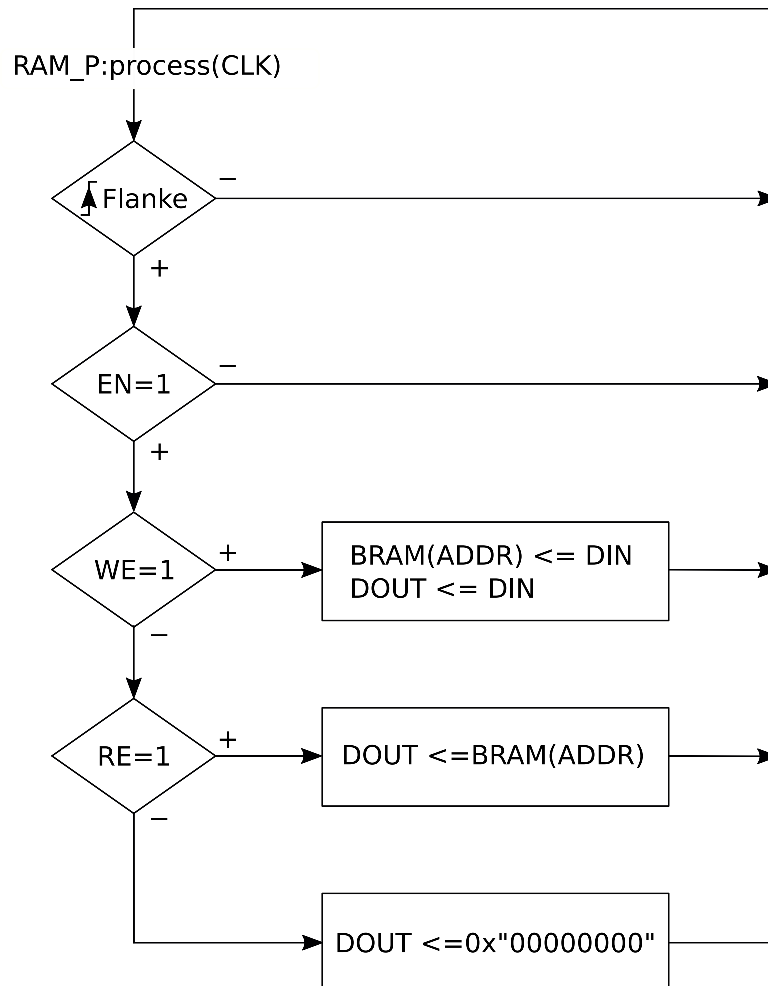


Abbildung 4.2: Programmablaufplan des Prozesses zur Modellierung des Verhaltens des RAM-Speichers

Eine Aktion innerhalb des Prozesses kann nur zum Zeitpunkt einer positive Taktflanke stattfinden. Weiterhin muss der Enable-Eingang des RAM-Moduls gesetzt sein. Bei einem High-Pegel am Write Enable-Signal wird der am Dateneingang `DIN` anliegende Wert in das der anliegenden Adresse ensprechende Wort im Speicherarray geschrieben. Ist das Write Enable-Signal nicht gesetzt, stattdessen aber der Read Enable-Eingang, so wird das Speicherwort entsprechend der anliegenden Adresse auf den Datenausgang `DOUT` geschrieben. Dem Schreibvorgang wird somit eine höhere Priorität eingeräumt als dem Lesevorgang.

Zu Beachten ist, dass der `nRESET`-Eingang keinerlei Funktionalität besitzt. Dies ist einem unverhältnismäßig hohen Hardwareaufwand bei der Implementierung einer Speicher-Rücksetzfunktion geschuldet. In der FPGA-Umgebung ist eine Reset-Funktion nicht zwangsläufig notwendig, da sich das Programm einfach neu laden lässt. Dies ist bei der Chip-Implementierung nicht möglich, daher wird dort eine Speicher-Rücksetzfunktion benötigt. Aus Kompatibilitätsgründen wurde der Eingang beibehalten.

4.2 Definition von Zugriffsfunktionen und -abläufen

Die Zugriffsverfahren zum Lesen und Schreiben des RAMs durch die Signalverarbeitungsmodule wurden bereits in Abschnitt 3.2 beschrieben. Darauf aufbauend werden hier die implementierten Lösungen für den externen Speicherzugriff durch den Mikrocontroller und den Zugriff mehrerer Module auf einen gemeinsamen RAM beschrieben.

Eine Möglichkeit zur Realisierung des gemeinsamen Speicherzugriffs durch mehrere Module ist die Modellierung der Ansteuerung durch Three-State-Treiber. Diese können neben einem High- und Low-Pegel einen hochohmigen Zustand annehmen, der mit "Z" bezeichnet wird [14, S.161]. Der hochohmige Zustand wird angenommen, wenn am zugehörigen Freigabeeingang ein Low-Pegel anliegt. Three-State-Ausgänge können in einer Busschaltung miteinander verbunden werden, wenn sichergestellt ist, dass zu jedem Zeitpunkt nur einer der Freigabeeingänge aktiv ist. Der aktuelle Pegel auf dem Bus wird durch den Dateneingang definiert, dessen Ausgang gerade freigeschaltet ist [14, S.162]. Die Ansteuerung durch Three-State-Treiber ist nur für die Eingänge des RAMs notwendig. An den Datenausgang des Speichers können direkt mehrere Module parallel angeschlossen werden.

Die Exklusivität der Freigabe wird durch die sequentielle Aktivierung der Signalverbeitung gewährleistet. Als Freigabesignal der RAM-Ansteuerung dient das Signal `MOD_EN`. Um zusätzlich externe RAM-Zugriffe zu ermöglichen, werden die Adress- und Dateneingangsleitungen zu allen Signalverarbeitungsmodulen mithilfe des Signals `CTRL_EXT_EN` hochohming geschaltet. Dieses ist mit dem `MOD_EN`-Signal UND-verküpft. Damit die Module während des externen Zugriffs nicht weiterhin versuchen auf den RAM zuzugreifen, was zu Informationsverlust führen würde, wird der Takt der Signalverarbeitungsmodule während des externen RAM-Zugriffs deaktiviert. Im Folgenden werden alle notwendigen Aktionen für einen externen Speicherzugriff in ihrer zeitlichen Abfolge aufgelistet:

1. Abschalten des Taktes der Signalverarbeitungsmodule. Diese verbleiben in ihrem aktuellen Zustand.
2. Hochohmigschalten der Verbindung zwischen dem RAM und dem aktiven Signalverarbeitungsmodul. Zeitgleich wird der Memory Control Block mit dem RAM verbunden.
3. Durchführen des Speicherzugriffs über den Memory Control Block.
4. Hochohmigschalten der Verbindung zwischen dem Memory Control Block und dem RAM. Wiederherstellung der Verbindung zwischen dem RAM und dem aktiven Signalverarbeitungsmodul.
5. Zuschalten des Taktes der Signalverarbeitungsmodule. Das aktive Modul setzt seine Verarbeitung fort.

Dieses Verfahren ermöglicht es, nach jedem Takt auf den gesamten Speicherinhalt zuzugreifen.

Nehmen die Read/Write Enable-Eingänge des VHDL-Modells des IP-Cores-Speicher innerhalb der Simulationsumgebung einen hochohmigen Wert an, werden die Daten im Speicher mit einem undefinierten Wert überschrieben. Um dies zu vermeiden, werden die Write/Read Enable-Signale aller auf den RAM zugreifenden Module mit einer ODER-Verknüpfung verschaltet. Die Freigabesignale `MOD_SEL` und `CTRL_EXT_EN` sorgen dafür, dass die genannten Eingänge bei Inaktivität des Moduls oder bei externem Speicherzugriff auf Low geschaltet werden. Eine schematische Darstellung der Realisierung des Speicherzugriffs durch mehrere Module ist in Abbildung 4.3 darstellt.

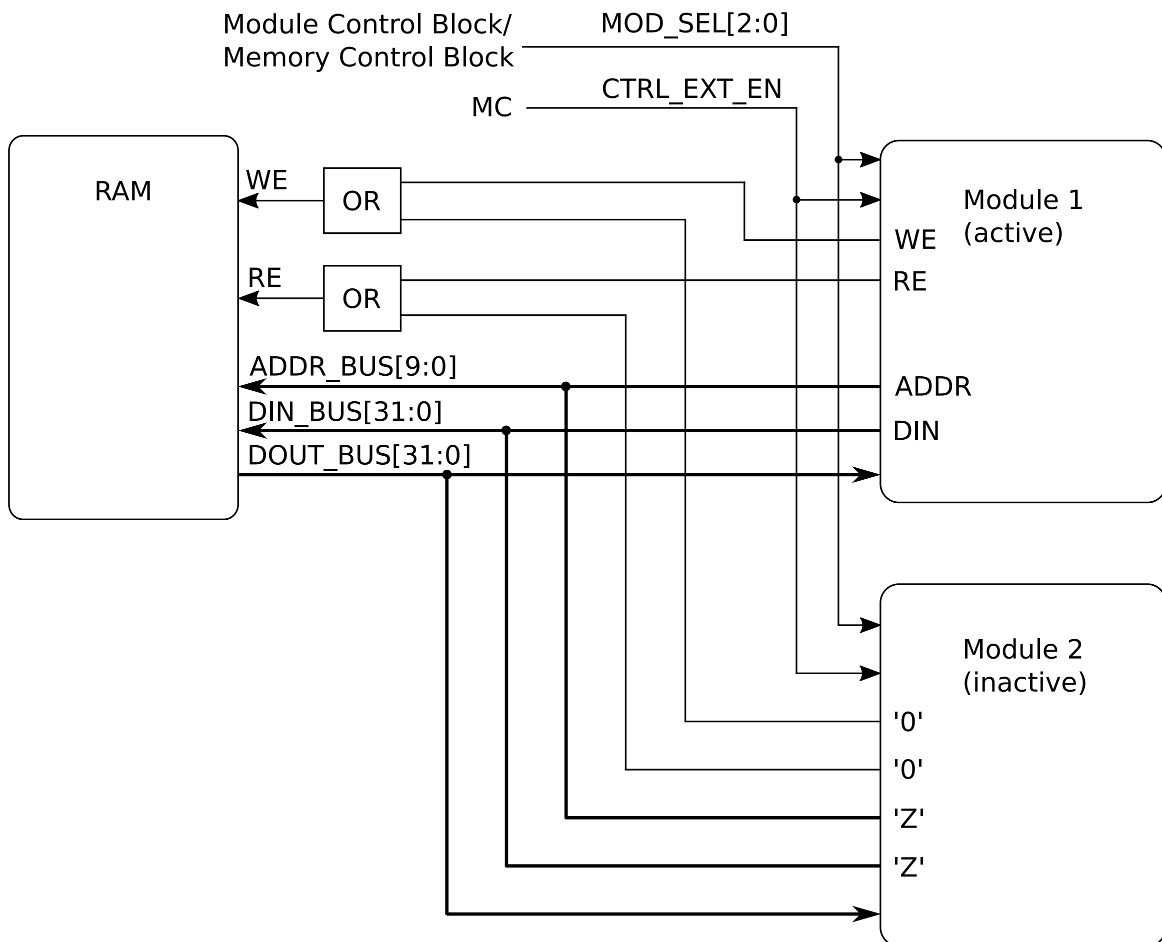


Abbildung 4.3: Schematische Übersicht des Anschlusses mehrerer Module an den RAM

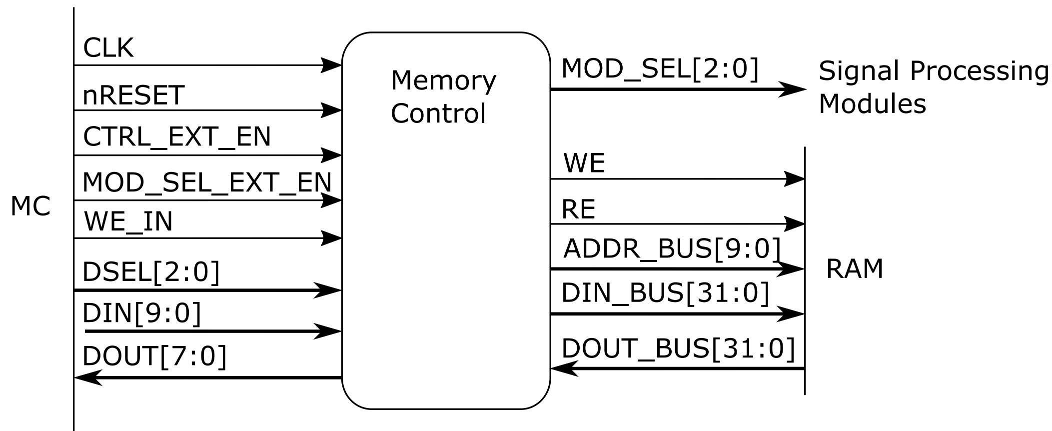


Abbildung 4.4: Blockschaltbild des Memory Control Blocks

Der Memory Control Block ermöglicht den externen Zugriff auf den RAM. Der vollständige VHDL-Code befindet sich in Listing B.3. Abbildung 4.4 zeigt das Blockschaltbild des Memory Control Blocks mit den definierten Ein- und Ausgängen. Durch ein Multiplexverfahren werden verschiedene Daten zur Ansteuerung des Speichers sequentiell über dieselben IO-Pins übertragen.

Die Auswertung der externen Ansteuerung durch den Mikrocontroller erfolgt innerhalb eines Prozesses in Form von getakteter Logik. Der in dem Memory Control Block modelierte Prozess ist in Abbildung 4.5 dargestellt. Die Synchronierung innerhalb des Prozesses erfolgt durch die negative Taktflanke. Auf diese Weise wird gewährleistet, dass der RAM wie in Abschnitt 3.2 beschrieben stehts zur negativen Taktflanke beschaltet wird.

Da der Memory Control Block parallel zu den Signalverarbeitungsmodulen an den Speicher geschaltet wird, werden die Adress- und Dateneingangsleitung ebenfalls über Three-State-Treiber realisiert. Die Write/Read Enable-Signale werden mit denen der Signalverarbeitungsmodule ODER-verknüpft. Nur wenn der Memory Control Block durch das Signal `CTRL_EXT_EN` an den RAM geschaltet ist, werden die Lese- und Schreibbefehle des Mikrocontrollers verarbeitet.

Dabei wird durch das `WE_IN`-Signal festgelegt, ob es sich über einen schreibenden oder lesenden Zugriff handelt. Abhänig davon wird der Wert des `DSEL`-Signals interpretiert.

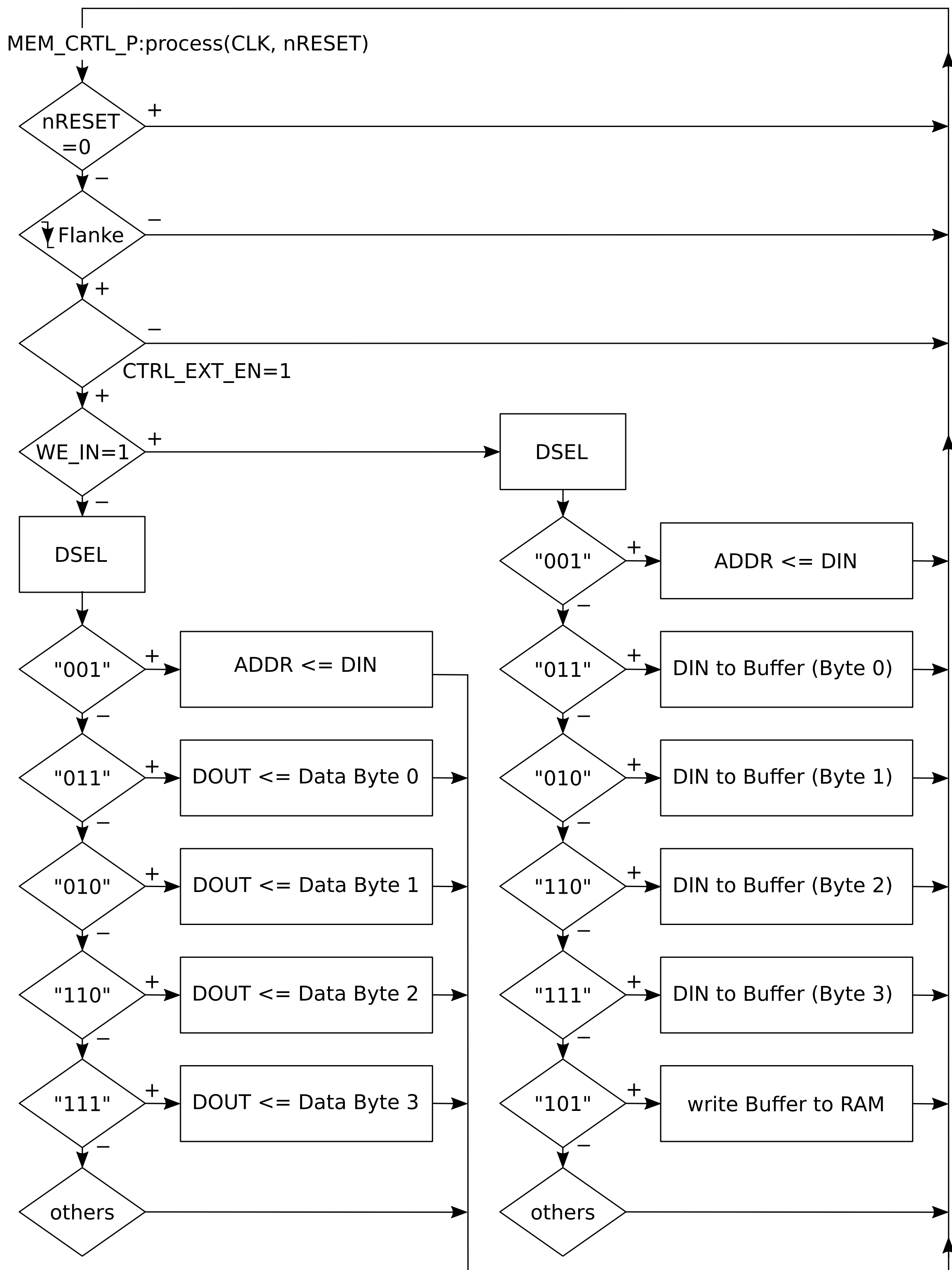


Abbildung 4.5: Programmablaufplan zur Beschreibung der Funktion innerhalb des Prozesses des Memory Control Blocks

Durch die nachfolgende Wertekombination des Signals DSEL lässt sich ein Lesezugriff auf den RAM realisieren:

- "001": Der am Eingang DIN anliegende Wert wird als Speicheradresse interpretiert und an den RAM anlegt.
- "011": Schreibt Bit 31-24 des adressierten RAM-Worts auf den Ausgang DOUT.
- "010": Schreibt Bit 23-16 des adressierten RAM-Worts auf den Ausgang DOUT.
- "110": Schreibt Bit 15-8 des adressierten RAM-Worts auf den Ausgang DOUT.
- "111": Schreibt Bit 7-0 des adressierten RAM-Worts auf den Ausgang DOUT.

Zu beachten ist, dass die Speicheradresse angelegt wird, bevor die einzelnen Bytes ausgelesen werden. Bei einem Schreibzugriff müssen die vom Mikrocontroller angelegten Daten zwischengespeichert werden, bevor alle vier Bytes gemeinsam in den RAM geschrieben werden, andernfalls würden die vorher geschrieben Bytes überschrieben werden.

Der Schreibzugriff wird über einen Schreibbuffer realisiert:

- "001": Der am Eingang DIN anliegende Wert wird als Speicheradresse interpretiert und an den RAM anlegt.
- "011": Schreibt Bit 7-0 am Eingang DIN als Bit 31-24 in den Schreib-Buffer.
- "010": Schreibt Bit 7-0 am Eingang DIN als Bit 23-16 in den Schreib-Buffer.
- "110": Schreibt Bit 7-0 am Eingang DIN als Bit 15-8 in den Schreib-Buffer.
- "111": Schreibt Bit 7-0 am Eingang DIN als Bit 7-0 in den Schreib-Buffer.
- "101": Der Werte im Schreib-Buffer wird an den Dateneingang des RAMs angelegt, um bei der nächsten positiven Taktflanke in den Speicher geschrieben zu werden.

Die einzelnen Befehle werden entsprechend ihrer vorgesehenen Ablaufreihenfolge im Grey-Code codiert. Dies minimiert bei der Ansteuerung die Pegel-Umschaltung im Signal DSEL.

Die Möglichkeit, die Signalverarbeitungsmodule extern zu aktiveren, wurde ebenfalls über den Memory Control Block realisiert. Durch Aktivierung des Signals MOD_SEL_EXT_EN wird das Eingangssignal DSEL mit dem Ausgangssignal MOD_SEL verknüpft. Dazu wird das der MOD_SEL Ausgang am Module Control Block per Three-State-Treiber hochohmig geschaltet. Somit lassen sich über das Signal DSEL Module extern aktivieren und deaktivieren.

4.3 Ansteuerung der Signalverarbeitungsmodule

Die Steuerung des Ablaufes der Signalverarbeitungsmodule erfolgt durch den Module Control Block. Die zugehörige VHDL-Beschreibung befindet sich in Listing B.4. Das Blockschaltbild des Module Control Blocks ist in Abbildung 4.6 dargestellt. Die Aktivierung der Module erfolgt sequentiell. Alle Signalverarbeitungsmodule sind an den Module Select Bus geschaltet. Der Module Control Block schaltet eine Nummer auf den Bus. Innerhalb der Module sind ihre Modulnummern hinterlegt. Das Modul, dessen Nummer auf den Bus geschaltet wird, wird aktiviert.

Die Ansteuerung erfolgt im Handshake-Verfahren. Hat das Modul seine Verarbeitung beendet, sendet es ein Ready-Signal zurück an den Module Control Block. Da bei dem Ready-Signal mehrere Signalverarbeitungsmodule den Module Control Block senden müssen, wird die Ansteuerung per Three-State-Treiber realisiert. Als Freigabesignal dient das Signal `MOD_SEL`.

Die Anwahl der Signalverarbeitungsmodule innerhalb des Module Control Blocks wird durch einen Zustandsautomaten realisiert, welcher in Abbildung 4.7 dargestellt ist. Jeder Zustand repräsentiert ein Modul. Der Automat besitzt Moore-Verhalten. Während des gesamten Zustands wird die Modulnummer auf den `MOD_SEL` Ausgang geschaltet. Das Ready-Signal dient für jeden Zustand als Transitionsbedingung. Ein Rücksetzen führt zum Zustand `M1`.

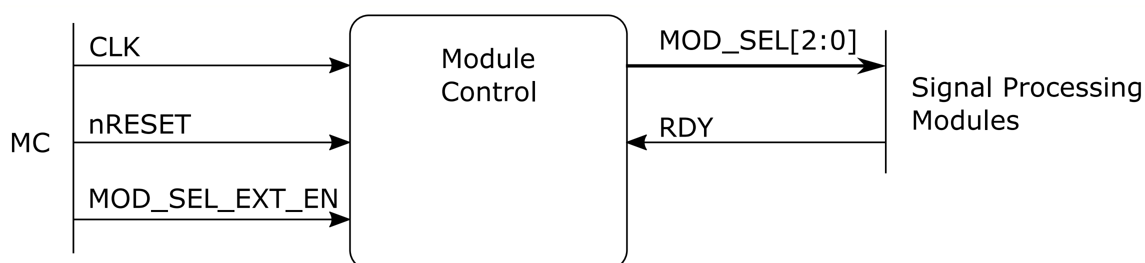


Abbildung 4.6: Blockschaltbild des Module Control Blocks

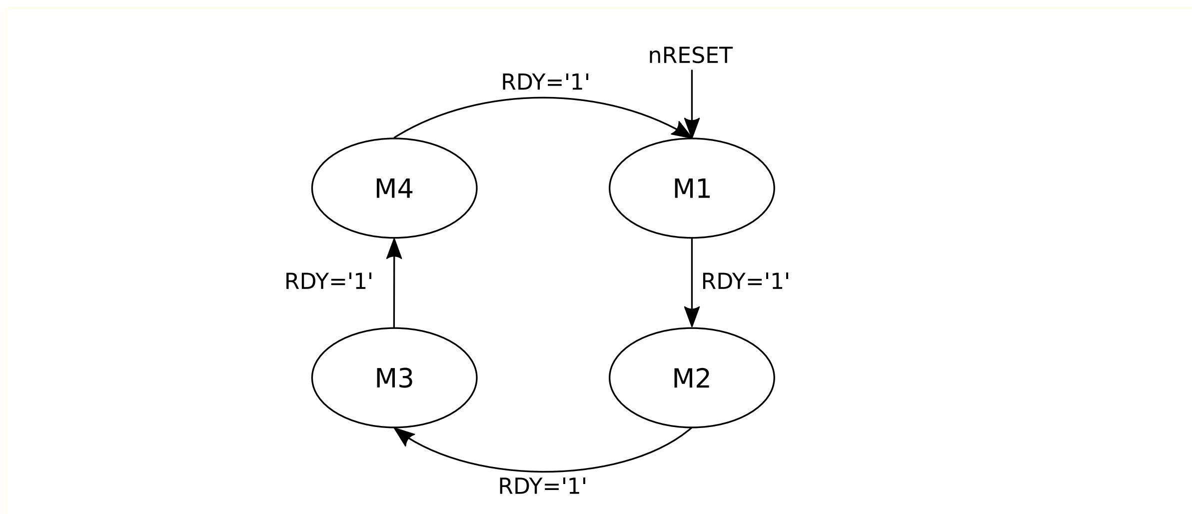


Abbildung 4.7: Zustandsautomat Module Control Block

4.4 Test-Modul zur Ansteuerung des RAMs

Zum Zeitpunkt der Implementierung des RAMs stehen noch keine Signalverarbeitungsmodule außer der Winkelberechnung zur Verfügung. Deshalb wird ein Test-Modul konzipiert. Das Test-Modul dient als Prototyp für die Umsetzung der Ansteuerung des RAMs und der Ansteuerung der Module selbst. Es verfügt über die gleichen Schnittstellen wie die Signalverarbeitungsmodule. Das Blockschaltbild des Test-Moduls ist in Abbildung 4.8 dargestellt. Der interne Aufbau für die Zugriffsfunktionen auf den RAM kann für die Signalverarbeitungsmodule übernommen werden. Die VHDL-Beschreibung des Test-Moduls befindet sich in Listing B.6.

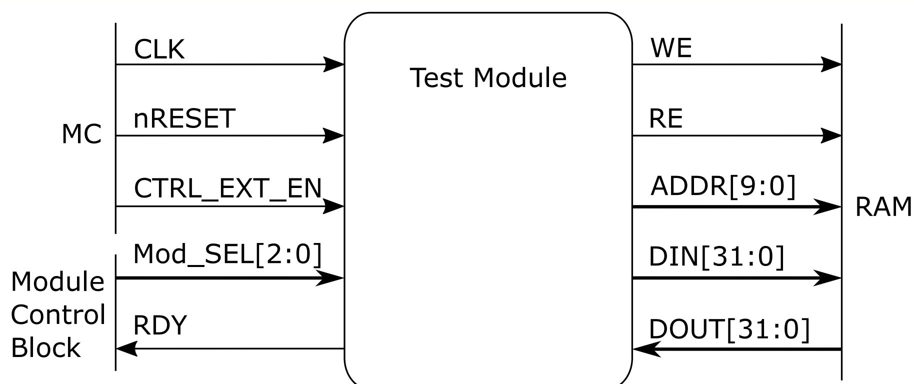


Abbildung 4.8: Blockschaltbild des Test-Moduls

Die Funktionen innerhalb des Moduls werden durch einen Zustandsautomat mit Moore-Verhalten realisiert. Die Implementierung des Automaten erfolgt innerhalb von drei Prozessen. Neben einem Schaltwerk zur Übernahme des Folgezustands und einem Schaltnetz zur Zustandsauswahl wird ein weiteres Schaltwerk realisiert. Dieses dient dem Zugriff auf den RAM. Das Schaltwerk wird auf die negative Taktflanke synchronisiert. Das bedeutet, dass die Speicheransteuerung stets um einen halben Takt zur Zustandsübernahme verzögert ist (bei symmetrischem Takt). Dies ermöglicht die Beschaltung des Speichermoduls entsprechend Abschnitt 3.2. Der Zustandsautomat ist in Abbildung 4.9 dargestellt.

Die Funktion der einzelnen Zustände wird im Folgenden beschrieben:

- `INIT`: Initialisierungszustand. Warten auf Aktivierung des Moduls.
- `RD0`: Anlegen der ersten Lese-Adresse und des Read Enable-Signals an den RAM.
- `RD1`: Einlesen des jeweils letzten RAM-Datenausgangs, Anlegen der nächsten Lese-Adresse. Zustand wird ausgeführt bis End-Adresse erreicht.
- `RD2`: Einlesen des letzen RAM-Datenausgangs. Das Read Enable-Signal wird auf Low gesetzt.
- `WT0`: Warteschritt zur Simulation der Berechnungszeit der Signalverarbeitungsmodule.
- `WR0`: Anlegen der ersten Schreib-Adresse sowie der ersten Modul-Ausgangsdaten an den RAM. Das Write Enable-Signal wird auf High gesetzt.
- `WR1`: Anlegen der nächsten Schreib-Adresse sowie der nächsten Modul-Ausgangsdaten an den RAM. Zustand wird ausgeführt bis End-Adresse erreicht.
- `WR2`: Abschalten des Write Enable-Signals.
- `WT1`: Warten auf Deaktivierung des Moduls.

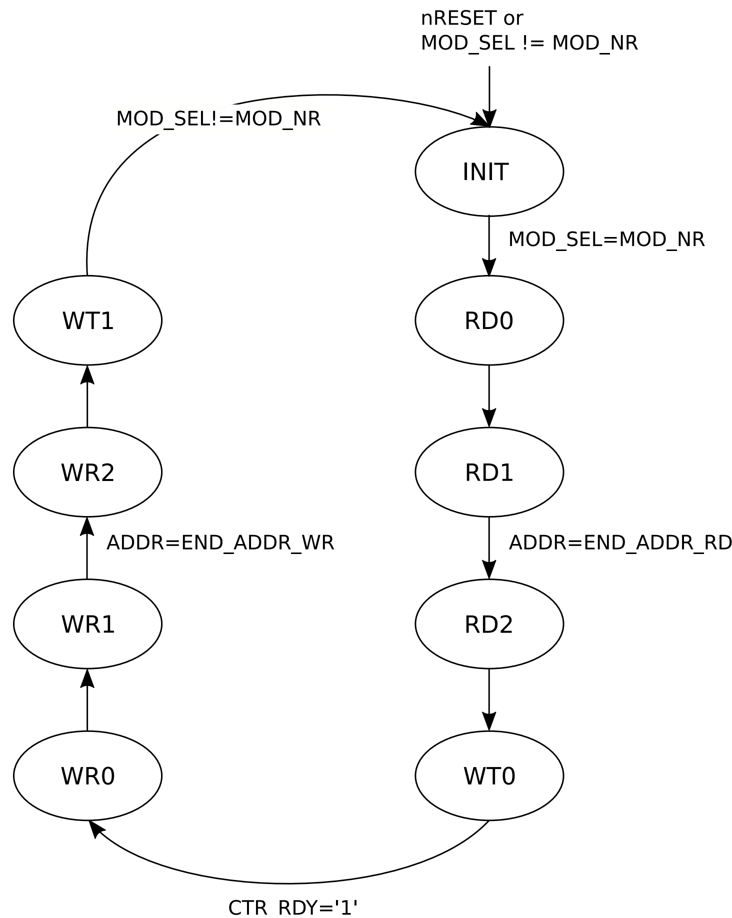


Abbildung 4.9: Zustandsautomat zur Realisierung der Funktion des Test-Moduls

Das Test-Modul ermöglicht es eine festgelegte Anzahl an Wörtern aus dem RAM auszulesen und an anderer Stelle im RAM wieder zu speichern. Zur Simulation des Signalverarbeitungsprozesses lässt sich eine definierbare Zahl an Wartetakten zwischen dem Lese- und den Schreibvorgang einstellen.

Die Vorgehensweise, vor der Bearbeitung alle Werte aus dem RAM auszulesen ist aufgrund des hohen Speicherbedarfs im Modul nicht sinnvoll. Bei der Implementierung der Signalverarbeitungsmodule sollte daher ein Ansatz zur Parallelisierung der Beschaffung, Verarbeitung und Speicherung der Werte verfolgt werden.

Ein Signal-Zeit-Diagramm mit den Ein- und Ausgangssignalen des Test-Moduls ist in Abbildung A.1 im Anhang dargestellt. Die Simulation wurde mit dem Cadence-Tool NC-Sim erstellt und zeigt das Einlesen einer Sequenz von RAM-Wörtern, eine kurze simulierte Verarbeitungszeit und das Schreiben der eingelesenen Werte an anderer Stelle in den RAM.

5 Entwurf und Umsetzung der Winkelberechnung

In diesem Kapitel wird das Winkelberechnungsmodul konzeptioniert und implementiert. Es werden verschiedene Verfahren zur Winkelberechnung verglichen. Darauf aufbauend werden Designentscheidungen für die Winkelberechnung getroffen. Weiterhin werden Betrachungen zur numerischen Genauigkeit des zu implementierenden Algorithmus vorgenommen. Abschließend wird die Implementierung in VHDL beschrieben.

5.1 Verfahren zur Winkelberechnung

Es existieren verschiedenene Methoden zur Berechnung elementarer Funktionen auf digitaler Hardware. Neben dem CORDIC-Algorithmus können zur Winkelberechnung etwa Taylor-Polynome oder Lookup-Tabellen verwendet werden.

Bei der Berechnung mittels Taylor-Polynom müssen bei zunehmender Genauigkeit immer mehr parallele Multiplikationen durchgeführt werden, was zu einen zunehmenden Bedarf an Hardwareressourcen führt, da Multiplikationen in digitaler Hardware aufwändig zu realisieren sind. Das spricht gegen die Verwendung dieses Verfahrens [7].

Eine Realisierung als Lookup-Tabelle ist aufgrund des hohen Speicherbedarfs ebenfalls nicht geeignet, insbesondere bei einer hohen Auflösung wie dem gewählten $s2Q9$-Format [10].

Der CORDIC-Algorithmus hat diese Nachteile nicht. Er kommt ohne Multiplikationen aus und der Speicherbedarf der zu hinterlegenden Koeffizienten ist ebenfalls gering [10]. Der Algorithmus ist effizient und lässt sich gut skalieren [7]. Aufgrund der beschrieben Vorteile wird der CORDIC-Algorithmus für die Implementierung des Winkelberechnungsmoduls gewählt.

5.2 Konzeption

Bei der Implementierung des CORDIC-Algorithmus sind unterschiedliche Varianten denkbar. Die einzelnen Iterationsschritte können sequentiell berechnet werden oder parallel in einer Pipeline-Struktur.

Aufgrund der Tatsache, dass alle Signalverarbeitungsmodule sequentiell angesteuert werden, ist eine Parallelisierung der Winkelberechnung nicht sinnvoll. Für eine serielle Verarbeitung empfielt sich in VHDL eine Implementierung als Zustandsautomat.

Vor der Realisierung des Winkelberechnungsmoduls in VHDL ist ein Test des Algorithmus in Matlab sinnvoll, da Änderungen im Algorithmus hier ohne großen Aufwand zu realisieren sind. Es bietet sich an, eine Funktion zu entwerfen, sodass diese direkt mit der vorhandenen `atan2`-Funktion von Matlab verglichen werden kann. Die implementierte Testfunktion `atan2_cordic` befindet sich in Listing B.15. Die grundlegende Struktur wurde aus [17] übernommen.

5.3 Praktische Implementierung der Winkelberechnung

Zusätzlich zu dem in Abschnitt 1.4 beschriebenen Algorithmus müssen für die praktische Implementierung noch einige Fallunterscheidungen vorgenommen werden [17]. Die Ablaufreihenfolge der Fallunterscheidungen wird in Abbildung 5.1 dargestellt. Die einzelnen Schritte sollen werden im Folgenden beschrieben:

1. Bestimmung des Quadranten der Eingangswerte.
2. Bildung des Absolutbetrags der Eingangswerte.
3. Wenn der Y-Wert größer als der X-Wert ist, werden die beiden Eingangswerte getauscht.
4. Wenn einer der beiden Eingangswerte Null ist, so wird die iterative CORDIC-Berechnung nicht durchlaufen, stattdessen wird das Ergebnis per Definition gebildet.
5. Korrektur des Ergebnisses, wenn die Eingangswerte vor der Berechnung getauscht wurden.
6. Wenn sich das Ergebnis außerhalb des ersten Quadranten befindet, erfolgt eine Korrektur für den jeweiligen Quadranten.

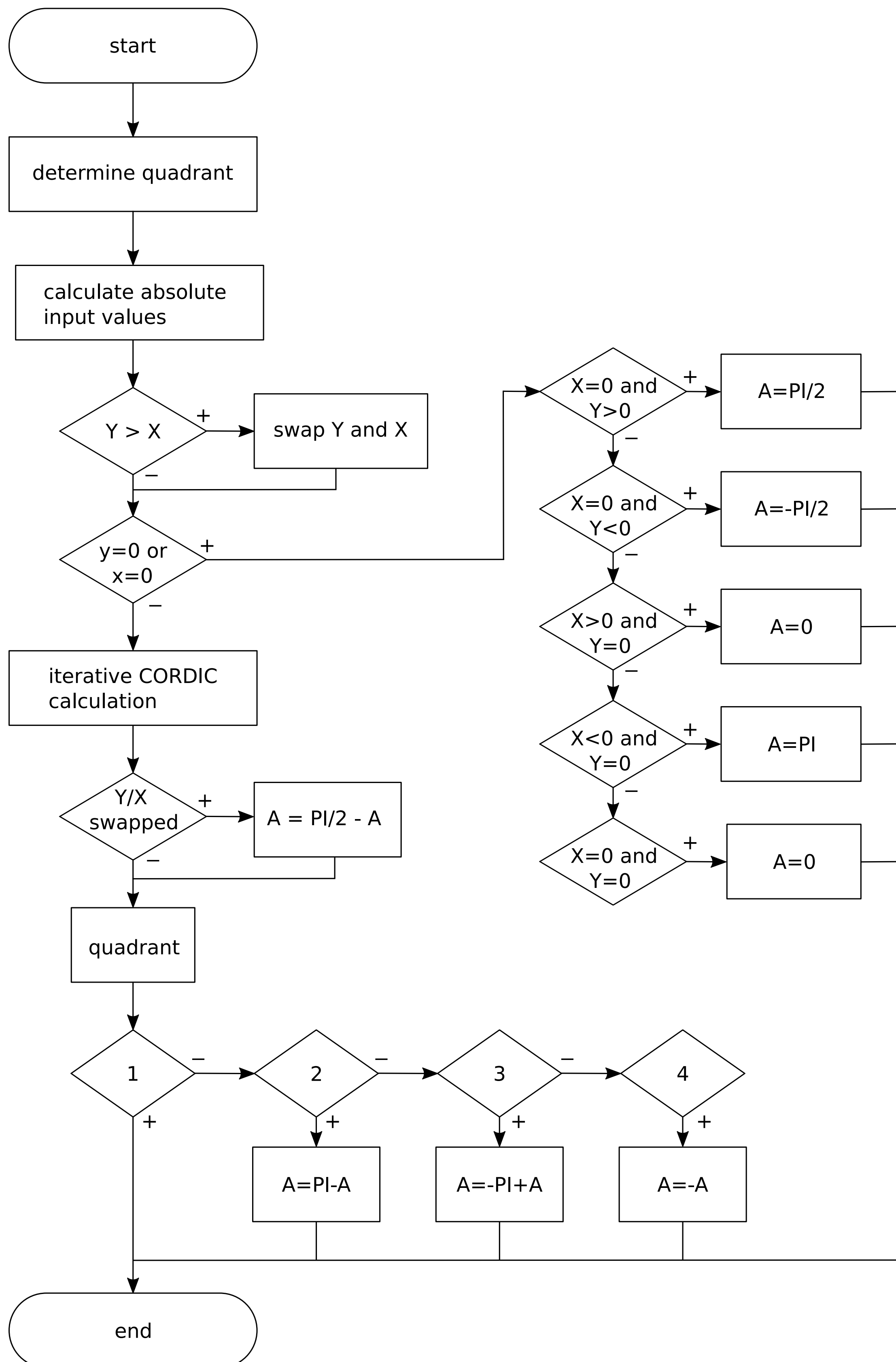


Abbildung 5.1: Fallunterscheidungen für die praktische Implementierung des CORDIC-Algorithmus zur Winkelberechnung

5.4 Betrachtung der numerischen Genauigkeit

Mithilfe der in Matlab implementierten Funktion `atan2_cordic` (B.15) wird die Genauigkeit des implementierten Algorithmus untersucht. Als Referenz dient die Matlab-`atan2`-Funktion. Damit die Aussagen über die Genauigkeit auch auf die VHDL-Implementierung übertragen werden können, wurde die Berechnung innerhalb der implementierten Funktion in Festkommaarithmetik durchgeführt. Das Zahlenformat wird als $s2Q9$-Format festgelegt. Für die sequentielle Berechnung der CORDIC-Iterationsschritte wird eine for-Schleife verwendet. Die im vorherigen Abschnitt beschriebenen Fallunterscheidungen werden ebenfalls vorgenommen.

In Tabelle 5.1 wird die Übereinstimmung der `atan2_cordic`-Funktion mit der Matlab-`atan2`-Funktion für verschiedene Zahlenformate und CORDIC-Iterationszahlen untersucht. Um eine Aussage über die Übereinstimmung der beiden Funktionen liefern zu können, werden die Winkelergebnisse für die Eingangswerte Y und X im Intervall von -4 bis 4 berechnet. Dies entspricht dem darstellbaren Zahlenbereich des $s2Q9$-Formats. Um den Berechnungsaufwand zu begrenzen, wurde eine Schrittweite von 0,1 gewählt. Die beiden Ergebnismatrizen werden voneinander subtrahiert. In Tabelle 5.1 sind die gemittelten Absolutbeträge der Differenzmatrix zum jeweiligen Zahlenformat bzw. zur Anzahl der Iterationen des CORDIC-Algorithmus dargestellt.

Das Ergebnis der Winkelberechnung ist genauer, wenn die Koeffizienten des CORDIC-Algorithmus aufgerundet werden. Um nicht jeden Koeffizienten für jedes Zahlenformat explizit berechnen zu müssen, wurde für die Berechnung der Rundungsmodus "nearest" verwendet.

Tabelle 5.1: Gemittelter Betrag der Differenz der implementierten `atan2_cordic`-Funktion und der Matlab-`atan2`-Funktion für verschiedene Zahlenformate und Iterationszahlen. Die Abweichungen sind in Radiant angegeben.

		Iterationen						
		6	7	8	9	10	11	12
Nachkommastellen	12	0,0083	0,0049	0,0031	0,0022	0,0019	0,0018	0,0018
	11	0,0084	0,0049	0,0032	0,0023	0,0021	0,0020	0,0020
	10	0,0083	0,0050	0,0032	0,0025	0,0023	0,0023	0,0023
	9	0,0084	0,0052	0,0035	0,0029	0,0029	0,0029	0,0029
	8	0,0090	0,0060	0,0048	0,0048	0,0048	0,0048	0,0048
	7	0,0093	0,0067	0,0067	0,0067	0,0067	0,0067	0,0067
	6	0,0138	0,0138	0,0138	0,0138	0,0138	0,0138	0,0138

Wie in Tabelle 5.1 deutlich zu erkennen ist, führt eine Anzahl an Iterationen höher als die Anzahl der Nachkommastellen der jeweiligen Zahlenformate zu keinem besseren Ergebnis. Dies ist darin begründet, dass die Winkelkoeffizienten für die Iterationschritte nicht mehr darstellbar sind. Für das verwendete Zahlenformat $s2Q9$ lässt sich das optimale Winkelergebnis also mit 9 Iterationsschritten erreichen.

Für die folgenden Berechnungen wird der Rundungsmodus auf "floor" gestellt, alle Berechnungsergebnisse werden somit abgerundet. Dies entspricht dem Verhalten der in Abschnitt 5.5 in VHDL verwendeten Rechenoperationen. Die Koeffizienten des Algorithmus werden für das $s2Q9$-Format explizit berechnet.

Abbildung 5.2 zeigt das mit der `atan2_cordic`-Funktion berechnete Winkelergebnis im $s2Q9$-Format mit 9 Iterationsschritten. Dargestellt ist das Ergebnis der Berechnung im Intervall der beiden Eingangswerte von -4 bis 4. Die Schrittweite beträgt 0,1.

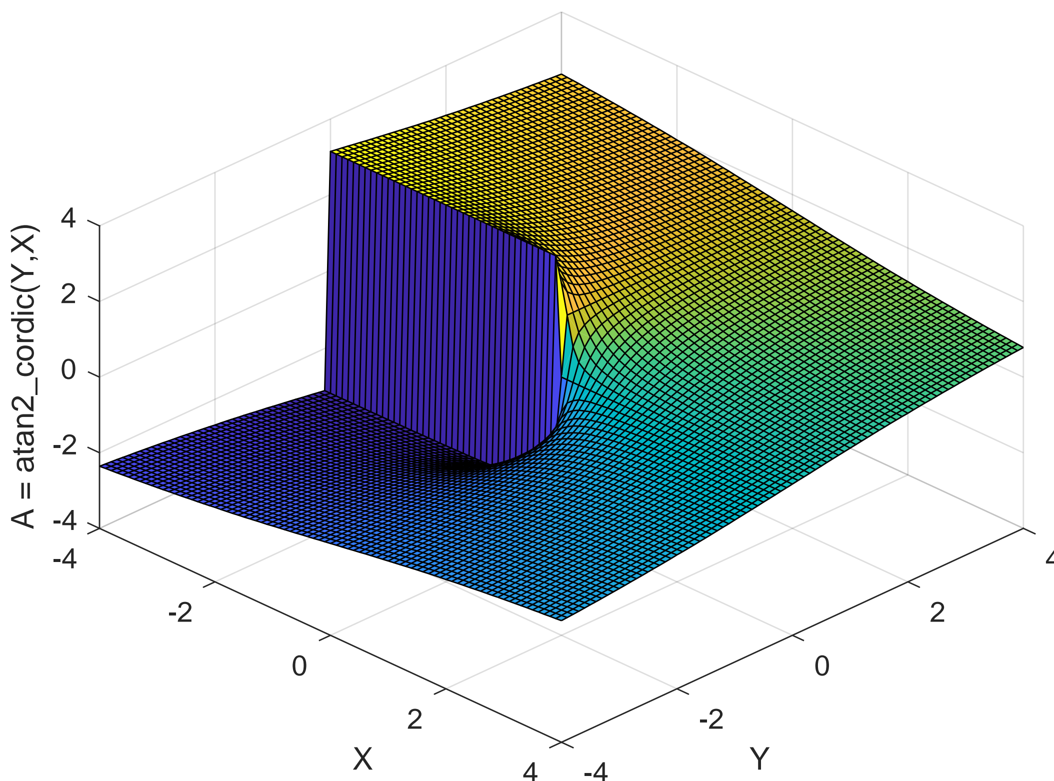


Abbildung 5.2: Berechnetes Winkelergebnis der in Matlab implementierten `atan2_cordic`-Funktion. Die Einheit des berechneten Winkels ist Radiant.

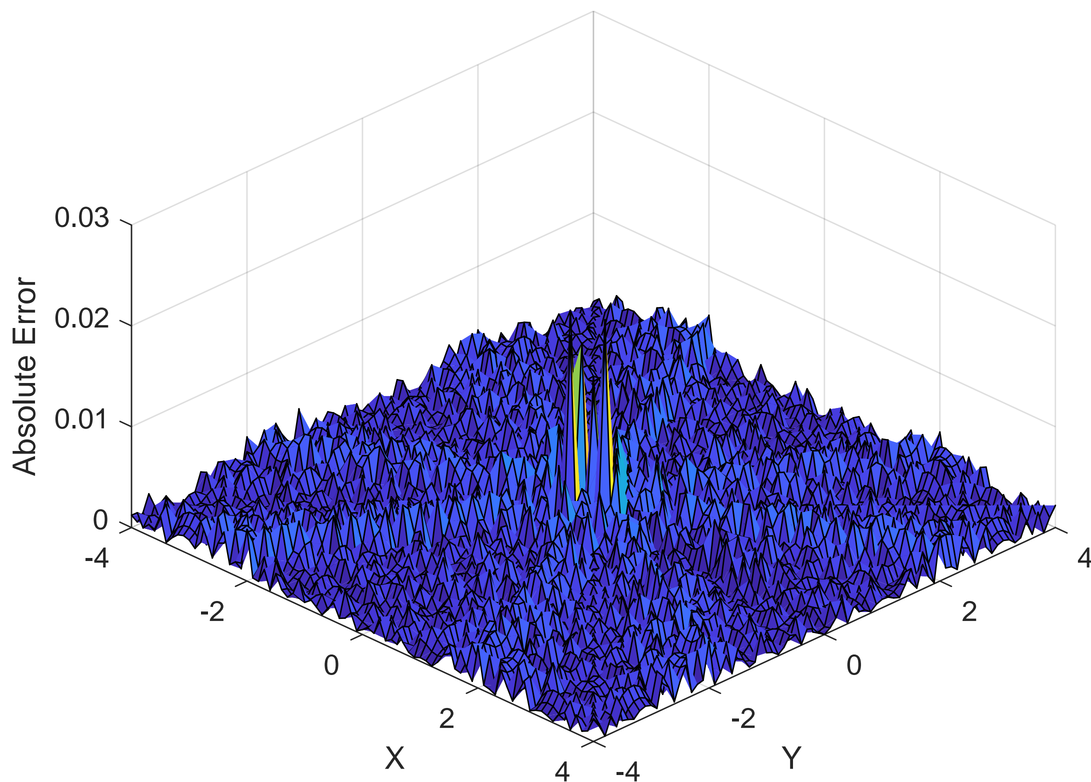


Abbildung 5.3: Betragsmäßiger absoluter Fehler der in Matlab implementierten `atan2_cordic`-Funktion für den im $s2Q9$-Format darstellbaren Bereich der Eingangswerte. Die Einheit des berechneten Fehler ist Radiant.

In Abbildung 5.3 ist der betragsmäßige absolute Fehler der `atan2_cordic`-Funktion im Intervall von -4 bis 4 der beiden Eingangswerte dargestellt. Als Schrittweite wurde der Wert 0,1 gewählt. Zur Berechnung wurde die Ergebnismatrix der `atan2_cordic`-Funktion von der Matlab-atan2-Funktion subtrahiert. Dargestellt ist der Absolutbetrag der Differenzmatrix.

Es ist eindeutig zu erkennen, dass die größte Abweichung in dem Bereich um den Nullpunkt der beiden Eingangswerte vorliegt.

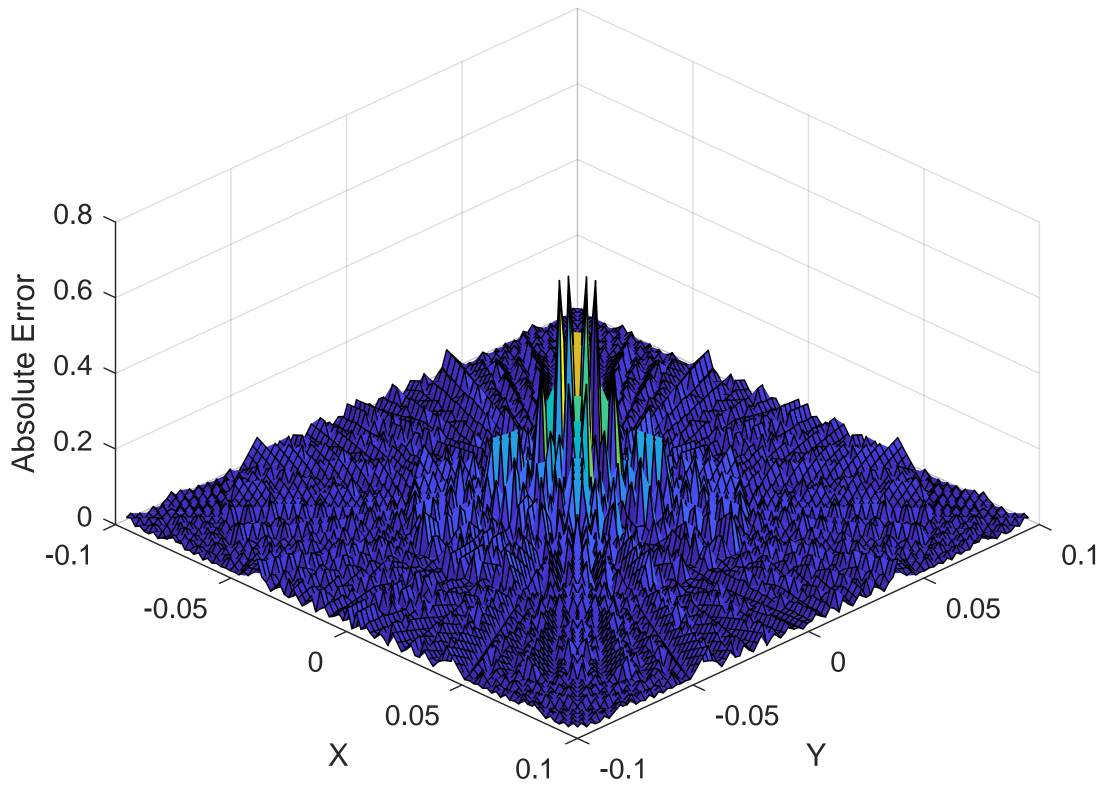


Abbildung 5.4: Betragsmäßiger absoluter Fehler der in Matlab implementierten `atan2_cordic`-Funktion im Bereich um den Nullpunkt der Eingangswerte. Die Einheit des berechneten Fehler ist Radiant.

Um eine genauere Aussage über die maximale Abweichung treffen zu können, wurde der betragsmäßige absolute Fehler im Intervall von $-0,1$ bis $0,1$ der beiden Eingangswerte untersucht. Als Schrittweite wurde diesmal die gerinste darstellbare Auflösung des $s2Q9$-Formats gewählt, d.h. $1,953125 \cdot 10^{-3}$. Abbildung 5.4 zeigt das Ergebnis der Berechnung. Um den Nullpunkt der Eingangswerte ergibt sich ein maximaler Winkelfehler von $0,6343$ *rad*. Dies entspricht im Gradmaß einem Winkel von $36,3427°$.

5.5 VHDL-Umsetzung

Der VHDL-Code des implementierten Moduls zur Winkelberchnung befindet sich in Listing B.5. Die Schnittstellen des Winkelberechnungsmoduls werden entsprechend dem in Abschnitt 4.4 entwickelten Testmodul definiert. Das Blockschaltbild des Winkelberechnungsmoduls ist in Abbildung 5.5 dargestellt. Innerhalb des Moduls wird die Verarbeitung durch einen Zustandsautomaten realisiert (Abbildung 5.6). Dieser ist neben der Ausführung des Winkelberechnungsalgorithmus auch für den Zugriff auf dem RAM zuständig. Daher erfolgt wie beim Testmodul eine Realisierung des Zustandsautomaten mit drei Prozessen. Die Übernahme des Folgezustands wird in einem synchronen Prozess ausgeführt (positive Taktflanke). Die Zustandsauswahl und die Winkelberechnung werden in einem weiteren Prozess ausgeführt. Um zu gewährleisten, dass jeder Iterationsschritt des CORDIC-Algorithmus nur einmal ausgeführt wird, wird der Prozess getaktet. Damit Zustandsauswahl und -Übernahme nicht zum gleichen Zeitpunkt stattfinden, wird der Prozess auf die negative Taktflanke synchronisiert.

Der Speicherzugriff erfolgt ebenfalls wie bei dem Test-Modul durch ein auf die negative Taktflanke synchronisiertes Ausgangsschaltwerk, die Möglichkeiten der Parametrisierung wurden jedoch auf eine Leseadresse und eine Schreibadresse reduziert, da beide Eingangswerte aus einem RAM-Wort gelesen werden können und nur ein Winkelergebnis pro Modulaktivierung in den RAM geschrieben wird.

Die für die Winkelberechnung benötigten Winkelkoeffizienten werden vorab berechnet und im Array `ANGLES` abgelegt (Zeile 79-83). Um ein besseres Ergebnis zu erziehen, werden die Koeffizienten innerhalb des $s2Q9$-Formats aufgerundet. Die in den einzelnen Schritten des CORDIC-Algorithmus vorgenommenen Bitshift-Operation wurden mithilfe der `Resize`-Funktion realisiert. Diese gewährleistet für negative Werte, dass die Bitshift-Operation vorzeichengerecht ausgeführt wird. Der `Resize`-Operator wird ebenfalls verwendet, um die Eingangswerte aus dem eingelesenen Speicherwort vor der Berechnung vom $s1Q10$ ins $s2Q9$-Format zu übersetzen (Zeile 137-138).

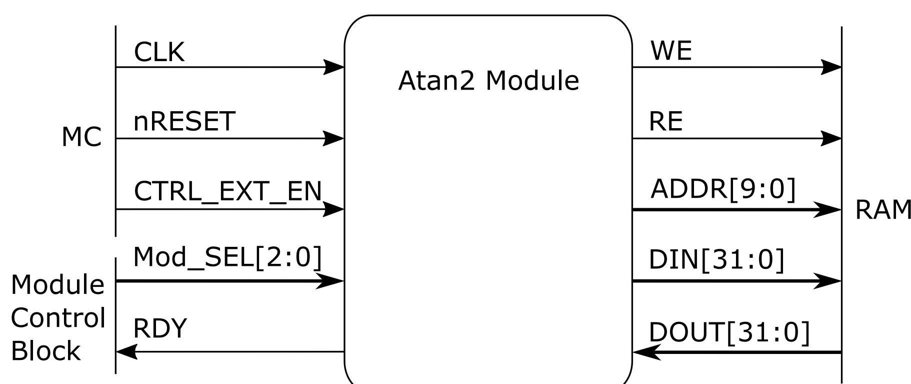


Abbildung 5.5: Blockschaltbild des Winkelberechungsmoduls

In der realisierten Form des CORDIC-Algorithmus wird ein Iterationsschritt pro Takt berechnet. Durch Verwendung von Variablen wäre es auch möglich, mehrere Iterationsschritte innerhalb eines Taktes durchzuführen. Variablen in VHDL können nach einer Wertzuweisung sofort abgefragt werden, Signale erst am Ende eines Prozesses [14, S.128]. Eine Parallelisierung der Berechnung führt jedoch zu mehr Hardwareaufwand. Letztendlich muss ein Kompromiss zwischen Hardwareaufwand und Berechnungszeit gefunden werden. Die hier beschriebene Implementation des CORDIC-Algorithmus benötigt für ihre Berechnung 11 Takte (ohne Lese- und Schreibvorgang). Ist einer der Eingangswerte gleich Null wird das Ergebnis per Definition innerhalb eines Taktes gebildet.

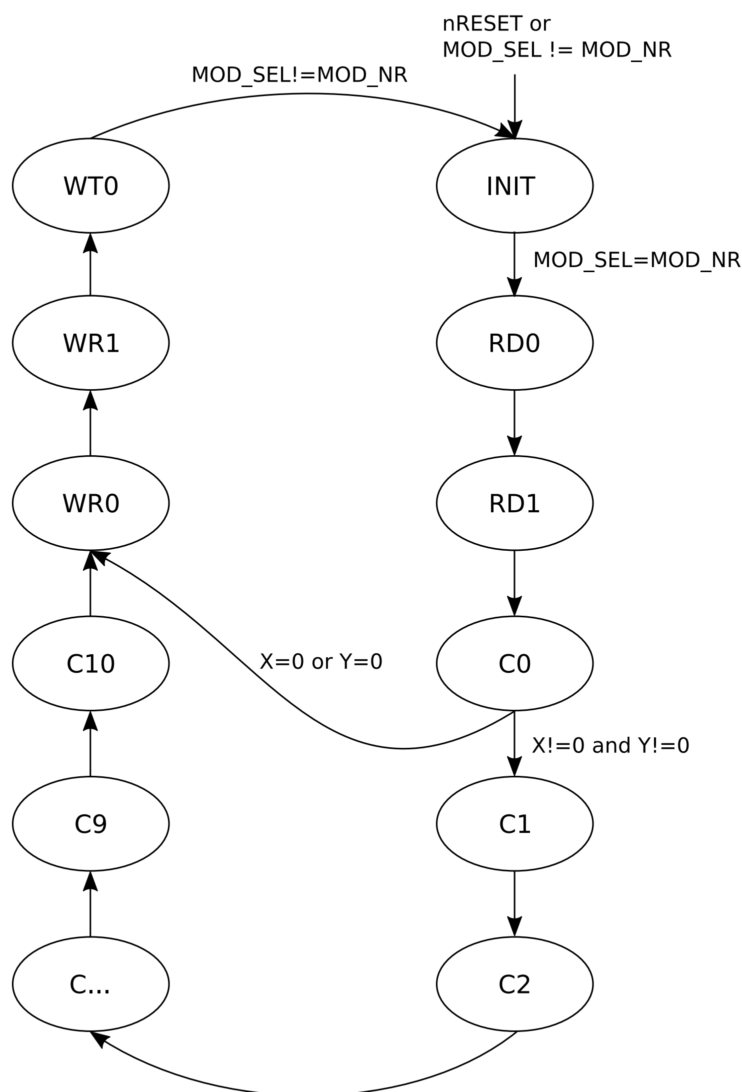


Abbildung 5.6: Zustandsautomat des in VHDL implementierten CORDIC-Algorithmus zur Winkelberechnung

Die Funktion der einzelnen Zustände des realisierten Zustandsautomaten wird im Folgenden beschrieben:

- `INIT`: Initialisierungszustand. Warten auf Aktivierung des Moduls.
- `RD0`: Anlegen der Lese-Adresse sowie des Read Enable-Signals an den RAM.
- `RD1`: Einlesen des RAM-Datenausgangs. Das Read Enable-Signal wird auf Low gesetzt.
- `C0`: Bestimmung des Quadranten der Eingangswerte. Zur weiteren Berechnung wird der Absolutbetrag der Eingangswerte gebildet. Wenn $Y > X$ ist, werden die Eingangswerte getauscht. Hat einer der Eingangswerte den Wert Null, wird die iterative Berechnung übersprungen.
- `C1 - C9`: Iterative Berechnung mit CORDIC-Algorithmus.
- `C10`: Wenn der berechete Winkel außerhalb des ersten Quadranten liegt oder die Eingangswerte getauscht wurden, wird das Ergebnis mit einem Korrekturfaktor angepasst.
- `WR0`: Anlegen der Schreib-Adresse und der Ausgangsdaten (berechneter Winkel) an den RAM. Zuschalten des Write Enable-Signals.
- `WR1`: Abschalten des Write Enable-Signals.
- `WT0`: Warten auf Deaktivierung des Moduls.

Ein Signal-Zeit-Diagramm mit den Ein- und Ausgangssignalen des Winkelberechnungsmoduls ist in Abbildung A.2 im Anhang dargestellt. Die Simulation wurde mit dem Cadence-Tool NC-Sim erstellt und zeigt das Einlesen der Eingangswerte aus einem RAM-Wort, die Verarbeitung und das Schreiben des berechneten Winkels in den RAM. Für beide Eingangswerte wurde der Wert 1 gewählt (s1Q10), als Winkel wurden $0,7822\ rad$ $(44,82°)$ berechnet (s2Q9).

6 Funktionstest und Aufwandsabschätzung

In diesem Kapitel wird die Mikrocontroller-Software für den RAM-Zugriff realisiert. Um eine Eingabe- und Auswertemöglichkeit zu schaffen, werden weiterhin Skripte und Funktionen für GNU Octave entwickelt. Außerdem sollen mithilfe der Cadence-Umgebung die in VHDL entwickelten Module zum Zweck einer ersten Aufwandsabschätzung synthetisiert werden.

6.1 Entwicklung der Mikrocontroller-Steuerung für das RAM-Modul

Als Plattform dient das Entwicklungsboard EK-TM4C1294XL [13]. Für dieses stellt der Hersteller Texas Instruments mit der Tivaware SW-TM4C [18] eine umfassende Bibliothek zur Verfügung. Die Tivaware wird in dieser Arbeit verwendet, da sie Funktionen zur Ansteuerung der benötigten Peripheriekomponenten (GPIOs, Timer, UART) bereitstellt.

Für eine bessere Übersicht wird die Software in mehrere Dateien aufgeteilt. Die Strukturierung erfolgt nach den verwendeten Peripheriekomponenten. Abbildung 6.1 zeigt die Aufteilung in Header- und Sourcedateien. Die aufgelisteten Dateien befinden sich in den Listings B.7 bis B.14.

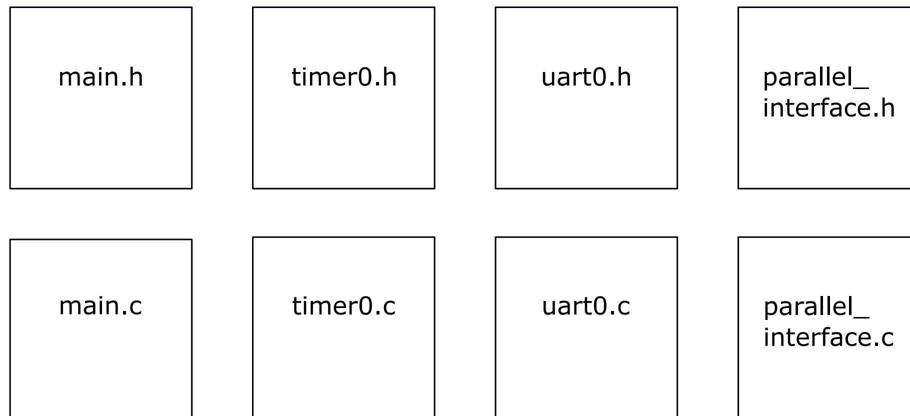


Abbildung 6.1: Strukturierung der Software auf dem EK-TM4C1294XL als Header- und Sourcedatein

In der Software soll neben dem RAM-Zugriff auch die Kommunikation über eine UART-Schnittstelle realisiert werden. Dies erfordert die parallele Verarbeitung von zwei Tasks. Daher werden für die Entwicklung des Steuerungsprogramms Interrupts verwendet.

Die festgelegten Interrupt-Prioritäten sind in Tabelle 6.1 dargestellt. Dem UART-Interrupt wird eine höhere Priorität zugewiesen als dem Timer-Interrupt. Diese Entscheidung ist darin begründet, dass für die Ansteuerung des FPGAs in dieser Testumgebung keine Echtzeit-Beschränkungen eingehalten werden müssen. Eine verlustlose Datenübertragung zwischen Mikrocontroller und PC ist jedoch notwendig für einen reibungslosen Testablauf.

Tabelle 6.1: Festlegung der Interrupt-Priotitäten

Interrupt-Handler	**Interrupt-Quelle**	**Priorität**
TIMER0_handler	Timer 0	0x20 medium priority
UART_handler	UART 0	0x00 high priority

Die Parameter zur Ansteuerung werden zwischen den ISRs mithilfe von globalen Variablen übergeben. Im Folgenden wird die Funktion der implementierten globalen Variablen beschrieben:

- `nreset`, `mod_clk_en`, `mod_sel_ext_en`, `dsel`: Diese Variablen dienen der direkten Ansteuerung des FPGAs. Sie werden im UART-Handler gesetzt und im Timer-Handler auf die zugehörigen GPIO geschrieben.
- `next_clk`: Dient der Ansteuerung der Signalverarbeitungsmodule mit genau einem Takt. Die Variable wird im UART-Handler gesetzt und im Timer-Handler zurückgesetzt.
- `read_ram`: Startet den Lesevorgang des RAMs. Diese Variable wird im UART-Handler gesetzt und im Timer-Handler zurückgesetzt.
- `write_ram`: Startet den Schreibvorgang des RAMs. Diese Variable wird im UART-Handler gesetzt und im Timer-Handler zurückgesetzt.
- `ram_input_data`: Dient der Speicherung der in den RAM zu speichernden Daten im Mikrocontroller.
- `ram_output_data`: Dient der Speicherung der aus dem RAM gelesenen Daten im Mikrocontroller.

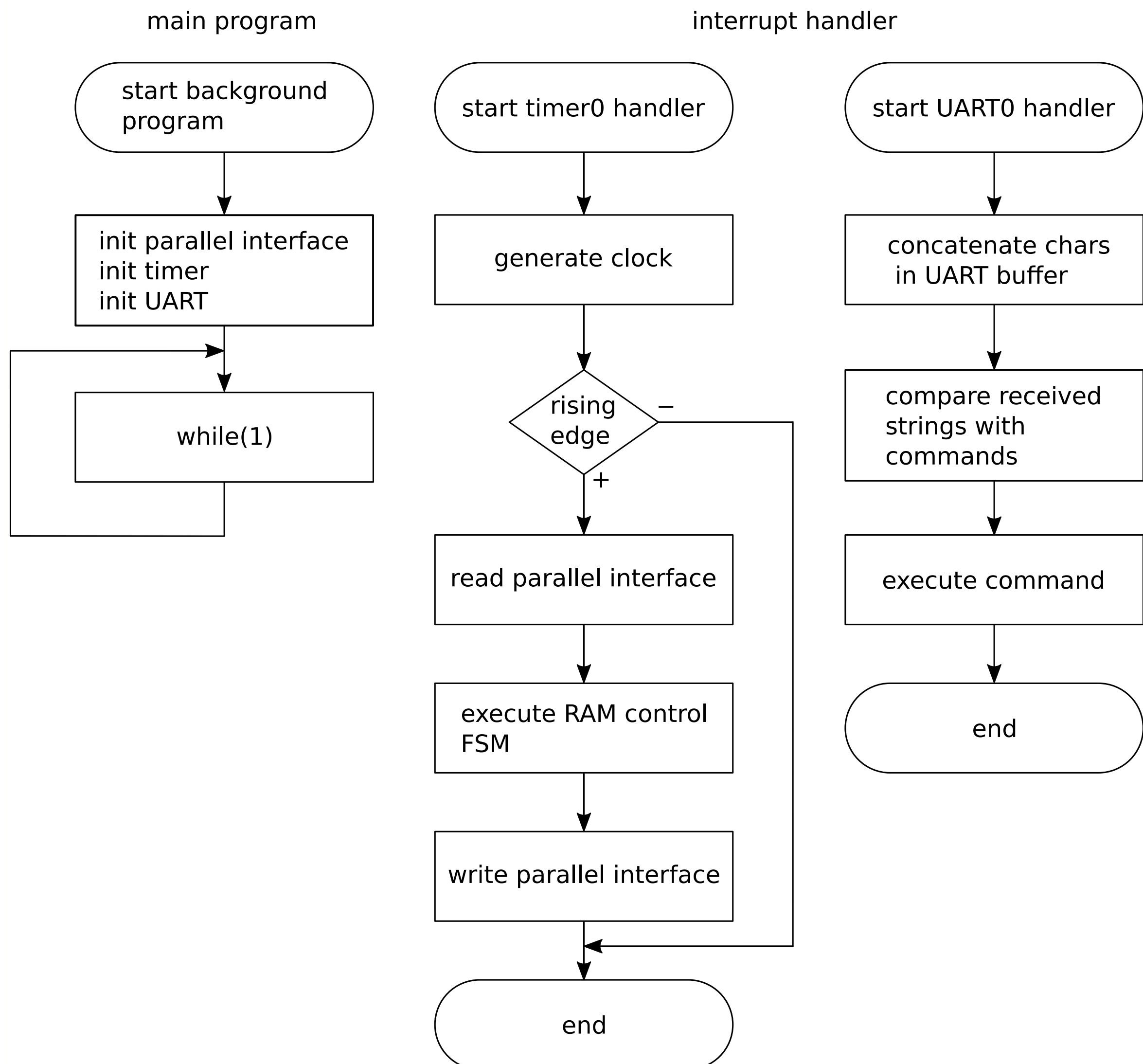


Abbildung 6.2: Programmablaufplan der parallel ablaufenden Tasks auf dem Mikrocontroller

Abbildung 6.2 zeigt den Programmablauf der implementierten Threads. Dargestellt sind das Hauptprogamm und jeweils eine Interrupt Service Routine (ISR) für ein Timer-Modul und ein UART-Modul.

Im Folgenden wird der Ablauf der main-Funktion und der ISRs beschrieben:

- `main`:
  - Innerhalb der main-Funktion werden Funktionen zur Initialisierung der verwendeten Peripheriekomponenten ausgeführt.
  - Nach der Initialisierung nimmt die main-Funktion einen aktiven Wartezustand an. Alle weiteren Verarbeitungen werden in den ISRs ausgeführt.
- `TIMER0_handler`:
  - Diese ISR wird zyklisch durch einen Timer-Interrupt ausgelöst. Sie dient zur Ansteuerung des FPGAs inklusive des RAM-Zugriffs und der Generierung des Taktes für den FPGA.
  - Der erste Schritt der Verarbeitung ist die Generierung des FPGA-Taktes durch Umschalten des entsprechenden GPIOs bei jedem Aufruf der ISR. Bei steigender Taktflanke erfolgt das Schreiben des `mod_clk_en`-Signals noch vor dem Schreiben des Taktsignals. Durch eine minimale Wartezeit wird sichergestellt, dass das `mod_clk_en`-Signal beim Ein- bzw. Ausschalten des Taktes der Signalverarbeitungsmodule rechtzeitig zum Takt anliegt. Andernfalls wäre die Ein- bzw. Abschaltung der Signalverarbeitungsmodule um einen Takt verzögert, was in der praktischen Realisierung schwierig zu handhaben ist.
  - Die Ansteuerung des FPGAs erfolgt nur bei positiven Taktflanken. Durch die Funktion `read_parallel_interface` werden die Werte an den GPIOs der Eingangssignale der parallelen Schnittstelle zum FPGA eingelesen. Danach erfolgt die Ausführung des Zustandsautomaten zur Realisierung des RAM-Zugriffs. Anschließend werden die GPIOs der Ausgangssignale der parallelen Schnittstelle durch die Funktion `write_parallel_interface` beschrieben.
- `UART_handler`:
  - Diese ISR wird aufgerufen, sobald Daten im FIFO-Speicher der UART verfügbar sind.
  - Die verfügbaren Daten werden zu einem String verkettet. Der entstehende String wird mit hinterlegten String-Befehlen verglichen. Sobald sich eine Übereinstimmung ergibt, wird der entsprechende Befehl ausgeführt.
  - Eine Besonderheit stellt das Einlesen der in den RAM zu schreibenden Daten in den Mikrocontroller über die UART dar. Nachdem der Einlesebefehl erkannt wurde, wird der Modus `send_ram_mode` aktiviert. In diesem werden die eingehenden Daten nicht mehr mit den String-Befehlen verglichen, sondern direkt in das Array `ram_input_data` geschrieben.

Zum Übertragen der RAM-Daten über die UART werden die folgenden Funktionen verwendet:

- `vals_to_terminal`: Sendet den Inhalt des globalen Arrays `ram_output_data` elementweise als hexadezimalen String über die UART an den PC.
- `read_parallel_interface`: Die eingehenden Daten werden zu einem String verkettet. Sobald ein String eine Länge von 8 Zeichen erreicht hat, wird er zum `uint32_t`-Datentyp konvertiert und in das globale Array `ram_input_data` geschrieben. Durch den entsprechenden Befehl wird der `send_ram_mode` nach abgeschlossener Übertragung wieder deaktiviert.

Im Folgenden sollen die innerhalb der Timer-ISR verwendeten Funktionen zum Lesen und Schreiben der Schnittstelle zum FPGA beschrieben werden:

- `read_parallel_interface`: Die an den als Eingang konfigurierten GPIOs anliegenden Werte werden in dieser Funktion in Variablen gespeichert. Diese Variablen werden per Referenz an die Funktion übergeben, dadurch stehen die beschriebenen Variablen direkt in der Timer-ISR zur Verfügung. Die Werte der GPIOs werden per Bitshift auf ihren binären Stellenwert gesetzt und innerhalb der Variablen addiert.
- `write_parallel_interface`: In dieser Funktion werden die Variablen zur Ansteuerung des FPGAs auf die GPIO-Ausgänge geschrieben. Die Variablen werden per Referenz übergeben. Die GPIOs werden portweise beschrieben. Dazu werden die einzelnen Bits aus den Variablen maskiert und per Bitshift so umgerechnet, dass der entsprechende Pin mit dem maskierten Wert beschrieben wird.

Die Funktionen zur Initialisierung der Periphierekomponenten sollen ebenfalls beschrieben werden:

- `init_parallel_interface`: Initialisiert die GPIOs, die zur Ansteuerung des FPGAs verwendet werden.
- `init_TIMER0`: Initialisiert Timer 0, sodass dieser periodisch im 32-Bit Modus abwärts zählt. Beim Erreichen des Endwerts wird ein Timer-Interrupt ausgelöst. Die Periodendauer des Taktes zur Ansteuerung des FPGAs und den Systemtakts des Mikrocontrollers werden als Parameter übergeben. Der Timer-Startwert wird aus den Parametern berechnet. Der Systemtakt wird auf 120 MHz festgelegt. Für die Ansteuerung des FPGA wird eine Taktfrequenz von 10 kHz gewählt. Dies gewährleistet, dass beim RAM-Zugriff keine störenden Wartezeiten auftreten.
- `init_UART`: Initialisiert UART 0 mit dem Protokoll `8N1`. Die Übertragung erfolgt mit einer Bitrate von 19200 Bit/s. Zusätzlich wird eine ISR registiert, in der eingehende Daten verarbeitet werden.

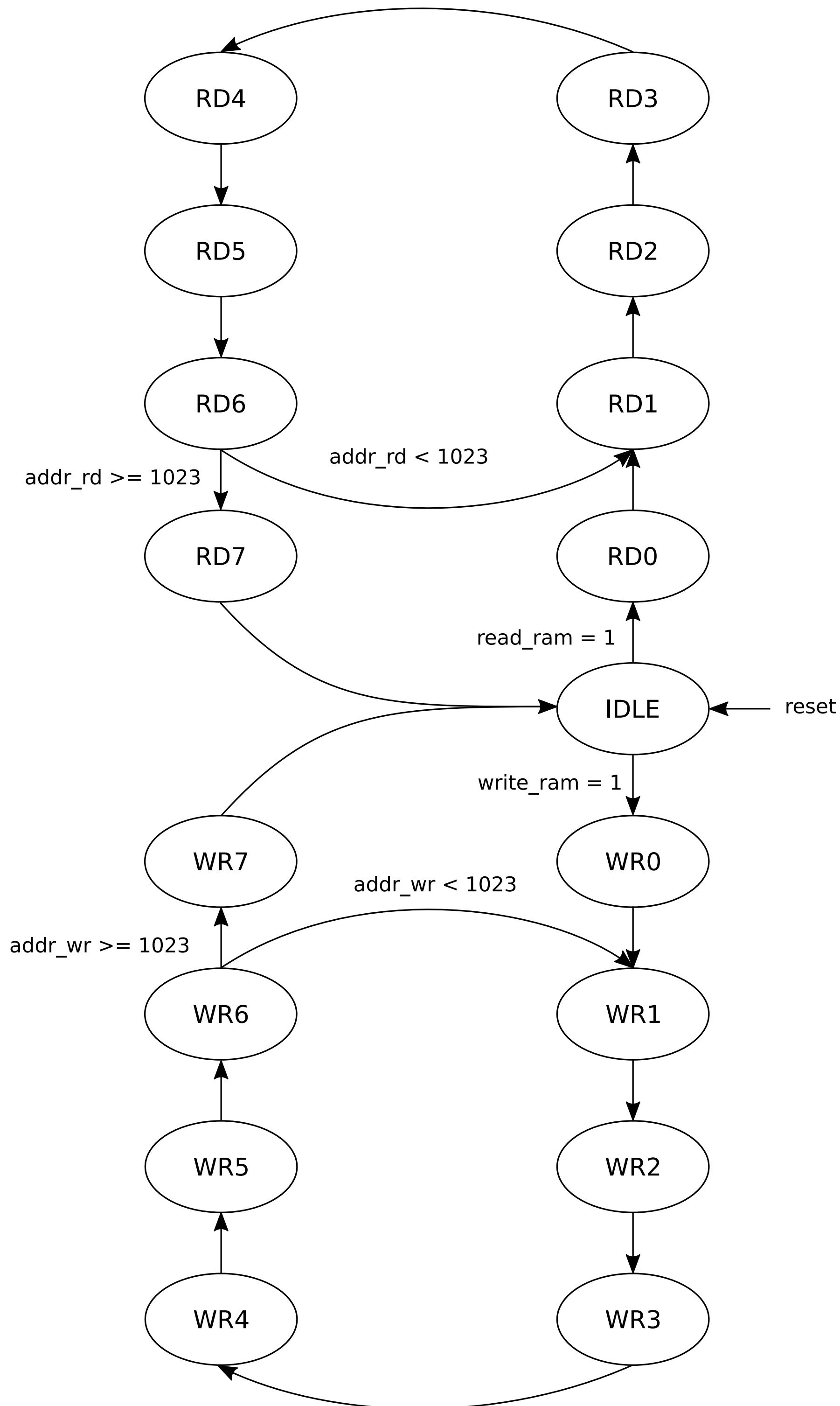


Abbildung 6.3: Zustandsautomat zur Ansteuerung des RAMs über den Memory Control Block

Abbildung 6.3 beschreibt den in der Timer-ISR ausgeführten Zustandsautomaten zur Realisierung des RAM-Zugriffs. Der Zugriff erfolgt über den Memory Control Block wie in Abschnitt 4.2 beschrieben. Der Zustandsautomat steuert den Memory Control Block mit der entsprechenden Schreib-/Lesesequenz an. Ein Lese- oder Schreibzugriff erfolgt stets für den gesamten RAM-Speicher. Nachfolgend sollen die einzelnen Zustände des Automaten beschrieben werden:

- `IDLE`: Initialisierungszustand. Warten auf Lese- oder Schreibbefehl.
- `RD0`: Deaktivierung des Modultaktes und der externen Modulansteuerung.
- `RD1`: Hochohmigschalten der Three-State-Treiber der Signalverarbeitungsmodule. Zuschalten des Memory Control Blocks an den RAM. Einlesen der Adresse in den Memory Control Block.
- `RD2`: Anfordern der Datenbits 31-24 aus dem Memory Control Block.
- `RD3`: Anfordern der Datenbits 23-16 aus dem Memory Control Block. Speichern der anliegenden Datenbits 31-24 in das Element der aktuellen Adresse des Arrays `ram_output_data`.
- `RD4`: Anfordern der Datenbits 15-8 aus dem Memory Control Block. Speichern der anliegenden Datenbits 23-16 in das Element der aktuellen Adresse des Arrays `ram_output_data`.
- `RD5`: Anfordern der Datenbits 7-0 aus dem Memory Control Block. Speichern der anliegenden Datenbits 15-8 in das Element der aktuellen Adresse des Arrays `ram_output_data`.
- `RD6`: Speichern der anliegenden Datenbits 7-0 in das Element der aktuellen Adresse des Arrays `ram_output_data`. Solange die letze Speicheradresse nicht erreicht ist, wird die Adresse inkrementiert und der Lesevorgang für das nächste Speicherwort wiederholt.
- `RD7`: Wenn der gesamte RAM gelesen wurde, werden die Three-State-Treiber des Memory Control Blocks hochohmig geschaltet und das aktive Signalverarbeitungsmodul wieder an den RAM geschaltet.
- `WR0`: Deaktivierung des Modultaktes und der externen Modulansteuerung.
- `WR1`: Hochohmigschalten der Three-State-Treiber der Signalverarbeitungsmodule. Zuschalten des Memory Control Blocks an den RAM. Einlesen der Adresse in dem Memory Control Block.
- `WR2`: Schreiben der Datenbits 31-24 aus dem Element des Array `ram_input_data` der aktuellen Schreibadresse in den Buffer des Memory Control Blocks.
- `WT3`: Schreiben der Datenbits 23-16 aus dem Element des Array `ram_input_data` der aktuellen Schreibadresse in den Buffer des Memory Control Blocks.

- `WR4`: Schreiben der Datenbits 15-8 aus dem Element des Array `ram_input_data` der aktuellen Schreibadresse in den Buffer des Memory Control Blocks.
- `WR5`: Schreiben der Datenbits 7-0 aus dem Element des Array `ram_input_data` der aktuellen Schreibadresse in den Buffer des Memory Control Blocks.
- `WR6`: Schreiben des Buffers in den RAM. Solange die letze Speicheradresse nicht erreicht ist, wird die Adresse inkrementiert und der Schreibvorgang für das nächste Speicherwort wiederholt.
- `WR7`: Wenn der gesamte RAM beschrieben wurde werden die Three-State-Treiber des Memory Control Blocks hochohmig geschaltet und das aktive Signalverarbeitungsmodul wieder an den RAM geschaltet.

6.2 Test der Gesamtfunktion auf dem FPGA

Zum Testen der Signalverarbeitungsmodule auf dem FPGA lassen sich in Octave Testskripte erstellen. Der typische Ablauf eines Testskriptes ist das Schreiben von Testdaten in den RAM, die Ausführung einer oder mehrerer Signalverarbeitungsmodule und die anschließende Auswertung der auf dem RAM befindlichen Daten.

Im Projekt stehen Testdaten für das Sensor-Array zur Verfügung. Diese sind im Octave-Workspace verfügbar. Das Testskript muss diese in ein für den RAM geeignetes Format übersetzen. Auch nach der Verarbeitung müssen die Daten aus dem RAM übersetzt werden, um eine Aussage über das Ergebnis der Verarbeitung treffen zu können.

Zur Ansteuerung der Testumgebung auf dem FPGA stehen die in Tabelle 6.2 aufgelisteten Befehle zur Verfügung. Diese werden vom Mikrocontroller ausgewertet und initiieren die entsprechende Ansteuerung des FPGAs durch den Mikrocontroller.

Tabelle 6.2: Implementierte Debugging-Befehle zur Erstellung von Testskripten in GNU Octave

Befehl	Beschreibung
-reset	Rücksetzen der Module auf dem FPGA
-nxtclk	Springt einen Takt vorwärts
-clkon	Einschalten des Modultakts
-clkoff	Ausschalten des Modultakts
-write	Schreibt das im MC gespeicherte Array ram_input_data in den RAM
-read	Liest den RAM und schreibt die Werte in das Array ram_output_data auf dem MC
-mselext	Externe Modulauswahl
-mselint	Interne Modulauswahl
-mod<x>	Auswahl Modul <x> (0-7)

Für die komplexeren Abläufe, wie das Senden und Empfangen der RAM-Daten sowie die Übersetzung der Daten in die Darstellungform auf dem RAM werden Funktionen implementiert. Diese Funktionen befinden sich in Listing B.17 bis B.21. Im folgenden sollen die einzelnen Funktionen beschrieben werden:

- `calculate_input_data`: Schreibt die Rohdaten des Sensor-Arrays in das Array `ram_input_data`. Die Daten werden so geschrieben, dass sie im Array vorliegen, wie im Abschnitt 3.2 beschrieben. Dazu erfolgt eine Umrechnung vom Gleitkommaformat `Double` ins Festkommaformat $s1Q10$. Die Daten werden als String gespeichert. Um über die UART-Schnittstelle weniger Daten übertragen zu müssen, werden die 32 Bits eines RAM-Wortes hexadezimal dargestellt, dazu sind 8 Zeichen erforderlich. Als Paramenter werden die Testdaten in Form eines Sinus- und Cosinusarrays übergeben.
- `serial_write_data`: Überträgt das gesamte Array `ram_input_data` an den Mikrocontroller. Als Parameter werden die initialisierte serielle Verbindung und das Array `ram_input_data` übergeben.
- `serial_read_data`: Sendet die Anforderung der RAM-Daten an den Mikrocontroller. Nachdem die Daten übertragen wurden, werden sie im Array `ram_output_data` gespeichert. Als Parameter wird die initialisierte serielle Verbindung übergeben.
- `calculate_output_data`: Erzeugt aus dem Array `ram_output_data` jeweils ein Array aus Sinus und Cosinuswerten. Die Umrechnung erfolgt aus dem Festkommaformat $s1Q10$, welches als hexadezimaler String dargestellt ist, in das Gleitkommaformat `Double`. Als Parameter werden das Array `ram_output_data` sowie die Start- und Endadresse der Elemente angegeben, aus denen die Sinus und Cosinusarrays erzeugt werden sollen.
- `calculacte_angle`: Extrahiert den auf dem FPGA berechneten Winkel aus dem Array `ram_output_data`. Dazu erfolgt eine Umrechnung vom Festkommaformat $s2Q9$ in das Gleitkommaformat `Double`. Das Winkelergebnis wird graphisch dargestellt. Als Parameter wird das Element des Arrays `ram_output_data` übergeben, welches die Winkelinformation beinhaltet.

6.3 Test der Winkelberechnung auf dem FPGA

Zum Test der Funktion der Winkelberechnung wird ein Testskript in Octave geschrieben. Dieses befindet sich in Listung B.16. Das Skript basiert auf den im vorangegangenen Abschnitt beschriebenen Befehlen und Funktionen. Der sequentielle Ablauf des Testskriptes ist in Abbildung 6.4 dargestellt.

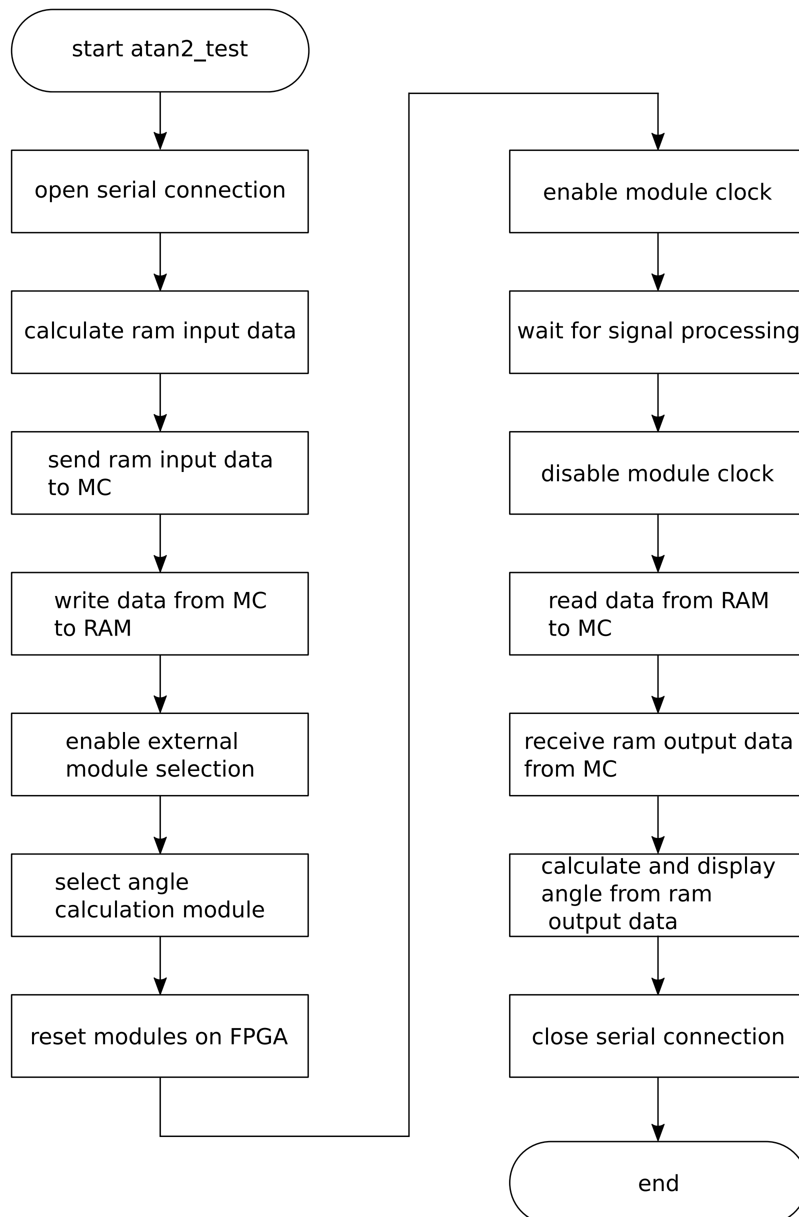


Abbildung 6.4: Programmablauf des erstellten Skriptes zum Testen des Winkelberechungsmoduls auf dem FPGA

Zur Ausführung des Testskriptes müssen die Testdaten des Sensor-Arrays im Octave Workspace vorhanden sein. Der Ablauf des Testskriptes wird im Folgenden beschrieben:

1. Aufbau der seriellen Verbindung zum Mikrocontroller.
2. Schreiben der Testdaten in das Array `ram_input_data` mithilfe der Funktion `calculate_input_data`. Als Parameter werden jeweils 8×8 Sinus- und Cosinuswerte übergeben. Das Array `ram_input_data` umfasst daher 64 Elemente.
3. Übertragen des Arrays `ram_input_data` auf den Mikrocontroller mit der Funktion `serial_write_data`.
4. Externe Auswahl des Winkelberechnungsmoduls und Rücksetzen vor der Verarbeitung.
5. Ausführung des Verarbeitungsprozesses des Winkelberechnungsmoduls über eine definierte Zeit. Alternativ kann das System auch mit einer definierten Anzahl an Takten angesteuert werden.
6. Auslesen des RAMs und Speicherung der Daten im Mikrocontroller.
7. Übertragen des Speicherinhalts aus dem Mikrocontroller und Speicherung der Daten im Array `ram_output_data` mit der Funktion `serial_read_data`.
8. Umrechnung und Darstellung des berechneten Winkels aus dem durch das Modul beschrieben Element des Arrays `ram_output_data` durch die Funktion `calculate_angle`.
9. Schließen der seriellen Verbindung.

Tabelle 6.3: Berechnete Winkel auf dem FPGA

cos	sin	Idealer Winkel (rad)	Idealer Winkel (°)	Berechneter Winkel (rad)	Berechneter Winkel (°)	|Abweichung| (rad)	|Abweichung| (°)
1,0000	0,0000	0,0000	0,00	0,0000	0,00	0,0000	0,00
0,8660	0,5000	0,5236	30,00	0,5215	29,88	0,0021	0,12
0,5000	0,8660	1,0472	60,00	1,0488	60,09	0,0016	0,09
0,0000	1,0000	1,5708	90,00	1,5704	89,98	0,0004	0,02
-0,5000	0,8660	2,0944	120,00	2,0919	119,86	0,0025	0,14
-0,8660	0,5000	2,6180	150,00	2,6191	150,06	0,0011	0,06
-1,0000	0,0000	3,1416	180,00	3,1406	179,94	0,0010	0,06
-0,8660	-0,5000	-2,6180	-150,00	-2,6191	-150,06	0,0011	0,06
-0,5000	-0,8660	-2,0944	-120,00	-2,0918	-119,85	0,0026	0,15
0,0000	-1,0000	-1,5708	-90,00	-1,5703	-89,97	0,0005	0,03
0,5000	-0,8660	-1,0472	-60,00	-1,0488	-60,09	0,0016	0,09
0,8660	-0,5000	-0,5236	-30,00	-0,5215	-29,88	0,0021	0,12

Mit dem Testskript wird das Winkelberechnungsmodul für verschiedene Winkel getestet. Gewählt werden die Winkel im Abstand von 30°, beginnend bei 0°. Dies hat im Gegensatz zu den Vielfachen von 45° den Vorteil, das auch Fehler bei der Korrektur des Quadranten erkannt werden können, bei denen der Winkel an einer der Achsen gespiegelt ist. In Tabelle 6.3 werden die idealen und die durch das Winkelberechungsmodul auf dem FPGA berechneten Winkel dargestellt und verglichen.

6.4 Abschätzung des Syntheseaufwands mit Cadence

Zum Zweck einer ersten Aufwandsabschätzung wird mit den Candence Tools Genus und Innovus eine Synthese des entwickelten VHDL-Codes durchgeführt. Synthetisiert wurden der Memory Control Block, der Module Control Block und das Winkelberechnungsmodul. Die Ergebnisse sind in den Synthesereports dokumentiert. Diese befinden sich in den Listings C.1 bis C.9. In Tabelle 6.4 werden ausgewählte Parameter aus den Synthesereports für die einzelnen Module gegenübergestellt. Die Ergebnisse der Synthese werden in Abschnitt 7.1 diskutiert.

Tabelle 6.4: Übersicht der Ergebnisse der Cadence-Synthese

Modul	Zellen	Fläche in μm^2	Leistung in nW	Timing Slack in ps
Atan2	1077	105814,800	2541821,191	21057
Memory Control	321	45117,800	1434382,025	28019
Module Control	7	1401,400	37221,941	60733

7 Schlussfolgerungen

In diesem Kapitel werden die Ergebnisse dieser Arbeit und die erreichten Ziele beschrieben. Außerdem werden weiterführende Aufgaben und Ansätze für Verbesserungen in folgenden Arbeiten genannt.

7.1 Zusammenfassung

In dieser Bachelorarbeit wurde ein Testsystem für die Signalverarbeitung eines Sensor-Arrays für ein experimentelles Chipdesign entwickelt.

Auf einem FPGA wurde ein System implementiert, in dem Signalverarbeitungsmodule über einen gemeinsamen RAM kommunizieren. Ein Mikrocontroller ermöglicht den externen Zugriff auf den RAM-Speicher und das Debuggen des Signalverarbeitungsablaufs auf dem FPGA. Zur Bereitstellung und Auswertung von Testdaten sowie zur Steuerung des Testablaufs wurden in GNU Octave Skripte und Funktionen erstellt, die mit dem Mikrocontroller kommunizieren. Weiterhin wurde ein Signalverarbeitungsmodul zur Winkelberechnung entworfen. Mit der Cadence-Umgebung wurde eine Synthese der in VHDL beschriebenen Module für eine erste Aufwandsabschätzung durchgeführt.

Auf dem FPGA wurde ein Konzept zur Kommunikation zwischen den Signalverarbeitungsmodulen über den gemeinsamen RAM entwickelt. Das Problem der parallelen Beschaltung des RAMs durch mehrere Signalverarbeitungsmodule konnte dabei mithilfe von Three-State-Treibern gelöst werden. Auch der externe Speicherzugriff konnte durch die Implementierung des Memory Control Blocks realisiert werden. Mithilfe von Three-State-Treibern und dem zusätzlichen Abschalten des Modultaktes ist ein Speicherzugriff nach jedem Takt möglich.

Das Winkelberechnungsmodul konnte erfolgreich mithilfe des CORDIC-Algorithmus implementiert werden. Der umgesetzte Algorithmus benötigt lediglich 11 Takte zur Verarbeitung, was sehr wenig ist, verglichen mit anderen im ISAR-Projekt entwickelten Signalverarbeitungsalgorithmen. Die Winkelberechnung hat eine mittlere Abweichung von $< 0,2°$ und liefert somit ein gutes Ergebnis. Der Fehler der Winkelberechnung ist deutlich kleiner als der Winkelfehler gängiger magnetischer Sensoren. Nur im Bereich der beiden Eingangswerte um Null ergeben sich größere Abweichungen, im Extremfall

bis zu $36,3427°$. Als Grund für diese hohen Abweichungen wird eine numerische Instabilität im Bereich der kleinsten, im Festkomma-Zahlenformat darstellbaren Eingangswerte vermutet.

Mit dem Mikrocontroller konnte durch Schaffung eines parallelen Interfaces aus GPIOs eine Ansteuerung des FPGAs realisiert werden. Diese ermöglicht das Lesen und Schreiben des RAMs und das Debugging der Signalverarbeitungsmodule. Die Mikrocontroller-Umgebung kann ebenfalls zum Testen des gefertigten ASICs verwendet werden. Die Eingabe der Test-Daten und -Befehle sowie das Auslesen der Daten aus dem RAM wurden mittels der UART-Schnittstelle implementiert. Die Software auf dem Mikrocontroller erfüllt die an sie gestellten Anforderungen. Im Bereich der UART-Schnittstelle gibt es jedoch noch Optimierungspotential, so verfügt diese über keinerlei Erkennung von Übertragungsfehlern und die hardware-aufwändige Auswertung von String-Funktionen innerhalb des Interrupthändlers ist ebenfalls noch nicht optimal gelöst.

In GNU Octave wurde ein Test-Skript erstellt, mit dem die Winkelberechnung auf dem FPGA erfolgreich getestet werden konnte. Mit dem Test-Skript konnte außerdem die Funktionalität des Gesamtsystems nachgewiesen werden. Für die korrekte Ausführung der Test-Sequenz müssen die implementierten Funktionen auf dem Mikrocontroller und FPGA ebenfalls korrekt ausgeführt werden. Die in Octave implementierten Funktionen und Debug-Befehle dienen als Grundgerüst für den Test weiterer Signalverarbeitungsmodule mit der entwickelten Testumgebung.

Durch die Cadence-Synthese konnte gezeigt werden, dass sich eine Winkelberechnungsfunktion mit einem geringem Hardwareaufwand von 1077 Standardzellen auf einem ASIC implementieren lässt. Auch hier ist der Syntheseaufwand für andere Signalverarbeitungsalgorithmen im ISAR-Projekt deutlich höher. Die Komponenten zur Ansteuerung des RAMs (Memory Control Block, 321 Standardzellen) und der Signalverarbeitungsmodule (Module Control Block, 7 Standardzellen) konnten mit noch geringerem Hardwareaufwand synthetisiert werden. Das Synthesergebnis ist positiv zu bewerten und bestätigt die Auswahl des CORDIC-Algorithmus zur Umsetzung der Winkelberechnung.

Zusammenfassend lässt sich festhalten, dass eine funktionierende Testumgebung geschaffen wurde, mit deren Hilfe weitere, im Projekt entwickelte Signalverarbeitungsmodule auf einem FPGA getestet werden können. Es hat sich gezeigt, dass eine intensive Testphase auf einer FPGA-Plattform neben der Simulation des VHDL-Codes deutliche Vorteile hat. Während der Bearbeitung dieser Thesis konnten bereits weitere im Projektteam entwickelte Signalverarbeitungsmodule erfolgreich getestet werden. Auch das Winkelberechungsmodul konnte erfolgreich und effizient implementiert werden. Das Ziel, ein Testsystem für die Signalverarbeitung eines Sensor-Arrays für ein experimentelles Chipdesign inklusive eines Winkelberechungsmoduls zu entwickeln, kann somit als erreicht angesehen werden.

7.2 Ausblick

Mithilfe des entwickelten Test-Systems können weitere, im ISAR-Projekt entwickelte Signalverarbeitungsmodule auf dem Zedboard getestet werden. Dazu können die implementierten Debug-Befehle und -Funktionen sukzessive erweitert werden.

Wenn alle weiteren Signalverarbeitungsmdule implementiert und erfolgreich getestet wurden, kann ein Chipentwurf erfolgen. Das Konzept der Kommunikation der Signalverarbeitungsmodule über einen gemeinsamen RAM eignet sich gut zum Testen der Module, auch in der Phase des ersten Chipentwurfs. Für die spätere Anwendung des Chips ist der gewählte Ansatz der sequentiellen Verarbeitung jedoch aufgrund des ineffizienten Zeitverhaltens nicht geeignet, stattdessen würde sich eine Pipeline-Struktur anbieten. Die entwickelte Software zur Ansteuerung des FPGAs kann zukünftig ebenfalls zum Testen des gefertigten Chips genutzt werden.

Weiterhin können Verbesserungen am Debug-System vorgenommen werden, wie die Erweiterung der UART-Schnittstelle um ein Protokoll zur Fehlererkennung. Ebenso kann die Software auf dem Mikrocontroller effektiver strukturiert werden, sodass die Auswertung der empfangenen String-Befehle nicht mehr innerhalb eines Interrupt-Handlers durchgeführt wird. Dies könnte es ermöglichen, die Übertragungsgeschwindigkeit UART zu erhöhen, ohne dass es zu Datenverlust kommt.

Ein weiterer Ansatzpunkt ist die Verbesserung der Genauigkeit der Winkelberechnung mit dem CORDIC-Algorithmus im Bereich um Null beider Eingangswerte. So wäre es denkbar, die Winkelergebnisse mit der größten Abweichung in einer Lookup-Tabelle zu hinterlegen, anstatt sie zu berechnen. Außerdem können Untersuchungen zum Syntheseaufwand des CORDIC-Algorithmus bei der Verarbeitung mehrerer Iterationsschritte innerhalb eines Taktes angestellt werden.

Literaturverzeichnis

[1] Gross, Rudolf ; Marx, Achim: *Spinelektronik.* April 2004. – Vorlesungsskript

[2] Krey, Martin: *Systemarchitektur und Signalverarbeitung für die Diagnose von magnetischen ABS-Sensoren.* Januar 2015. – Dissertation

[3] Slatter, Rolf: Tunnelmagnetoresistive Sensoren für die Antriebstechnik. In: *Elektronikpraxis* 14 (2017), Juli, S. 28–30

[4] Airich, Viktor: *Charakterisierung magnetoresistiver Sensor-Arrays mittels eines automatisierten Messsystems.* Januar 2018. – Bachelorarbeit

[5] NVE Corporation: *Ultralow Power TMR Angle Sensors*, April 2017

[6] Begic, Abraham: *Tunnel-Magnetoresistives Sensor-Array - Controllersteuerung, Platinen-Layout und Prüfstands-Erprobung.* Juli 2018. – Bachelorarbeit

[7] Beuler, Marcel: *CORDIC-Algorithmus zur Auswertung elementarer Funktionen in Hardware.* Fachhochschule Giessen Friedberg `http://digdok.bib.thm.de/volltexte/2009/4148/pdf/CORDIC_Algorithmus.pdf`. – Zugriff am 16. Juli 2018

[8] Papula, L.: *Mathematische Formelsammlung: Für Ingenieure und Naturwissenschaftler.* Springer Fachmedien Wiesbaden, 2017. – ISBN 9783658161958

[9] Risse, Thomas: CORDIC-Algorithmen Verbinden Mathematik, Computer-Architektur und Anwendungen. In: *Global Journal of Engineering Education* Vol.8, No.3 (2004), S. 311–318

[10] Reichenbach, Marc ; Schmidt, Michael: *VHDL - CORDIC Verfahren.* Informatik 3 / Rechnerarchitektur Universität Erlangen Nürnberg `http://www3.informatik.uni-erlangen.de/Lehre/VHDL-RA/SS2012/lectures/02-cordic.pdf`

[11] Xilinx Incorporated: *Zynq-7000 SoC Data Sheet*, Juli 2018. – Rev. 1.11.1

[12] Avnet Incorporated: *ZedBoard (Zynq Evaluation and Development) Hardware User's Guide*, Januar 2014. – Rev. 2.2

[13] Texas Instruments Incorporated: *Tiva TM4C1294NCPDT Microcontroller Data Sheet*, Juni 2014. – Rev. 15863.2743

[14] Reichardt, J.: *Lehrbuch Digitaltechnik: Eine Einführung mit VHDL.* De Gruyter, 2013 (De Gruyter Studium). – ISBN 9783486753875

[15] Reichardt, J. ; Schwarz, B.: *VHDL-synthese: Entwurf Digitaler Schaltungen und Systeme*. De Gruyter, 2015 (De Gruyter Studium). – ISBN 9783110375053

[16] de Dinechin, Florent ; Istoan, Matei: *Hardware implementations of fixed-point Atan2*. INSA-Lyon, CITI, F-69621 Villeurbanne, France, `https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7203793&tag=1`. – Zugriff am 16. Juli 2018

[17] Zucchelli, Giorgia: *Fixed-Point ATAN2 using CORDIC* `https://de.mathworks.com/matlabcentral/fileexchange/19316-fixed-point-atan2-using-cordic?focused=6783300&tab=example`. – Zugriff am 16. Juli 2018

[18] Texas Instruments Incorporated: *TivaWare Peripheral Driver Library User's Guide*, Juli 2016. – Rev. 2.1.3.156

Anhang

A Diverses

Tabelle A.1: Pinbelegung der parallelen Schnittstelle zwischen Mikrocontroller und FPGA

Pin	Pinmode	Belegung TM4C1294	Funktion
JA1	-	PC4	RESERVE
JA2	In	PC5	WE
JA3	In	PC6	nRESET
JA4	In	PE5	CLK
JA7	Out	PE0	DOUT(0)
JA8	Out	PE1	DOUT(1)
JA9	Out	PE2	DOUT(2)
JA10	Out	PE3	DOUT(3)
JB1	In	PD3	CTRL_EXT_EN
JB2	Out	PC7	MOD_RDY
JB3	In	PB2	MOD_CLK_EN
JB4	In	PB3	MOD_SEL_EXT_EN
JB7	Out	PD7	DOUT(4)
JB8	Out	PA6	DOUT(5)
JB9	Out	PM4	DOUT(6)
JB10	Out	PM5	DOUT(7)
JC1_P	In	PP0	DIN(0)
JC1_N	In	PP1	DIN(1)
JC2_P	In	PD4	DIN(2)
JC2_N	In	PD5	DIN(3)
JC3_P	In	PB4	DIN(8)
JC3_N	In	PB5	DIN(9)
JC4_P	In	PK0	DSEL(0)
JC4_N	In	PK1	DSEL(1)
JD1_P	In	PQ0	DIN(4)
JD1_N	In	PP4	DIN(5)
JD2_P	In	PN5	DIN(6)
JD2_N	In	PN4	DIN(7)
JD3_P	In	PK2	DSEL(2)
JD3_N	Out	PK3	MOD_OUT(0)
JD4_P	Out	PA4	MOD_OUT(1)
JD4_N	Out	PA5	MOD_OUT(2)

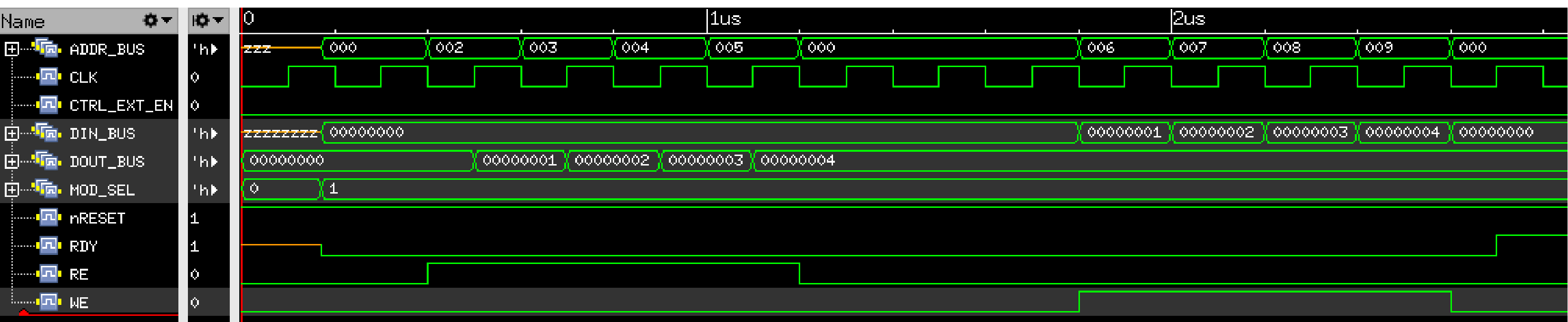


Abbildung A.1: Ausschnitt eines Signal-Zeit-Diagramms zur Darstellung der Ein- und Ausgangssignale des Test-Moduls. Die Simulation wurde mit dem Cadence-Tool NC-Sim erstellt. Die Zahlenwerte sind im hexadezimal dargestellt.

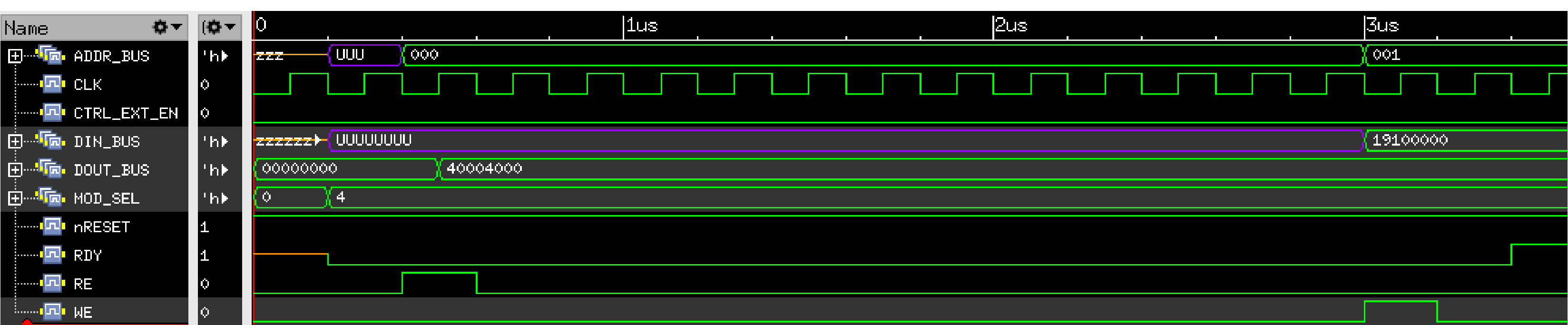


Abbildung A.2: Ausschnitt eines Signal-Zeit-Diagramms zur Darstellung der Ein- und Ausgangssingale des Winkelberechnungsmoduls. Die Simulation wurde mit dem Cadence-Tool NC-Sim erstellt. Die Zahlenwerte sind hexadezimal dargestellt.

B Quelltexte

B.1 VHDL-Quellcode

Quellcode B.1: toplevel.vhd

```
    ------------------------------------------------------------------------------------
    -- Entitiy : TOPLEVEL
    ------------------------------------------------------------------------------------
    -- Copyright 2018
 5  -- Filename          : toplevel.vhd
    -- Creation date     : 2018-04-26
    -- Authors(s)        : Jannes Helck
    -- Version           : 1.00
    -- Description       : Top level
10  ------------------------------------------------------------------------------------
    -- File History :
    -- Date          Version       Author        Comment
    -- 2018-04-26    1.00          J.Helck       Creation of file
    ------------------------------------------------------------------------------------
15
    library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

20  use work.BRAM_PKG.all;
    use work.MEMORY_CONTROL_PKG.all;
    use work.MODULE_CONTROL_PKG.all;
    use work.ATAN2_PKG.all;
    --use work.TEST_MODULE_PKG.all;
25
    entity TOPLEVEL is
    port (
        CLK                : in  std_logic;
        nRESET             : in  std_logic;
30      CTRL_EXT_EN        : in  bit;
        MOD_SEL_EXT_EN     : in  bit;
        MOD_OUT            : out std_logic_vector(2 downto 0);
        MOD_RDY            : out std_logic;
        MOD_CLK_EN         : in  bit;
35      DSEL               : in  std_logic_vector(2 downto 0);
        WE                 : in  bit;
        DIN                : in  std_logic_vector(9 downto 0);
        DOUT               : out std_logic_vector(7 downto 0)

40  );
    end TOPLEVEL;

    architecture ARCH of TOPLEVEL is

45  signal RAM_EN        : std_logic;
    signal MOD_SEL       : std_logic_vector(2 downto 0);
    signal MOD_CLK       : std_logic;
```

```
    signal RDY           : std_logic;
    signal ADDR_BUS      : std_logic_vector(9 downto 0);
50  signal DIN_BUS       : std_logic_vector(31 downto 0);
    signal DOUT_BUS      : std_logic_vector(31 downto 0);

    signal WE_RAM        : std_logic;
    signal RE_RAM        : std_logic;
55
    signal WE_C1         : std_logic;
    signal RE_C1         : std_logic;
    signal WE_C4         : std_logic;
    signal RE_C4         : std_logic;
60

    begin

    RAM_EN  <= '1';       -- RAM permanent enabled
65  MOD_OUT <= MOD_SEL;
    MOD_RDY <= RDY;
    MOD_CLK <= CLK when MOD_CLK_EN = '1' else '0';


70  --logic OR-combination of RAM WE/RE signals
    WE_RAM <= WE_C1 or WE_C4;
    RE_RAM <= RE_C1 or RE_C4;

    -- instantiation of component : MEMORY_CONTROL
75  C1: MEMORY_CONTROL port map (
        CLK               => CLK,
        nRESET            => nRESET,
        CTRL_EXT_EN       => CTRL_EXT_EN,
        MOD_SEL_EXT_EN    => MOD_SEL_EXT_EN,
80      MOD_SEL           => MOD_SEL,
        DIN               => DIN,
        DOUT              => DOUT,
        DSEL              => DSEL,
        WE_IN             => WE,
85      WE                => WE_C1,
        RE                => RE_C1,
        ADDR_BUS          => ADDR_BUS,
        DIN_BUS           => DIN_BUS,
        DOUT_BUS          => DOUT_BUS
90  );

    -- instantiation of component: BRAM
    C2: BRAM port map (
        CLK               => CLK,
95      nRESET            => nRESET,
        EN                => RAM_EN,
        WE                => WE_RAM,
        RE                => RE_RAM,
        ADDR              => ADDR_BUS,
100     DIN               => DIN_BUS,
        DOUT              => DOUT_BUS
    );

    -- instantiation of component: MODULE_CONTROL
105 C3: MODULE_CONTROL port map (
        CLK               => MOD_CLK,
        nRESET            => nRESET,
        MOD_SEL_EXT_EN    => MOD_SEL_EXT_EN,
        MOD_SEL           => MOD_SEL,
110     RDY               => RDY
    );
```

```
      -- instantiation of component: ATAN2
      C4: ATAN2 port map (
115       CLK              => MOD_CLK,
          nRESET           => nRESET,
          MOD_SEL          => MOD_SEL,
          CTRL_EXT_EN      => CTRL_EXT_EN,
          RDY              => RDY,
120       WE               => WE_C4,
          RE               => RE_C4,
          ADDR_BUS         => ADDR_BUS,
          DIN_BUS          => DIN_BUS,
          DOUT_BUS         => DOUT_BUS
125   );

      -- instantiation of component: TEST_MODULE
      --C5: TEST_MODULE port map (
      --     CLK              => MOD_CLK,
130   --     nRESET           => nRESET,
      --     CTRL_EXT_EN      => CTRL_EXT_EN,
      --     MOD_SEL          => MOD_SEL,
      --     RDY              => RDY,
      --     WE               => WE_C5,
135   --     RE               => RE_C5,
      --     ADDR_BUS         => ADDR_BUS,
      --     DIN_BUS          => DIN_BUS,
      --     DOUT_BUS         => DOUT_BUS
      --);
140
      end ARCH;
```

Quellcode B.2: bram.vhd

```
-----------------------------------------------------------------------------
-- Entitiy : BRAM
-----------------------------------------------------------------------------
-- Copyright 2018
-- Filename         : bram.vhd
-- Creation date    : 2018-05-03
-- Authors(s)       : Jannes Helck
-- Version          : 1.00
-- Description      : Block RAM for transferring data between the
--                    signal processing modules on FPGA
-----------------------------------------------------------------------------
-- File History :
-- Date         Version     Author      Comment
-- 2018-05-03   1.00        J.Helck     Creation of file
-----------------------------------------------------------------------------

----------------------------------- Package -------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package BRAM_PKG is

component BRAM
port (
    CLK   : in  std_logic;
    nRESET: in  std_logic;
    EN    : in  std_logic;
    WE    : in  std_logic;
    RE    : in  std_logic;
    ADDR  : in  std_logic_vector(9 downto 0);
    DIN   : in  std_logic_vector(31 downto 0);
    DOUT  : out std_logic_vector(31 downto 0)
);
end component;

end BRAM_PKG;

--------------------------------- end Package -----------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity BRAM is
port (
    CLK   : in  std_logic;
    nRESET: in  std_logic;
    EN    : in  std_logic;
    WE    : in  std_logic;
    RE    : in  std_logic;
    ADDR  : in  std_logic_vector(9 downto 0);
    DIN   : in  std_logic_vector(31 downto 0);
    DOUT  : out std_logic_vector(31 downto 0)
);
end BRAM;

architecture ARCH of BRAM is

 type RAM_TYPE is array (1023 downto 0) of std_logic_vector(31 downto 0);
 signal BRAM : RAM_TYPE:= (others => "00000000000000000000000000000000");
```

```
     attribute RAM_STYLE : string;
65   attribute RAM_STYLE of BRAM : signal is "block";

    begin

    RAM_P: process(CLK)
70  begin
    if CLK'event and CLK = '1' then
        if EN = '1' then
            if WE = '1' then
                BRAM(to_integer(unsigned(ADDR))) <= DIN;
75              DOUT <= DIN;
            elsif RE = '1' then
                DOUT <= BRAM(to_integer(unsigned(ADDR)));
            else
                DOUT <= "00000000000000000000000000000000";
80          end if;
        end if;
    end if;
    end process;

85  end ARCH;
```

Quellcode B.3: memory_control.vhd

```
1  ----------------------------------------------------------------------------
   -- Entitiy : MEMORY_CONTROL
   ----------------------------------------------------------------------------
   -- Copyright 2018
5  -- Filename           : memory_control.vhd
   -- Creation date      : 2018-05-11
   -- Authors(s)         : Jannes Helck
   -- Version            : 1.00
   -- Description        : Interface for external RAM access
10 ----------------------------------------------------------------------------
   -- File History :
   -- Date          Version      Author      Comment
   -- 2018-05-11    1.00         J.Helck     Creation of file
   ----------------------------------------------------------------------------
15
   ---------------------------------------- Package ----------------------------------------

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

   package MEMORY_CONTROL_PKG is

   component MEMORY_CONTROL
25 port (
       CLK              : in  std_logic;
       nRESET           : in  std_logic;
       CTRL_EXT_EN      : in  bit;
       MOD_SEL_EXT_EN   : in  bit;
30     MOD_SEL          : out std_logic_vector(2 downto 0);
       DIN              : in  std_logic_vector(9 downto 0);
       DOUT             : out std_logic_vector(7 downto 0);
       DSEL             : in  std_logic_vector(2 downto 0);
       WE_IN            : in  bit;
35     WE               : out std_logic;
       RE               : out std_logic;
       ADDR_BUS         : out std_logic_vector(9 downto 0);
       DIN_BUS          : out std_logic_vector(31 downto 0);
       DOUT_BUS         : in  std_logic_vector(31 downto 0)
40 );
   end component;

   end MEMORY_CONTROL_PKG;

45 -------------------------------------- end Package --------------------------------------

   library ieee;
   use ieee.std_logic_1164.all;
   use ieee.numeric_std.all;
50
   entity MEMORY_CONTROL is
   port (
       CLK              : in  std_logic;
       nRESET           : in  std_logic;
55     CTRL_EXT_EN      : in  bit;
       MOD_SEL_EXT_EN   : in  bit;
       MOD_SEL          : out std_logic_vector(2 downto 0);
       DIN              : in  std_logic_vector(9 downto 0);
       DOUT             : out std_logic_vector(7 downto 0);
60     DSEL             : in  std_logic_vector(2 downto 0);
       WE_IN            : in  bit;
       WE               : out std_logic;
       RE               : out std_logic;
```

```
        ADDR_BUS          : out std_logic_vector(9 downto 0);
65      DIN_BUS           : out std_logic_vector(31 downto 0);
        DOUT_BUS          : in  std_logic_vector(31 downto 0)
    );
    end MEMORY_CONTROL;

70  architecture ARCH of MEMORY_CONTROL is

    signal WRITE_BUFFER : std_logic_vector(31 downto 0);
    signal DIN_INT      : std_logic_vector(31 downto 0);
    signal ADDR         : std_logic_vector(9 downto 0);
75  signal WE_INT       : std_logic;
    signal RE_INT       : std_logic;


    begin
80
    MEM_CTRL_P: process (CLK, nRESET)
    begin
    if nRESET = '0' then
        ADDR <= "0000000000";
85      DOUT <= "00000000";
        WRITE_BUFFER <= "00000000000000000000000000000000";
        DIN_INT <= "00000000000000000000000000000000";
        WE_INT <= '0';
        RE_INT <= '0';
90  elsif CLK'event and CLK = '0' then
        if CTRL_EXT_EN = '1' then
            if WE_IN = '1' then
                case DSEL is
                    -- write adress
95                  when "001" => ADDR <= DIN;
                    -- Byte 0 to buffer
                    when "011" => WRITE_BUFFER(31 downto 24) <= DIN(7 downto 0);
                    -- Byte 1 to buffer
                    when "010" => WRITE_BUFFER(23 downto 16) <= DIN(7 downto 0);
100                 -- Byte 2 to buffer
                    when "110" => WRITE_BUFFER(15 downto 8) <= DIN(7 downto 0);
                    -- Byte 3 to buffer
                    when "111" => WRITE_BUFFER(7 downto 0) <= DIN(7 downto 0);
                    -- write buffer
105                 when "101" => DIN_INT <= WRITE_BUFFER;
                    when others => null;
                end case;
                DOUT <= "00000000";
                WE_INT <= '1';
110             RE_INT <= '0';
            else
                case DSEL is
                    -- write adress
                    when "001" => ADDR <= DIN; DOUT <= "00000000";
115                 -- read Byte 0
                    when "011" => DOUT <= DOUT_BUS(31 downto 24);
                    -- read Byte 1
                    when "010" => DOUT <= DOUT_BUS(23 downto 16);
                    -- read Byte 2
120                 when "110" => DOUT <= DOUT_BUS(15 downto 8);
                    -- read Byte 3
                    when "111" => DOUT <= DOUT_BUS(7 downto 0);
                    when others => DOUT <= "00000000";
                end case;
125             WRITE_BUFFER <= "00000000000000000000000000000000";
                DIN_INT <= "00000000000000000000000000000000";
                WE_INT <= '0';
                RE_INT <= '1';
```

```
          end if;
130     else
          DOUT <= "00000000";
          WE_INT <= '0';
          RE_INT <= '0';
        end if;
135   end if;
      end process;

      -- Drivers for parallel RE/WE-Signals to RAM
      RE <= RE_INT when CTRL_EXT_EN = '1' else '0';
140   WE <= WE_INT when CTRL_EXT_EN = '1' else '0';

      -- Three state bus drivers
      ADDR_BUS <= ADDR when CTRL_EXT_EN = '1' else (others=>'Z');
      DIN_BUS <= DIN_INT when CTRL_EXT_EN = '1' else (others=>'Z');
145   MOD_SEL <= DSEL when MOD_SEL_EXT_EN = '1' else (others=>'Z');

      end ARCH;
```

Quellcode B.4: module_control.vhd

```
----------------------------------------------------------------------------
-- Entitiy : MODULE_CONTROL
----------------------------------------------------------------------------
-- Copyright 2018
-- Filename           : module_control.vhd
-- Creation date      : 2018-05-12
-- Authors(s)         : Jannes Helck
-- Version            : 1.00
-- Description        : Control Unit for sequential activation of the
--                      signal processing modules
----------------------------------------------------------------------------
-- File History :
-- Date           Version     Author       Comment
-- 2018-05-12     1.00        J.Helck      Creation of file
----------------------------------------------------------------------------

------------------------------------ Package ------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package MODULE_CONTROL_PKG is

component MODULE_CONTROL
port (
    CLK             : in  std_logic;
    nRESET          : in  std_logic;
    MOD_SEL_EXT_EN  : in  bit;
    MOD_SEL         : out std_logic_vector(2 downto 0);
    RDY             : in  std_logic
);
end component;

end MODULE_CONTROL_PKG;

---------------------------------- end Package ----------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity MODULE_CONTROL is
port (
    CLK             : in  std_logic;
    nRESET          : in  std_logic;
    MOD_SEL_EXT_EN  : in  bit;
    MOD_SEL         : out std_logic_vector(2 downto 0);
    RDY             : in  std_logic
);
end MODULE_CONTROL;

architecture ARCH of MODULE_CONTROL is

type STATES is (M1, M2, M3, M4);
signal STATE, NEXT_STATE: STATES := M1;
signal MOD_SEL_INT : std_logic_vector(2 downto 0);

begin

-- State machine
STATE_MEM: process(CLK, nRESET)
begin
```

```
        if nRESET = '0' then STATE <= M1;
65      elsif CLK  = '1' and CLK'event then
               STATE <= NEXT_STATE;
        end if;
    end process;

70  TRANSITION_LOGIC: process(RDY, STATE)
    begin
        NEXT_STATE <= STATE;
        case STATE is

75          when M1 => MOD_SEL_INT <= "001";
                       --if RDY = '1' then NEXT_STATE <= M2;
                       --end if;
                       NEXT_STATE <= M2;    -- no transition condition active

80          when M2 => MOD_SEL_INT <= "010";
                       --if RDY = '1' then NEXT_STATE <= M3;
                       --end if;
                       NEXT_STATE <= M3;    -- no transition condition active

85          when M3 => MOD_SEL_INT <= "011";
                       --if RDY = '1' then NEXT_STATE <= M4;
                       --end if;
                       NEXT_STATE <= M4;    -- no transition condition active

90          when M4 => MOD_SEL_INT <= "100";
                       if RDY = '1' then NEXT_STATE <= M1;
                       end if;
                       --NEXT_STATE <= M1;

95      end case;
    end process;

    -- Three state bus drivers
    MOD_SEL <= MOD_SEL_INT when MOD_SEL_EXT_EN = '0' else (others=>'Z');
100
    end ARCH;
```

Quellcode B.5: atan2.vhd

```
--------------------------------------------------------------------------------
-- Entitiy : ATAN2
--------------------------------------------------------------------------------
-- Copyright 2018
-- Filename          : atan2.vhd
-- Creation date     : 2018-03-09
-- Authors(s)        : Jannes Helck
-- Version           : 1.00
-- Description       : Signal processing module for calculating the angle of
--                     two input arguments
--------------------------------------------------------------------------------
-- File History :
-- Date          Version      Author       Comment
-- 2018-03-09    1.00         J.Helck      Creation of file
--------------------------------------------------------------------------------

---------------------------------- Package -------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package ATAN2_PKG is

component ATAN2
port(
    CLK             : in  std_logic;
    nRESET          : in  std_logic;
    MOD_SEL         : in  std_logic_vector(2 downto 0);
    CTRL_EXT_EN     : in  bit;
    RDY             : out std_logic;
    WE              : out std_logic;
    RE              : out std_logic;
    ADDR_BUS        : out std_logic_vector(9 downto 0);
    DIN_BUS         : out std_logic_vector(31 downto 0);
    DOUT_BUS        : in  std_logic_vector(31 downto 0)
);
end component;

end ATAN2_PKG;
-------------------------------- end Package -----------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ATAN2 is
generic(
    MOD_NR          : std_logic_vector := "100";          -- module number
    ADDR_RD         : std_logic_vector := "0000000000";   -- address read
    ADDR_WR         : std_logic_vector := "0000000001"    -- address write
);

port(
    CLK             : in  std_logic;
    nRESET          : in  std_logic;
    MOD_SEL         : in  std_logic_vector(2 downto 0);
    CTRL_EXT_EN     : in  bit;
    RDY             : out std_logic;
    WE              : out std_logic;
    RE              : out std_logic;
    ADDR_BUS        : out std_logic_vector(9 downto 0);
```

```
        DIN_BUS         : out std_logic_vector(31 downto 0);
65      DOUT_BUS        : in  std_logic_vector(31 downto 0)
    );
    end ATAN2;

    architecture ARCH of ATAN2 is
70
    type STATES is (
        INIT, RD0, RD1, C0, C1, C2, C3, C4, C5,
        C6, C7, C8, C9, C10, WR0, WR1, WT0
    );
75
    signal STATE, NEXT_STATE: STATES := INIT;

    type ANGLES_TYPE is array(1 to 9) of signed(11 downto 0);
    constant ANGLES : ANGLES_TYPE := (
80      "000011101101", "000001111101", "000001000000", "000000100000",
        "000000010000", "000000001000", "000000000100", "000000000010",
        "000000000001"
    );

85  type QUADRANT_TYPE is (Q1, Q2, Q3, Q4);
    signal QUADRANT: QUADRANT_TYPE;

    constant PIDIV1 : signed(11 downto 0) := "011001001000";
    constant PIDIV2 : signed(11 downto 0) := "001100100100";
90

    signal INPUT    : signed(31 downto 0);
    signal DIN      : std_logic_vector(31 downto 0);
    signal WE_INT   : std_logic;
95  signal RE_INT   : std_logic;
    signal ADDR     : std_logic_vector(9 downto 0);
    signal RDY_INT  : std_logic;
    signal RESULT   : signed(11 downto 0);

100 begin

    -- atan2 CORDIC state machine
    STATE_MEM: process(CLK, nRESET)
    begin
105     if nRESET = '0' or MOD_SEL /= MOD_NR then
            STATE <= INIT;
        elsif CLK = '1' and CLK'event then
            STATE <= NEXT_STATE;
        end if;
110 end process;

    TRANSITION_LOGIC: process(STATE, MOD_SEL, CLK)
    variable A      : signed(11 downto 0);
    variable X_TEMP : signed(11 downto 0);
115 variable X      : signed(11 downto 0);
    variable Y      : signed(11 downto 0);
    variable X_IN   : signed(11 downto 0);
    variable Y_IN   : signed(11 downto 0);
    variable SWAP   : bit := '0';
120
    begin

    if nRESET = '0' or MOD_SEL /= MOD_NR then
        NEXT_STATE <= INIT;
125 elsif CLK = '0' and CLK'event then
        NEXT_STATE <= STATE;
        case STATE is
            when INIT => RDY_INT <= '0';
```

```
                        if MOD_SEL = MOD_NR then
130                        NEXT_STATE <= RD0;
                        end if;

               when RD0   => NEXT_STATE <= RD1;

135            when RD1   => NEXT_STATE <= C0;

               when C0    => X_IN := resize(INPUT(31 downto 21), 12); --s1Q10 to S2Q9
                             Y_IN := resize(INPUT(15 downto 5), 12);  --s1Q10 to S2Q9

140                     if Y_IN(11) = '0' then          -- determination
                           if X_IN(11) = '0' then       -- of quadrant
                              Y := Y_IN;
                              X := X_IN;
                              QUADRANT <= Q1;
145                        else
                              Y := Y_IN;
                              X := -X_IN;
                              QUADRANT <= Q2;
                           end if;
150                     else
                           if X_IN(11) = '1' then
                              Y := -Y_IN;
                              X := -X_IN;
                              QUADRANT <= Q3;
155                        else
                              Y := -Y_IN;
                              X := X_IN;
                              QUADRANT <= Q4;
                           end if;
160                     end if;

                        if X < Y then
                           X_TEMP := X;
                           X := Y;
165                        Y := X_TEMP;
                           SWAP := '1';
                        else
                           SWAP := '0';
                        end if;
170                     NEXT_STATE <= C1;

                        -- X_IN = 0 or Y_IN = 0
                        if X_IN = "000000000000" or Y_IN = "000000000000" then
                           -- X_IN = 0 and Y_IN > 0
175                        if X_IN = "000000000000" and Y_IN(11) = '0' then
                              A := PIDIV2;
                           -- X_IN = 0 and Y_IN < 0
                           elsif X_IN = "000000000000" and Y_IN(11) = '1' then
                              A := -PIDIV2;
180                        end if;
                           -- X_IN > 0 and Y_IN = 0
                           if X_IN(11) = '0' and Y_IN = "000000000000" then
                              A := "000000000000";
                           -- X_IN < 0 and Y_IN = 0
185                        elsif X_IN(11) = '1' and Y_IN = "000000000000" then
                              A := PIDIV1;
                           end if;
                           -- X_IN = 0 and Y_IN = 0
                           if X_IN = "000000000000" and Y_IN = "000000000000" then
190                           A := "000000000000";
                           end if;
                        NEXT_STATE <= WR0;
                        end if;
```

```
195
            when C1    => if Y(11) = '1' then
                             A := -ANGLES(1);
                             X_TEMP := X - resize(Y(11 downto 1), 12);
                             Y := resize(X(11 downto 1), 12) + Y;
200                          X := X_TEMP;
                          else
                             A := ANGLES(1);
                             X_TEMP := X + resize(Y(11 downto 1), 12);
                             Y := -resize(X(11 downto 1), 12) + Y;
205                          X := X_TEMP;
                          end if;
                          NEXT_STATE <= C2;


210         when C2    => if Y(11) = '1' then
                             A := A - ANGLES(2);
                             X_TEMP := X - resize(Y(11 downto 2), 12);
                             Y := resize(X(11 downto 2), 12) + Y;
                             X := X_TEMP;
215                       else
                             A := A + ANGLES(2);
                             X_TEMP := X + resize(Y(11 downto 2), 12);
                             Y := -resize(X(11 downto 2), 12) + Y;
                             X := X_TEMP;
220                       end if;
                          NEXT_STATE <= C3;


            when C3    => if Y(11) = '1' then
225                          A := A - ANGLES(3);
                             X_TEMP := X - resize(Y(11 downto 3), 12);
                             Y := resize(X(11 downto 3), 12) + Y;
                             X := X_TEMP;
                          else
230                          A := A + ANGLES(3);
                             X_TEMP := X + resize(Y(11 downto 3), 12);
                             Y := -resize(X(11 downto 3), 12) + Y;
                             X := X_TEMP;
                          end if;
235                       NEXT_STATE <= C4;

            when C4    => if Y(11) = '1' then
                             A := A - ANGLES(4);
                             X_TEMP := X - resize(Y(11 downto 4), 12);
240                          Y := resize(X(11 downto 4), 12) + Y;
                             X := X_TEMP;
                          else
                             A := A + ANGLES(4);
                             X_TEMP := X + resize(Y(11 downto 4), 12);
245                          Y := -resize(X(11 downto 4), 12) + Y;
                             X := X_TEMP;
                          end if;
                          NEXT_STATE <= C5;

250         when C5    => if Y(11) = '1' then
                             A := A - ANGLES(5);
                             X_TEMP := X - resize(Y(11 downto 5), 12);
                             Y := resize(X(11 downto 5), 12) + Y;
                             X := X_TEMP;
255                       else
                             A := A + ANGLES(5);
                             X_TEMP := X + resize(Y(11 downto 5), 12);
                             Y := -resize(X(11 downto 5), 12) + Y;
```

```
                             X := X_TEMP;
260                      end if;
                         NEXT_STATE <= C6;

            when C6    => if Y(11) = '1' then
                             A := A - ANGLES(6);
265                          X_TEMP := X - resize(Y(11 downto 6), 12);
                             Y := resize(X(11 downto 6), 12) + Y;
                             X := X_TEMP;
                          else
                             A := A + ANGLES(6);
270                          X_TEMP := X + resize(Y(11 downto 6), 12);
                             Y := -resize(X(11 downto 6), 12) + Y;
                             X := X_TEMP;
                         end if;
                         NEXT_STATE <= C7;
275
            when C7    => if Y(11) = '1' then
                             A := A - ANGLES(7);
                             X_TEMP := X - resize(Y(11 downto 7), 12);
                             Y := resize(X(11 downto 7), 12) + Y;
280                          X := X_TEMP;
                          else
                             A := A + ANGLES(7);
                             X_TEMP := X + resize(Y(11 downto 7), 12);
                             Y := -resize(X(11 downto 7), 12) + Y;
285                          X := X_TEMP;
                         end if;
                         NEXT_STATE <= C8;

            when C8    => if Y(11) = '1' then
290                          A := A - ANGLES(8);
                             X_TEMP := X - resize(Y(11 downto 8), 12);
                             Y := resize(X(11 downto 8), 12) + Y;
                             X := X_TEMP;
                          else
295                          A := A + ANGLES(8);
                             X_TEMP := X + resize(Y(11 downto 8), 12);
                             Y := -resize(X(11 downto 8), 12) + Y;
                             X := X_TEMP;
                         end if;
300                      NEXT_STATE <= C9;

            when C9    => if Y(11) = '1' then
                             A := A - ANGLES(9);
                             X_TEMP := X - resize(Y(11 downto 9), 12);
305                          Y := resize(X(11 downto 9), 12) + Y;
                             X := X_TEMP;
                          else
                             A := A + ANGLES(9);
                             X_TEMP := X + resize(Y(11 downto 9), 12);
310                          Y := -resize(X(11 downto 9), 12) + Y;
                             X := X_TEMP;
                         end if;
                         NEXT_STATE <= C10;

315
            when C10   => if SWAP = '1' then      -- abs(X_IN) < abs(Y_IN)
                             A := PIDIV2 - A;
                         end if;

320                      case QUADRANT is         -- angle correction
                            when Q1 => null;      -- depending on quadrant
                            when Q2 => A := PIDIV1 - A;
                            when Q3 => A := -PIDIV1 + A;
```

```
                              when Q4 => A := -A;
325                       end case;

                          RESULT <= A;
                          NEXT_STATE <= WR0;

330           when WR0  => NEXT_STATE <= WR1;

              when WR1  => NEXT_STATE <= WT0;

              when WT0  => RDY_INT <= '1';
335                        if MOD_SEL /= MOD_NR then
                              NEXT_STATE <= INIT;
                           end if;

          end case;
340   end if;
      end process;

      -- Dedicated process for memory access
      RAM_IO: process(CLK)
345   begin
          if CLK = '0' and CLK'event then
              case STATE is
                  when INIT => RE_INT <= '0';
                               WE_INT <= '0';
350
                  when RD0  => ADDR <= ADDR_RD;
                               RE_INT <= '1';

                  when RD1  => INPUT <= signed(DOUT_BUS);
355                            RE_INT <= '0';

                  when WR0  => WE_INT <= '1';
                               ADDR <= ADDR_WR;
                               DIN <= std_logic_vector(RESULT) & "00000000000000000000";
360
                  when WR1  => WE_INT <= '0';

                  when others => null;
              end case;
365       end if;
      end process;

      -- Drivers for parallel RE/WE-Signals to RAM
      RE <= RE_INT when MOD_SEL = MOD_NR and CTRL_EXT_EN = '0' else '0';
370   WE <= WE_INT when MOD_SEL = MOD_NR and CTRL_EXT_EN = '0' else '0';

      -- Three state bus drivers
      ADDR_BUS <= ADDR when MOD_SEL = MOD_NR and CTRL_EXT_EN = '0' else (others=>'Z');
      DIN_BUS <= DIN when MOD_SEL = MOD_NR and CTRL_EXT_EN = '0' else (others=>'Z');
375   RDY <= RDY_INT when MOD_SEL = MOD_NR else 'Z';

      end ARCH;
```

Quellcode B.6: test_module.vhd

```
    ----------------------------------------------------------------------------------
    -- Entitiy : TEST_MODULE
    ----------------------------------------------------------------------------------
    -- Copyright 2018
 5  -- Filename          : test_module.vhd
    -- Creation date     : 2018-05-17
    -- Authors(s)        : Jannes Helck
    -- Version           : 1.00
    -- Description       : Module for testing the memory access of different
10  --                     signal processing modules
    ----------------------------------------------------------------------------------
    -- File History :
    -- Date          Version      Author      Comment
    -- 2018-05-17    1.00         J.Helck     Creation of file
15  ----------------------------------------------------------------------------------

    ------------------------------------ Package ------------------------------------

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

    package TEST_MODULE_PKG is

25  component TEST_MODULE
    port(
        CLK                 : in  std_logic;
        nRESET              : in  std_logic;
        CTRL_EXT_EN         : in  bit;
30      MOD_SEL             : in  std_logic_vector(2 downto 0);
        RDY                 : out std_logic;
        WE                  : out std_logic;
        RE                  : out std_logic;
        ADDR_BUS            : out std_logic_vector(9 downto 0);
35      DIN_BUS             : out std_logic_vector(31 downto 0);
        DOUT_BUS            : in  std_logic_vector(31 downto 0)
    );
    end component;

40  end TEST_MODULE_PKG;

    ---------------------------------- end Package ----------------------------------

    library ieee;
45  use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

    entity TEST_MODULE is
    generic(
50      MOD_NR              : std_logic_vector := "001";      -- module number
        DATA_SIZE           : natural := 16;                  -- internal data storage
        COUNTER_WIDTH       : natural := 3;                   -- counter witdth
        QEND                : unsigned := "010";              -- counter end value
        START_ADDR_RD       : unsigned := "0000000010";       -- start address read
55      END_ADDR_RD         : unsigned := "0000000101";       -- end address read
        START_ADDR_WR       : unsigned := "0000000110";       -- start address write
        END_ADDR_WR         : unsigned := "0000001001"        -- end address write
    );

60  port(
        CLK                 : in  std_logic;
        nRESET              : in  std_logic;
        CTRL_EXT_EN         : in  bit;
```

```
         MOD_SEL              : in   std_logic_vector(2 downto 0);
65       RDY                  : out  std_logic;
         WE                   : out  std_logic;
         RE                   : out  std_logic;
         ADDR_BUS             : out  std_logic_vector(9 downto 0);
         DIN_BUS              : out  std_logic_vector(31 downto 0);
70       DOUT_BUS             : in   std_logic_vector(31 downto 0)
     );
     end TEST_MODULE;

     architecture ARCH of TEST_MODULE is
75
     signal WE_INT    : std_logic;
     signal RE_INT    : std_logic;
     signal ADDR      : std_logic_vector(9 downto 0);
     signal DIN       : std_logic_vector(31 downto 0);
80   signal RDY_INT   : std_logic;

     signal QPLUS     : unsigned(COUNTER_WIDTH-1 downto 0) := (others => '0');
     signal Q         : unsigned(COUNTER_WIDTH-1 downto 0) := (others => '0');
     signal CTR_RDY   : std_logic;
85
     type STATES is (INIT, RD0, RD1, RD2, WT0, WR0, WR1, WR2, WT1);
     signal STATE, NEXT_STATE: STATES := INIT;

     type DATA_TYPE is array(0 to DATA_SIZE-1) of signed(31 downto 0);
90   signal DATA : DATA_TYPE := (others => "00000000000000000000000000000000");

     begin

     -- Counter (for simulating module delay time)
95   COUNTER_STATE_MEM: process(nRESET, CLK, STATE)
     begin
         if nRESET = '0' or STATE /= WT0 then
             Q <= (others => '0');
         elsif CLK  = '0' and CLK'event then
100          Q <= QPLUS;
         end if;
     end process;

     COUNTER_TRANSITION_LOGIC: process(Q)
105  begin
         if Q = QEND then
             QPLUS <= (others => '0');
             CTR_RDY <= '1';
         else
110          QPLUS <= Q + 1;
             CTR_RDY <= '0';
         end if;
     end process;

115
     -- Module state machine
     STATE_MEM: process(CLK, nRESET)
     begin
         if nRESET = '0' or MOD_SEL /= MOD_NR then
120          STATE <= INIT;
         elsif CLK  = '1' and CLK'event then
             STATE <= NEXT_STATE;
         end if;
     end process;
125
     TRANSITION_LOGIC: process(STATE, MOD_SEL, CTR_RDY, ADDR)
     begin
         NEXT_STATE <= STATE;
```

```
        case STATE is
130
            when INIT  => RDY_INT <= '0';
                          if MOD_SEL = MOD_NR then
                             NEXT_STATE <= RD0;
                          end if;
135
            when RD0   => NEXT_STATE <= RD1;

            when RD1   => if unsigned(ADDR) >= END_ADDR_RD then
                             NEXT_STATE <= RD2;
140                       end if;

            when RD2   => NEXT_STATE <= WT0;

            when WT0   => if CTR_RDY = '1' then
145                          NEXT_STATE <= WR0;
                          end if;

            when WR0   => NEXT_STATE <= WR1;

150         when WR1   => if unsigned(ADDR) >= END_ADDR_WR then
                             NEXT_STATE <= WR2;
                          end if;

            when WR2   => NEXT_STATE <= WT1;
155
            when WT1   => RDY_INT <= '1';
                          if MOD_SEL /= MOD_NR then
                             NEXT_STATE <= INIT;
                          end if;
160
        end case;
    end process;

    -- Dedicated process for memory access
165 RAM_IO: process(CLK)
    variable ADDR_WR : unsigned(9 downto 0);
    variable ADDR_RD : unsigned(9 downto 0);
    begin
        if CLK = '0' and CLK'event then
170         case STATE is
                when INIT  => RE_INT <= '0';
                              WE_INT <= '0';
                              DIN <= "00000000000000000000000000000000";
                              ADDR <= "0000000000";
175
                when RD0   => ADDR_RD := START_ADDR_RD;
                              RE_INT <= '1';
                              ADDR <= std_logic_vector(START_ADDR_RD);

180             when RD1   => DATA(to_integer(ADDR_RD - START_ADDR_RD))
                                 <= signed(DOUT_BUS);

                              ADDR_RD := ADDR_RD + 1;
                              ADDR <= std_logic_vector(ADDR_RD);
185
                when RD2   => DATA(to_integer(ADDR_RD - START_ADDR_RD))
                                 <= signed(DOUT_BUS);

                              ADDR <= "0000000000";
190                           RE_INT <= '0';

                when WR0   => ADDR_WR := START_ADDR_WR;
                              WE_INT <= '1';
```

```
                              DIN <= std_logic_vector(DATA(to_integer(
195                              ADDR_WR - START_ADDR_WR)));

                              ADDR <= std_logic_vector(START_ADDR_WR);

                    when WR1   => ADDR_WR := ADDR_WR + 1;
200                           DIN <= std_logic_vector(DATA(to_integer(
                                 ADDR_WR - START_ADDR_WR)));

                              ADDR <= std_logic_vector(ADDR_WR);

205                 when WR2   => ADDR <= "0000000000";
                              WE_INT <= '0';
                              DIN <= "00000000000000000000000000000000";

                    when others => null;
210             end case;
            end if;
        end process;

    -- Drivers for parallel RE/WE-Signals to RAM
215 RE <= RE_INT when MOD_SEL = MOD_NR and CTRL_EXT_EN = '0' else '0';
    WE <= WE_INT when MOD_SEL = MOD_NR and CTRL_EXT_EN = '0' else '0';

    -- Three state bus drivers
    ADDR_BUS <= ADDR when MOD_SEL = MOD_NR and CTRL_EXT_EN = '0' else (others=>'Z');
220 DIN_BUS <= DIN when MOD_SEL = MOD_NR and CTRL_EXT_EN = '0' else (others=>'Z');
    RDY <= RDY_INT when MOD_SEL = MOD_NR else 'Z';

    end ARCH;
```

B.2 C-Quellcode

Quellcode B.7: main.h

```c
/*
 * main.h
 *
 *  Created on: Apr 13, 2018
 *      Author: Jannes Helck
 */

#ifndef MAIN_H_
#define MAIN_H_

#include <stdint.h>
#include <stdbool.h>
#include "tm4c1294ncpdt.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"

#include "parallel_interface.h"
#include "timer0.h"
#include "uart0.h"

#endif /* MAIN_H_ */
```

Quellcode B.8: main.c

```
/*
 * main.c
 *
 *  Created on: Apr 13, 2018
 *      Author: Jannes Helck
 */

#include "main.h"

void main(void){

    IntMasterDisable(); // enable all interrupts in NVIC
    uint32_t clk_period = 100; // clock period in us
    uint32_t sysclk;
    uint8_t wt = 0;

    // set clock to 120 Mhz
    sysclk = SysCtlClockFreqSet((SYSCTL_OSC_MAIN | SYSCTL_USE_PLL
            | SYSCTL_XTAL_25MHZ | SYSCTL_CFG_VCO_480), 120000000);

    wt++;    // wait

    init_parallel_interface();
    init_TIMER0(&clk_period, &sysclk);
    init_UART();
    IntMasterEnable(); // enable all interrupts in NVIC

    while(1);
}
```

Quellcode B.9: timer0.h

```
/*
 * timer0.h
 *
 *  Created on: Apr 29, 2018
 *      Author: Jannes Helck
 */

#ifndef TIMER0_H_
#define TIMER0_H_

#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "tm4c1294ncpdt.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/timer.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "parallel_interface.h"

//*****************************************************************************
// Function:     init_TIMER0
// Description:  Function for initialization of Timer 0. The first parameter is
//               the system clock of the microcontroller and the second the clock
//               period for the digital system under test.
// Parameters:   uint32_t *clk_period, uint32_t *sysclk
// Globals:      none
// Return:       void
//*****************************************************************************
void init_TIMER0(uint32_t*, uint32_t*);

//*****************************************************************************
// Function:     TIMER0_handler
// Description:  Interrupt handler for clock generation and hardware abstraction.
//               Executes a state machine for read/write actions of the RAM on
//               the tested system and calls GPIO read/write functions
//               synchronous to clock.
// Parameters:   void
// Globals:      bool nreset, bool mod_clk_en, bool mod_sel_ext_en, uint8_t dsel,
//               bool read_ram, bool write_ram, uint8_t next_clk,
//               uint32_t ram_input_data[1024], uint32_t ram_output_data[1024],
//               enum state_type state
// Return:       void
//*****************************************************************************
void TIMER0_handler(void);

#endif /* TIMER0_H_ */
```

Quellcode B.10: timer0.c

```
/*
 * timer0.c
 *
 *  Created on: Apr 29, 2018
 *      Author: Jannes Helck
 */

#include "timer0.h"

// macros
#define WAIT 0x00
#define ADRESS 0x01
#define BYTE0 0x03
#define BYTE1 0x02
#define BYTE2 0x06
#define BYTE3 0x07
#define WRITE_BUFFER 0x05

// state variables
enum state_type {
    IDLE, WT, WR0, WR1, WR2, WR3, WR4, WR5, WR6,
    WR7, RD0, RD1, RD2, RD3, RD4, RD5, RD6, RD7
};
enum state_type state = IDLE;

// Globals
bool nreset = 1;
bool mod_clk_en = 0;
bool mod_sel_ext_en = 0;
uint8_t dsel = 0x0;
bool read_ram = 0;
bool write_ram = 0;
uint8_t next_clk = 0;
uint32_t ram_input_data[1024] = {0};
uint32_t ram_output_data[1024] = {0};


void init_TIMER0(uint32_t *clk_period, uint32_t *sysclk){

    // 2 timer interrups each period, 0.5 us * clk_periode
    uint32_t timer_val = ((*sysclk/2000000) * *clk_period)-1;
    uint8_t wt = 0;

    printf("Clock period: %d us\n", *clk_period);

    //TIMER0 init
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);wt++; // clock enable
    TimerDisable(TIMER0_BASE, TIMER_A);                 // disable timer
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);    // 32 bit periodic downward
    TimerLoadSet(TIMER0_BASE, TIMER_A, timer_val);      // load timer value
    TimerIntRegister(TIMER0_BASE, TIMER_A, TIMER0_handler); // register interrupt
    IntPrioritySet(INT_TIMER0A, 0x20);                  // set medium int priority
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);    // enable timer interrupt
    TimerEnable(TIMER0_BASE, TIMER_A);                  // enable timer

}

void TIMER0_handler(void){

    static bool ctrl_ext_en = 0;
    static bool we = 0;
    static uint16_t din = 0x000;
    static uint8_t dout = 0x00;
```

```
static bool mod_rdy = 0;
static uint8_t mod_out = 0x00;

static uint32_t counter = 0;
static uint16_t addr_rd = 0;
static uint16_t addr_wr = 0;
static bool mod_clk_temp;

TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
//IntMasterDisable(); // disable all interrupts in NVIC

// generate clock
if(GPIOPinRead(GPIO_PORTE_BASE, GPIO_PIN_5)){        // falling edge

        GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_5, 0);

        // one clock forward (module clock)
        if(next_clk > 1){
            mod_clk_en = 0;
            next_clk = 0;
        }
        if(next_clk >= 1){
            mod_clk_en = 1;
            next_clk++;
        }

} else {                                             // rising edge

    if(nreset==0){
        counter = 0;
    }

    if(mod_clk_en==1){

        if(nreset==1){
            counter++;
        }
        if(mod_rdy==1){
            printf("Takt: %d ## Modul: %d ## mod_rdy: %d\n",
                   counter, mod_out, mod_rdy);
        }
    }

    // MOD_CLK_EN => PB2,
    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2 , mod_clk_en<<2);
    SysCtlDelay(100);
    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_5, GPIO_PIN_5); // FPGA-Clock


    read_parallel_interface(&mod_out, &dout, &mod_rdy);

    // state machine for RAM access
    switch(state){
        case IDLE: if(write_ram==1){
                       state = WR0;
                       write_ram = 0;
                       mod_clk_temp = mod_clk_en; // save state of mod_clk
                       mod_clk_en = 0;
                       mod_sel_ext_en = 0;

                   } else if(read_ram ==1){
                       state = RD0;
                       read_ram = 0;
                       mod_clk_temp = mod_clk_en; // save state of mod_clk
                       mod_clk_en = 0;
```

```
                mod_sel_ext_en = 0;
            } else {
                we = 0;
                din = 0;
                ctrl_ext_en = 0;
            }
            break;

        case WR0: state = WR1;
                  ctrl_ext_en = 1;
                  we = 1;
                  dsel = ADRESS;
                  din = addr_wr;
                  break;

        case WR1: state = WR2;                    // write bit 31-24
                  din = ((ram_input_data[addr_wr] & 0xFF000000)>>24);
                  dsel = BYTE0;
                  break;

        case WR2: state = WR3;                    // write bit 23-16
                  din = ((ram_input_data[addr_wr] & 0x00FF0000)>>16);
                  dsel = BYTE1;
                  break;

        case WR3: state = WR4;                    // write bit 15-8
                  din = ((ram_input_data[addr_wr] & 0x0000FF00)>>8);
                  dsel = BYTE2;
                  break;

        case WR4: state = WR5;                    // write bit 7-0
                  din = (ram_input_data[addr_wr] & 0x000000FF);
                  dsel = BYTE3;
                  break;

        case WR5: state = WR6;
                  din = 0;
                  dsel = WRITE_BUFFER;
                  break;

        case WR6: if(addr_wr < 1023){    // for each element of RAM
                      state = WR1;
                      dsel = ADRESS;
                      addr_wr++;
                      din = addr_wr;
                  } else {
                      state = WR7;
                      dsel = WAIT;
                      din = 0;
                      addr_wr = 0;
                      ctrl_ext_en = 0;
                      printf("Write RAM operation finished\n");
                  }
                  break;

        case WR7: state = IDLE;
                  mod_clk_en = mod_clk_temp; // restore mod_clk state
                  break;

        case RD0: state = RD1;
                  ctrl_ext_en = 1;
                  dsel = ADRESS;
                  din = addr_rd;
                  break;
```

```
                case RD1: state = RD2;
                          dsel = BYTE0;
                          break;

                case RD2: state = RD3;        // read bit 31-24
                          ram_output_data[addr_rd] = (dout<<24);
                          dsel = BYTE1;
                          break;

                case RD3: state = RD4;        // read bit 23-16
                          ram_output_data[addr_rd] += (dout<<16);
                          dsel = BYTE2;
                          break;

                case RD4: state = RD5;        // read bit 15-8
                          ram_output_data[addr_rd] += (dout<<8);
                          dsel = BYTE3;
                          break;

                case RD5: state = RD6;        // read bit 7-0
                          ram_output_data[addr_rd] += dout;
                          dsel = WAIT;
                          break;

                case RD6: if(addr_rd < 1023){ // for each element of RAM
                              state = RD1;
                              dsel = ADRESS;
                              addr_rd++;
                              din = addr_rd;
                          } else {
                              state = RD7;
                              dsel = WAIT;
                              din = 0;
                              addr_rd = 0;
                              ctrl_ext_en = 0;
                              printf("Read RAM operation finished\n");
                           }
                           break;

                case RD7: state = IDLE;
                          mod_clk_en = mod_clk_temp; // restore mod_clk state
                          break;

            }
            write_parallel_interface(&nreset, &ctrl_ext_en,
                                     &mod_sel_ext_en, &dsel, &we, &din);
        }

        //IntMasterEnable(); // enable all interrupts in NVIC
    }
```

Quellcode B.11: uart0.h

```
/*
 * uart0.h
 *
 *  Created on: May 14, 2018
 *      Author: Jannes Helck
 */

#ifndef UART0_H_
#define UART0_H_

#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>
#include "stdio.h"
#include "tm4c1294ncpdt.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/uart.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"

//****************************************************************************
// Function:    init_UART
// Description: Initialization of UART0
// Parameters:  void
// Globals:     none
// Return:      void
//****************************************************************************
void init_UART(void);

//****************************************************************************
// Function:    UART_send
// Description: Sends strings via UART. First argument is the start address of
//              the string to be send, the second is the string length.
// Parameters:  const uint8_t *pui8Buffer, uint32_t ui32Count
// Globals:     none
// Return:      void
//****************************************************************************
void UART_send(const uint8_t*, uint32_t);

//****************************************************************************
// Function:    UART_handler
// Description: Concatenates incoming char values and compares them with
//              defined commands. If they confirm the commands will be executed.
// Parameters:  void
// Globals:     bool nreset, bool mod_clk_en, uint8_t dsel, bool mod_sel_ext_en,
//              uint8_t next_clk, bool read_ram, bool write_ram,
//              uint32_t ram_input_data[1024], uint32_t ram_output_data[1024],
//              static int send_ram_mode
// Return:      void
//****************************************************************************
void UART_handler(void);

//****************************************************************************
// Function:    vals_to_terminal
// Description: writes all ram output values to serial terminal
// Parameters:  void
// Globals:     uint32_t ram_output_data[1024]
// Return:      void
//****************************************************************************
void vals_to_terminal(void);
```

```
//****************************************************************************
// Function:     vals_from_terminal
// Description: reads all ram input values from serial terminal
// Parameters:   void
// Globals:      uint32_t ram_input_data[1024], static int send_ram_mode
// Return:       void
//****************************************************************************
void vals_from_terminal(void);

// Globals
extern bool nreset;
extern bool mod_clk_en;
extern uint8_t dsel;
extern bool mod_sel_ext_en;
extern uint8_t next_clk;
extern bool read_ram;
extern bool write_ram;
extern uint32_t ram_input_data[1024];
extern uint32_t ram_output_data[1024];

static int send_ram_mode = 0;

#endif /* UART0_H_ */
```

Quellcode B.12: uart0.c

```
/*
 * uart0.c
 *
 *  Created on: May 14, 2018
 *      Author: Jannes Helck
 */

#include "uart0.h"

void init_UART(void)
{
    int sysClock = 120000000;

    // enable the GPIOs for UART
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    // configure GPIOs in UART mode
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    // initialize UART to 8N1 with 19200 Bit/s
    UARTConfigSetExpClk(UART0_BASE, sysClock, 19200,
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                         UART_CONFIG_PAR_NONE));

    // enable UART interrupt.
    IntEnable(INT_UART0);
    IntPrioritySet(INT_UART0, 0x00);                    //set high int priority
    UARTIntRegister(UART0_BASE,UART_handler);
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);

}


// UART-interrupt-handler
void UART_handler(void)
{
   uint32_t ui32Status;
   int32_t rcv_char;
   char rcv_char2[10] = {};
   int a = 0;

   // get interrupt status.
   ui32Status = UARTIntStatus(UART0_BASE, true);

   // clear UART-interrupt.
   UARTIntClear(UART0_BASE, ui32Status);

   if(send_ram_mode==0){

       // while receive FIFO not empty
       while(UARTCharsAvail(UART0_BASE))
       {
           // read character from UART
           rcv_char = UARTCharGetNonBlocking(UART0_BASE);

           // concatinate chars
           if(a<= 10)
           {
               rcv_char2[a] = (unsigned char) rcv_char;
           }
```

```
        a +=1;
    }
    // compare string with commands
    if (strcmp (rcv_char2,"-reset") == 0)
    {
        if(nreset){
            nreset = 0;
            printf("nReset = 0\n");
        } else {
            nreset = 1;
            printf("nReset = 1\n");
        }
    }
    if (strcmp (rcv_char2,"-sread") == 0)
    {
        IntMasterDisable(); // disable all interrupts in NVIC
        vals_to_terminal();
        IntMasterEnable(); // enable all interrupts in NVIC
    }
    if (strcmp (rcv_char2,"-swrite") == 0)
    {
        send_ram_mode = 1;
        memset(rcv_char2,0,sizeof(rcv_char2));
    }
    if (strcmp (rcv_char2,"-read") == 0)
    {
        read_ram = 1;
    }
    if (strcmp (rcv_char2,"-write") == 0)
    {
        write_ram = 1;
    }
    if (strcmp (rcv_char2,"-clkon") == 0)
    {
        IntMasterDisable(); // disable all interrupts in NVIC
        mod_clk_en = 1;
        printf("Module clock enabled\n");
        SysCtlDelay(1000);
        IntMasterEnable(); // enable all interrupts in NVIC
    }
    if (strcmp (rcv_char2,"-clkoff") == 0)
    {
        mod_clk_en = 0;
        printf("Module clock disabled\n");
    }
    if (strcmp (rcv_char2,"-nxtclk") == 0)
    {
        next_clk = 1;
    }

    if (strcmp (rcv_char2,"-mselext") == 0)
    {
        mod_sel_ext_en = 1;
        printf("Enable external module selection\n");
    }
    if (strcmp (rcv_char2,"-mselint") == 0)
    {
        mod_sel_ext_en = 0;
        printf("Disable external module selection\n");
    }
    if (strcmp (rcv_char2,"-mod0") == 0)
    {
        dsel = 0;
        printf("Module 0 selected\n");
    }
```

```
            if (strcmp (rcv_char2,"-mod1") == 0)
            {
                dsel = 1;
                printf("Module 1 selected\n");
            }
            if (strcmp (rcv_char2,"-mod2") == 0)
            {
                dsel = 2;
                printf("Module 2 selected\n");
            }
            if (strcmp (rcv_char2,"-mod3") == 0)
            {
                dsel = 3;
                printf("Module 3 selected\n");
            }
            if (strcmp (rcv_char2,"-mod4") == 0)
            {
                dsel = 4;
                printf("Module 4 selected\n");
            }
            if (strcmp (rcv_char2,"-mod5") == 0)
            {
                dsel = 5;
                printf("Module 5 selected\n");
            }
            if (strcmp (rcv_char2,"-mod6") == 0)
            {
                dsel = 6;
                printf("Module 6 selected\n");
            }
            if (strcmp (rcv_char2,"-mod7") == 0)
            {
                dsel = 7;
                printf("Module 7 selected\n");
            }

        } else {
            IntMasterDisable(); // disable all interrupts in NVIC
            vals_from_terminal();
            IntMasterEnable(); // enable all interrupts in NVIC
        }

}


// sends data to the UART terminal
void vals_to_terminal(void)
{
    // buffer for printf-function
    char buffer[100];
    int i = 0;

    // send complete ram output data via UART
    for(i=0;i<1023;i++){
        sprintf(buffer,"%08X", ram_output_data[i]);
        strcat(buffer, "break");
        UART_send((uint8_t*)buffer,strlen(buffer));
    }
    sprintf(buffer,"%08X", ram_output_data[i]);
    UART_send((uint8_t*)buffer,strlen(buffer));

    printf("Data Transmission to terminal completed\n");

    SysCtlDelay(10);
}
```

```
void vals_from_terminal(void)
{
    int32_t rcv_char;
    char rcv_char2[8] = {};
    int a = 0;
    int i = 0;
    int val;
    static int addr = 0;
    char temp[2] = {0};

    while(UARTCharsAvail(UART0_BASE)){

        rcv_char = UARTCharGetNonBlocking(UART0_BASE);

        if(a <= 7)
        {
            rcv_char2[a] = (unsigned char) rcv_char;  // concatenate received chars
            a +=1;

            if (strcmp (rcv_char2,"-end") == 0) // terminate transmission
             {
                 send_ram_mode = 0;
                 addr = 0;
                 printf("Data transmission from terminal completed\n");
                 break;
             }
        }

        if(a > 7)
        {
            if(addr<1024){

                val = 0;

                for(i=0;i<8;i++){
                    temp[0] = rcv_char2[i];
                    // convert hex-char to uint32_t
                    val =  val + (((uint32_t) strtol(temp, NULL, 16)) << 4*(7-i));
                }
                ram_input_data[addr] = val; // store value
            }

            addr++;

            memset(rcv_char2,0,sizeof(rcv_char2)); // clear String
            a =0;
        }

    SysCtlDelay(10);
    }
}

void UART_send(const uint8_t *pui8Buffer, uint32_t ui32Count)
{
    // while characters to send
    while(ui32Count--)
    {
        UARTCharPut(UART0_BASE, *pui8Buffer++); //send char
    }
}
```

Quellcode B.13: parallel_interface.h

```
/*
 * main.h
 *
 *  Created on: Apr 29, 2018
 *      Author: Jannes Helck
 */

#ifndef PARALLEL_INTERFACE_H_
#define PARALLEL_INTERFACE_H_

#include <stdint.h>
#include <stdbool.h>
#include "tm4c1294ncpdt.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"

//*****************************************************************************
// Function:    init_parallel_interface
// Description: Initialization of GPIOs for interfacing the tested device.
// Parameters:  void
// Globals:     none
// Return:      void
//*****************************************************************************
void init_parallel_interface(void);

//*****************************************************************************
// Function:    read_parallel_interface
// Description: Reads the GPIOs of the parallel interface. The function writes
//              the GPIO values on the adresses of the arguments.
// Parameters:  uint8_t *mod_out, uint8_t *dout, bool *mod_rdy
// Globals:     none
// Return:      void
//*****************************************************************************
void read_parallel_interface(uint8_t*, uint8_t*, bool*);

//*****************************************************************************
// Function:    write_parallel_interface
// Description: Writes the GPIOs of the parallel interfaces.
// Parameters:  bool *nreset, bool *ctrl_ext_en, bool *mod_sel_ext_en,
//              uint8_t *dsel, bool *we, uint16_t *din
// Globals:     none
// Return:      void
//*****************************************************************************
void write_parallel_interface(bool*, bool*, bool*, uint8_t*, bool*, uint16_t*);

#endif /* PARALLEL_INTERFACE_H_ */
```

Quellcode B.14: parallel_interface.c

```
/*
 * main.c
 *
 *  Created on: Apr 29, 2018
 *      Author: Jannes Helck
 */

#include <parallel_interface.h>

void init_parallel_interface(){

    uint8_t wt = 0;

    //GPIO Clock enable
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);    //clock enable Port A
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);    //clock enable Port B
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);    //clock enable Port C
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);    //clock enable Port D
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);    //clock enable Port E
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);    //clock enable Port K
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOM);    //clock enable Port M
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);    //clock enable Port N
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOP);    //clock enable Port P
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOQ);    //clock enable Port Q

    wt++;

    //Unlock PD7, default NMI
    HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTD_BASE + GPIO_O_CR) = 0x80;

    wt++;

    // GPIO Inputs
    // PA4, PA5, PA6
    GPIOPinTypeGPIOInput(GPIO_PORTA_BASE, GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6);
    // PC7
    GPIOPinTypeGPIOInput(GPIO_PORTC_BASE, GPIO_PIN_7);
    // PD7
    GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, GPIO_PIN_7);
    // PE0, PE1, PE2, PE3
    GPIOPinTypeGPIOInput(GPIO_PORTE_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2
                         | GPIO_PIN_3);
    // PK3
    GPIOPinTypeGPIOInput(GPIO_PORTK_BASE, GPIO_PIN_3);
    // PM4, PM5
    GPIOPinTypeGPIOInput(GPIO_PORTM_BASE, GPIO_PIN_4 | GPIO_PIN_5);

    // GPIO Outputs
    // PB2, PB3, PB4, PB5
    GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4
                          | GPIO_PIN_5);
    // PC4, PC5, PC6
    GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6);
    // PD3, PD4, PD5
    GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5);
    // PE5
    GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_5);
    // PK0, PK1, PK2
    GPIOPinTypeGPIOOutput(GPIO_PORTK_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2);
    // PN4, PN5
    GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_4 | GPIO_PIN_5);
    // PP0, PP1, PP4
```

```
    GPIOPinTypeGPIOOutput(GPIO_PORTP_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_4);
    // PQ0
    GPIOPinTypeGPIOOutput(GPIO_PORTQ_BASE, GPIO_PIN_0);

    // GPIO Configuration
    // PA4, PA5, PA6
    GPIOPadConfigSet(GPIO_PORTA_BASE, GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6,
                     GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPD);
    // PB2, PB3, PB4, PB5
    GPIOPadConfigSet(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4
                     | GPIO_PIN_5, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPD);
    // PC3, PC4, PC6, PC7
    GPIOPadConfigSet(GPIO_PORTC_BASE, GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6
                     | GPIO_PIN_7, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPD);
    // PD3, PD4, PD5, PD7
    GPIOPadConfigSet(GPIO_PORTD_BASE, GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5
                     | GPIO_PIN_7, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPD);
    // PE0, PE1, PE2, PE3, PE5
    GPIOPadConfigSet(GPIO_PORTE_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2
                     | GPIO_PIN_3 | GPIO_PIN_5, GPIO_STRENGTH_2MA,
                     GPIO_PIN_TYPE_STD_WPD);
    // PK0, PK1, PK2, PK3
    GPIOPadConfigSet(GPIO_PORTK_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2
                     | GPIO_PIN_3, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPD);
    // PM4, PM5
    GPIOPadConfigSet(GPIO_PORTM_BASE, GPIO_PIN_4 | GPIO_PIN_5, GPIO_STRENGTH_2MA,
                     GPIO_PIN_TYPE_STD_WPD);
    // PN4, PN5
    GPIOPadConfigSet(GPIO_PORTN_BASE, GPIO_PIN_4 | GPIO_PIN_5, GPIO_STRENGTH_2MA,
                     GPIO_PIN_TYPE_STD_WPD);
    // PP0, PP1, PP4
    GPIOPadConfigSet(GPIO_PORTP_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_4,
                     GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPD);
    // PQ4
    GPIOPadConfigSet(GPIO_PORTQ_BASE, GPIO_PIN_0, GPIO_STRENGTH_2MA,
                     GPIO_PIN_TYPE_STD_WPD);

    // GPIO Initialize Outputs
    // PB2, PB3, PB4, PB5
    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4
                 | GPIO_PIN_5, 0);
    // PC4, PC5, PC6, PC7
    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6
                 | GPIO_PIN_7, 0);
    // PD3, PD4, PD5
    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5, 0);
    // PE5
    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_5, 0);
    // PK0, PK1, PK2
    GPIOPinWrite(GPIO_PORTK_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2, 0);
    // PN4, PN5
    GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_4 | GPIO_PIN_5, 0);
    // PP0, PP1, PP4
    GPIOPinWrite(GPIO_PORTP_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_4, 0);
    // PQ0
    GPIOPinWrite(GPIO_PORTQ_BASE, GPIO_PIN_0, 0);
}

void write_parallel_interface(bool *nreset, bool *ctrl_ext_en,
                 bool *mod_sel_ext_en, uint8_t *dsel, bool *we, uint16_t *din){

    // MOD_SEL_EXT_EN => PB3, din(8) => PB4, din(9) => PB5
    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4
                 | GPIO_PIN_5, (*mod_sel_ext_en<<3) | ((*din & 0x100)>>4)
                 | ((*din & 0x200)>>4));
```

```
        // 0 => PC4, we => PC5, nreset => PC6,
        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6,
                     (*we<<5) | (*nreset<<6));

        // ctrl_ext_en => PD3, din(2) => PD4, din(3) => PD5
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5,
                     (*ctrl_ext_en<<3) | ((*din & 0x4)<<2) | ((*din & 0x8)<<2));

        // dsel(0) => PK0, dsel(1) => PK1, dsel(2) => PK2
        GPIOPinWrite(GPIO_PORTK_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2,
                     (*dsel & 0x1) | (*dsel & 0x2) | (*dsel & 0x4));

        // din(7) => PN4, din(6) => PN5
        GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_4 | GPIO_PIN_5, ((*din & 0x80)>>3)
                     | ((*din & 0x40)>>1));

        // din(0) => PP0, din(1) => PP1, din(5) => PP4
        GPIOPinWrite(GPIO_PORTP_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_4,
                     (*din & 0x1) | (*din & 0x2) | (*din & 0x20)>>1);

        // din(4) => PQ0
        GPIOPinWrite(GPIO_PORTQ_BASE, GPIO_PIN_0, ((*din & 0x10)>>4));
    }

    void read_parallel_interface(uint8_t *mod_out, uint8_t *dout, bool *mod_rdy){

        *mod_out = (GPIOPinRead(GPIO_PORTK_BASE, GPIO_PIN_3)>>3);   // PK3 => mod_out(0)
        *mod_out += (GPIOPinRead(GPIO_PORTA_BASE, GPIO_PIN_4)>>3); // PA4 => mod_out(1)
        *mod_out += (GPIOPinRead(GPIO_PORTA_BASE, GPIO_PIN_5)>>3); // PA5 => mod_out(2)

        *dout = GPIOPinRead(GPIO_PORTE_BASE, GPIO_PIN_0);          // PE0 => dout(0)
        *dout += GPIOPinRead(GPIO_PORTE_BASE, GPIO_PIN_1);         // PE1 => dout(1)
        *dout += GPIOPinRead(GPIO_PORTE_BASE, GPIO_PIN_2);         // PE2 => dout(2)
        *dout += GPIOPinRead(GPIO_PORTE_BASE, GPIO_PIN_3);         // PE3 => dout(3)
        *dout += (GPIOPinRead(GPIO_PORTD_BASE, GPIO_PIN_7)>>3);    // PD7 => dout(4)
        *dout += (GPIOPinRead(GPIO_PORTA_BASE, GPIO_PIN_6)>>1);    // PA6 => dout(5)
        *dout += (GPIOPinRead(GPIO_PORTM_BASE, GPIO_PIN_4)<<2);    // PM4 => dout(6)
        *dout += (GPIOPinRead(GPIO_PORTM_BASE, GPIO_PIN_5)<<2);    // PM5 => dout(7)

        *mod_rdy = (GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_7)>>7);   // PC7 --> mod_rdy

    }
```

B.3 Matlab/Octave-Quellcode

Quellcode B.15: atan2_cordic.m

```
%------------------------------------------------------------------------
% Function:        atan2_cordic
%------------------------------------------------------------------------
% File name:       atan2_cordic.m
% Creation date:   2018-03-09
% Author:          Jannes Helck
% Version:         1.0
%
% Parameters:      Y_IN              : Input Y-Value
%                  X_IN              : Input X-Value
%
% Return:          A_OUT             : Returns the calculated Angle of
%                                      the input values

% Description:     This function uses the CORDIC-algorithm to calculate the
%                  the angle of two input values in fixed point arithmetik.
%------------------------------------------------------------------------

function [ A_OUT ] = atan2_cordic( Y_IN, X_IN )


WordLength = 12;
FractionLength = 9;
N = 9; % number of iterations

T = numerictype(1,WordLength,FractionLength);
F = fimath('SumWordLength', WordLength, 'SumFractionLength', FractionLength,
           'RoundingMethod','Floor');
% 'RoundingMethod','Nearest' for the calculation with different formats
%  and numbers of iterations

Y_IN = fi(Y_IN,'numerictype',T,'fimath', F);
X_IN = fi(X_IN,'numerictype',T,'fimath', F);
Y = abs(Y_IN);
X = abs(X_IN);

% formula for calculation of angle coefficients for the calculation
% with different formats and numbers of iterations
%ANGLES = fi(atan(2.^-(1:N)),'numerictype', T, 'fimath', F);

% calculated angle coefficients (rounded up)
ANGLES = fi(zeros(1,9),'NumericType', T, 'FiMath', F);
ANGLES(1) = fi((0.462890625),'NumericType', T, 'FiMath', F);
ANGLES(2) = fi((0.244140625),'NumericType', T, 'FiMath', F);
ANGLES(3) = fi((0.125),'NumericType', T, 'FiMath', F);
ANGLES(4) = fi((0.0625),'NumericType', T, 'FiMath', F);
ANGLES(5) = fi((0.03125),'NumericType', T, 'FiMath', F);
ANGLES(6) = fi((0.015625),'NumericType', T, 'FiMath', F);
ANGLES(7) = fi((0.0078125),'NumericType', T, 'FiMath', F);
ANGLES(8) = fi((0.00390625),'NumericType', T, 'FiMath', F);
ANGLES(9) = fi((0.001953125),'NumericType', T, 'FiMath', F);

% constants
PIDIV2 = fi(1.5703125,'numerictype', T, 'fimath', F);
PIDIV1 = fi(3.140625,'numerictype', T, 'fimath', F);
ZERO = fi(0,'numerictype', T, 'fimath', F);

% pre-allocate output
A = fi(0,'numerictype', T, 'fimath', F);
```

```
X_TEMP = fi(0, 'numerictype',T, 'fimath', F); % temporary register

if X < Y
    X_TEMP = X;
    X = Y;
    Y = X_TEMP;
end

% CORDIC
for i = 1:N
    if bitget(Y,WordLength) % negative angle
        A = A - ANGLES(i);

        X_TEMP = X - bitshift(Y,-i);
        Y = bitshift(X,-i) + Y;
        X = X_TEMP;
    else
        A = A + ANGLES(i);

        X_TEMP = X + bitshift(Y,-i);
        Y = -bitshift(X,-i) + Y;
        X = X_TEMP;
    end
end

if abs(X_IN) < abs(Y_IN)
     A = PIDIV2 - A;
end

if (X_IN < 0) && (Y_IN > 0) % Quadrant 2
    A = PIDIV1 - A;
end

if (X_IN < 0) && (Y_IN < 0) % Quadrant 3
    A = -PIDIV1 + A;
end

if (X_IN > 0) && (Y_IN < 0) % Quadrant 4
    A = -A;
end

if (X_IN == ZERO) && (Y_IN > ZERO)
    A = PIDIV2;
elseif (X_IN == ZERO) && (Y_IN < ZERO)
    A = -PIDIV2;
end

if (Y_IN == ZERO) && (X_IN > ZERO)
    A = 0;
elseif (Y_IN == ZERO) && (X_IN < ZERO)
    A = PIDIV1;
end

if (X_IN == ZERO) && (Y_IN == ZERO)
    A = 0;
end

A_OUT = A;
```

Quellcode B.16: atan2_test.m

```
%------------------------------------------------------------------------------
% script:          atan2_test
%------------------------------------------------------------------------------
% File name:       atan2_test.m
% Creation date:   2018-06-05
% Author:          Jannes Helck
% Version:         1.0
%
% Description:     Script for testing cordic algorithm. It enables control
%                  of the digital test system via serial communication.
%                  Avalailabe serial commands are listed below.
%------------------------------------------------------------------------------
% Serial comands:  Description:
%
% "-reset"         Reset the digital test system
% "-nxtclk"        Step one clock forward
% "-clkon"         Turn module clock on
% "-clkoff"        Turn module clock off
% "-write"         Write ram input data from MC to RAM
% "-read"          Read RAM and write ram output data to MC
% "-mselext"       External module selection
% "-mselint"       Internal module selection
% "-mod<x>"        Select module <x> (0-7)
%------------------------------------------------------------------------------

pkg load instrument-control

try
  s1 = serial("/dev/ttyACM0", 19200, 1)
catch
end
try
  s1 = serial("/dev/ttyACM1", 19200, 1)
catch
end

% Generate ram input data
ram_input_data = calculate_input_data(DAT(1).COS_SIG_f2,DAT(1).SIN_SIG_f2);

% Send ram input dada to MC
serial_write_data(s1, ram_input_data);

pause(0.5);

% Write ram input Data into RAM
srl_write(s1, "-write");

pause(3);

srl_write(s1, "-mselext");  % external module selection
pause(0.5);
srl_write(s1, "-mod4");     % select module 4
pause(0.5);

srl_write(s1, "-reset");    % toggle Reset
pause(0.5);
srl_write(s1, "-reset");
pause(0.5);


srl_write(s1, "-clkon");    % enable module clock

pause(3);                   % wait for calculation
```

```
65  srl_write(s1, "-clkoff");    % disable module clock

    pause(0.1);

    % read ram output Data into MC
70  srl_write(s1, "-read");

    pause(4);

    % fetch ram output data from MC
75  ram_output_data = serial_read_data(s1);

    pause(1);

    % calculate und display angle
80  angle = calculate_angle(ram_output_data(2));



    try
85    close(s1);
    catch
    end
```

Quellcode B.17: serial_write_data.m

```
%-------------------------------------------------------------------------------
% Function:        serial_write_data
%-------------------------------------------------------------------------------
% File name:       serial_write_data.m
% Creation date:   2018-06-07
% Author:          Jannes Helck
% Version:         1.0
%
% Parameters:      s1                 : serial interface
%                  ram_input_data     : Input data for RAM
%
% Return:          none
%
% Description:     This function takes the precalculated input data array
%                  for the RAM and sends it to MC via serial interface.
%-------------------------------------------------------------------------------

function [] = serial_write_data (s1, ram_input_data)

  pkg load instrument-control

  try
      % flush input and output buffers
      srl_flush(s1);

      srl_write(s1, "-swrite");
      pause(0.4)
      for n = 1:length(ram_input_data)
        temp = char(ram_input_data(n));
        srl_write(s1, temp);
        pause(0.1)
      end
      srl_write(s1, "-end");
      pause(0.1)
  catch
  end

endfunction
```

Quellcode B.18: serial_read_data.m

```
%-------------------------------------------------------------------------------
% Function:        serial_read_data
%-------------------------------------------------------------------------------
% File name:       serial_read_data.m
% Creation date:   2018-06-07
% Author:          Jannes Helck
% Version:         1.0
%
% Parameters:      s1                  : serial interface
%
% Return:          ram_output_data     : RAM data received via serial interface
%
% Description:     This function reads the RAM data stored on MC via serial
%                  interface and returns them as cell array.
%-------------------------------------------------------------------------------

function [ram_output_data] = serial_read_data(s1)

  try
    % flush input and output buffers
    srl_flush(s1);
    pause(0.1)

    srl_write(s1, "-sread");
    pause(0.1)

    data = srl_read(s1,18000);
    pause(0.1)

    data_str = char(data);
    ram_output_data = strsplit(data_str,'break');
  catch
  end

endfunction
```

Quellcode B.19: calculate_input_data.m

```
%------------------------------------------------------------------------
% Function:       calculate_input_data
%------------------------------------------------------------------------
% File name:      calculate_input_data.m
% Creation date:  2018-06-07
% Author:         Jannes Helck
% Version:        1.0
%
% Parameters:     cos_in            : matrix of cosinus values
%                 sin_in            : matrix of sinus values
%
% Return:         ram_input_data    : calculated data for RAM input
%
% Description:    This function takes a cosinus and sinus matrix of generic
%                 size and converts the values from double to s1Q10 format.
%                 The values are stored in a cell array. Each cell contains
%                 the concatenated string: cos + "0000" + sin + "0000".
%------------------------------------------------------------------------

function [ram_input_data] = calculate_input_data (cos_in, sin_in)

  data = "";

  [rows, columns] = size(cos_in);
  m_max = rows*columns;

  for k = 1:rows
    for m = 1:columns

      cos = cos_in(k, m);
      sin = sin_in(k, m);
      cos_hex = "";
      sin_hex = "";


      if (cos < 0)                        % signed
        cos_bin = '1';
        cos = cos + 2;
      else
        cos_bin = '0';
      endif;

      for n = 0:10                        % convert cos-value
        if (cos >= (2^-n))                % in binary string
          cos_bin = strcat(cos_bin, '1');
          cos = cos - (2^-n);
        else
          cos_bin = strcat(cos_bin, '0');
        endif;
      endfor

      for n = 0:2                         % convert binary cos-string
        switch(cos_bin(1+(4*n):4+(4*n)))  % in hex-string
          case "0000" cos_hex = strcat(cos_hex, '0');
          case "0001" cos_hex = strcat(cos_hex, '1');
          case "0010" cos_hex = strcat(cos_hex, '2');
          case "0011" cos_hex = strcat(cos_hex, '3');
          case "0100" cos_hex = strcat(cos_hex, '4');
          case "0101" cos_hex = strcat(cos_hex, '5');
          case "0110" cos_hex = strcat(cos_hex, '6');
          case "0111" cos_hex = strcat(cos_hex, '7');
          case "1000" cos_hex = strcat(cos_hex, '8');
          case "1001" cos_hex = strcat(cos_hex, '9');
```

```
          case "1010" cos_hex = strcat(cos_hex, 'A');
          case "1011" cos_hex = strcat(cos_hex, 'B');
          case "1100" cos_hex = strcat(cos_hex, 'C');
          case "1101" cos_hex = strcat(cos_hex, 'D');
          case "1110" cos_hex = strcat(cos_hex, 'E');
          case "1111" cos_hex = strcat(cos_hex, 'F');
          otherwise disp("Error: illegal character");
        endswitch
      endfor


      if (sin < 0)                      % signed
        sin_bin = '1';
        sin = sin + 2;
      else
        sin_bin = '0';
      endif;

      for n = 0:10                      % convert binary sin-string
        if (sin >= (2^-n))              % in hex-string
          sin_bin = strcat(sin_bin, '1');
          sin = sin - (2^-n);
        else
          sin_bin = strcat(sin_bin, '0');
        endif;
      endfor

      for n = 0:2
        switch(sin_bin(1+(4*n):4+(4*n)))
          case "0000" sin_hex = strcat(sin_hex, '0');
          case "0001" sin_hex = strcat(sin_hex, '1');
          case "0010" sin_hex = strcat(sin_hex, '2');
          case "0011" sin_hex = strcat(sin_hex, '3');
          case "0100" sin_hex = strcat(sin_hex, '4');
          case "0101" sin_hex = strcat(sin_hex, '5');
          case "0110" sin_hex = strcat(sin_hex, '6');
          case "0111" sin_hex = strcat(sin_hex, '7');
          case "1000" sin_hex = strcat(sin_hex, '8');
          case "1001" sin_hex = strcat(sin_hex, '9');
          case "1010" sin_hex = strcat(sin_hex, 'A');
          case "1011" sin_hex = strcat(sin_hex, 'B');
          case "1100" sin_hex = strcat(sin_hex, 'C');
          case "1101" sin_hex = strcat(sin_hex, 'D');
          case "1110" sin_hex = strcat(sin_hex, 'E');
          case "1111" sin_hex = strcat(sin_hex, 'F');
          otherwise disp("Error: illegal character");
        endswitch
      endfor

      % arrenge values to RAM-words, concatenate words to string
      if ((k<rows) || (m<columns))
        data = strcat(data, cos_hex, '0', sin_hex, '0', "break");
      else
        data = strcat(data, cos_hex, '0', sin_hex, '0');
      endif

    endfor
  endfor

  ram_input_data = strsplit(data, 'break');  % split concatenated string
                                             % into cell-array
endfunction
```

Quellcode B.20: calculate_output_data.m

```
%---------------------------------------------------------------------------------
% Function:         calculate_output_data
%---------------------------------------------------------------------------------
% File name:        calculate_output_data.m
% Creation date:    2018-06-08
% Author:           Jannes Helck
% Version:          1.0
%
% Parameters:       ram_output_data  :  Full RAM output data
%                   start_addr       :  Start adress for calculation
%                   end_addr         :  End adress for calculation
%
% Return:           cos_out          :  Calculated cosinus values
%                   sin_out          :  Calculated sinus values
%
% Description:      This function takes the RAM output data from start to
%                   end adress and converts the cos and sin values stored in
%                   s1Q10 Format to double.
%---------------------------------------------------------------------------------

function [cos_out, sin_out] = calculate_output_data (ram_output_data, ...
   start_addr, end_addr)

  cos_data = "";
  sin_data = "";

  for n = start_addr:end_addr

    cos = 0;
    sin = 0;

    data = char(ram_output_data(n));

    switch(data(1:1))                   % decode 1st hex-char of cos-value
      case '0' cos = cos + 0;
      case '1' cos = cos + 0.25;
      case '2' cos = cos + 0.5;
      case '3' cos = cos + 0.75;
      case '4' cos = cos + 1;
      case '5' cos = cos + 1.25;
      case '6' cos = cos + 1.5;
      case '7' cos = cos + 1.75;
      case '8' cos = cos - 2;
      case '9' cos = cos - 1.75;
      case 'A' cos = cos - 1.5;
      case 'B' cos = cos - 1.25;
      case 'C' cos = cos - 1;
      case 'D' cos = cos - 0.75;
      case 'E' cos = cos - 0.5;
      case 'F' cos = cos - 0.25;
      otherwise disp("Error: illegal character in transmitted data");
    endswitch

    switch(data(2:2))                   % decode 2nd hex-char of cos-value
      case '0' cos = cos + 0;
      case '1' cos = cos + 0.015625;
      case '2' cos = cos + 0.03125;
      case '3' cos = cos + 0.046875;
      case '4' cos = cos + 0.0625;
      case '5' cos = cos + 0.078125;
      case '6' cos = cos + 0.09375;
      case '7' cos = cos + 0.109375;
      case '8' cos = cos + 0.125;
```

```
        case '9' cos = cos + 0.140625;
65      case 'A' cos = cos + 0.15625;
        case 'B' cos = cos + 0.171875;
        case 'C' cos = cos + 0.1875;
        case 'D' cos = cos + 0.203125;
        case 'E' cos = cos + 0.21875;
70      case 'F' cos = cos + 0.234375;
        otherwise disp("Error: illegal character in transmitted data");
      endswitch

      switch(data(3:3))                 % decode 3rd hex-char of cos-value
75      case '0' cos = cos + 0;
        case '1' cos = cos + 0.0009765625;
        case '2' cos = cos + 0.001953125;
        case '3' cos = cos + 0.0029296875;
        case '4' cos = cos + 0.00390625;
80      case '5' cos = cos + 0.0048828125;
        case '6' cos = cos + 0.005859375;
        case '7' cos = cos + 0.0068359375;
        case '8' cos = cos + 0.0078125;
        case '9' cos = cos + 0.0087890625;
85      case 'A' cos = cos + 0.009765625;
        case 'B' cos = cos + 0.0107421875;
        case 'C' cos = cos + 0.01171875;
        case 'D' cos = cos + 0.0126953125;
        case 'E' cos = cos + 0.013671875;
90      case 'F' cos = cos + 0.0146484375;
        otherwise disp("Error: illegal character in transmitted data");
      endswitch

      switch(data(5:5))                 % decode 1st hex-char of sin-value
95      case '0' sin = sin + 0;
        case '1' sin = sin + 0.25;
        case '2' sin = sin + 0.5;
        case '3' sin = sin + 0.75;
        case '4' sin = sin + 1;
100     case '5' sin = sin + 1.25;
        case '6' sin = sin + 1.5;
        case '7' sin = sin + 1.75;
        case '8' sin = sin - 2;
        case '9' sin = sin - 1.75;
105     case 'A' sin = sin - 1.5;
        case 'B' sin = sin - 1.25;
        case 'C' sin = sin - 1;
        case 'D' sin = sin - 0.75;
        case 'E' sin = sin - 0.5;
110     case 'F' sin = sin - 0.25;
        otherwise disp("Error: illegal character in transmitted data");
      endswitch

      switch(data(6:6))                 % decode 2nd hex-char sin-value
115     case '0' sin = sin + 0;
        case '1' sin = sin + 0.015625;
        case '2' sin = sin + 0.03125;
        case '3' sin = sin + 0.046875;
        case '4' sin = sin + 0.0625;
120     case '5' sin = sin + 0.078125;
        case '6' sin = sin + 0.09375;
        case '7' sin = sin + 0.109375;
        case '8' sin = sin + 0.125;
        case '9' sin = sin + 0.140625;
125     case 'A' sin = sin + 0.15625;
        case 'B' sin = sin + 0.171875;
        case 'C' sin = sin + 0.1875;
        case 'D' sin = sin + 0.203125;
```

```
          case 'E' sin = sin + 0.21875;
130       case 'F' sin = sin + 0.234375;
          otherwise disp("Error: illegal character in transmitted data");
        endswitch

        switch(data(7:7))                    % decode 3rd hex-char sin-value
135       case '0' sin = sin + 0;
          case '1' sin = sin + 0.0009765625;
          case '2' sin = sin + 0.001953125;
          case '3' sin = sin + 0.0029296875;
          case '4' sin = sin + 0.00390625;
140       case '5' sin = sin + 0.0048828125;
          case '6' sin = sin + 0.005859375;
          case '7' sin = sin + 0.0068359375;
          case '8' sin = sin + 0.0078125;
          case '9' sin = sin + 0.0087890625;
145       case 'A' sin = sin + 0.009765625;
          case 'B' sin = sin + 0.0107421875;
          case 'C' sin = sin + 0.01171875;
          case 'D' sin = sin + 0.0126953125;
          case 'E' sin = sin + 0.013671875;
150       case 'F' sin = sin + 0.0146484375;
          otherwise disp("Error: illegal character in transmitted data");
        endswitch

        cos_str = num2str(cos);     % convert cos-value to string
155     sin_str = num2str(sin);     % convert sin-value to string

        if (n<end_addr)
          cos_data = strcat(cos_data, cos_str, "break");    % concatenate strings
          sin_data = strcat(sin_data, sin_str, "break");
160     else
          cos_data = strcat(cos_data, cos_str);
          sin_data = strcat(sin_data, sin_str);
        endif

165   endfor

      cos_out_str = strsplit(cos_data, 'break');    % split strings into cell-arrays
      sin_out_str = strsplit(sin_data, 'break');

170   cos_out = str2double(cos_out_str);    % convert output cell-arrays
      sin_out = str2double(sin_out_str);    % to double-arrays

    endfunction
```

Quellcode B.21: calculate_angle.m

```
%-------------------------------------------------------------------------
% Function:       calculate_angle
%-------------------------------------------------------------------------
% File name:      calculate_angle.m
% Creation date:  2018-06-07
% Author:         Jannes Helck
% Version:        1.0
%
% Parameters:     ram_data     : RAM element containing angle information
%
% Return:         angle        : Calculated angle
%
% Description:    This function takes an Ram element containing the
%                 angle information calculated by cordic algorithm which
%                 is in s2Q9 format and converts it into double and
%                 returns it.
%-------------------------------------------------------------------------

function [angle] = calculate_angle (ram_data)

  angle = 0;
  data = char(ram_data);

switch(data(1:1))                 % decode 1st hex-char
  case '0' angle = angle + 0;
  case '1' angle = angle + 0.5;
  case '2' angle = angle + 1;
  case '3' angle = angle + 1.5;
  case '4' angle = angle + 2;
  case '5' angle = angle + 2.5;
  case '6' angle = angle + 3;
  case '7' angle = angle + 3.5;
  case '8' angle = angle - 4;
  case '9' angle = angle - 3.5;
  case 'A' angle = angle - 3;
  case 'B' angle = angle - 2.5;
  case 'C' angle = angle - 2;
  case 'D' angle = angle - 1.5;
  case 'E' angle = angle - 1;
  case 'F' angle = angle - 0.5;
  otherwise disp("Error: illegal character in transmitted data");
endswitch

switch(data(2:2))                 % decode 2nd hex-char
  case '0' angle = angle + 0;
  case '1' angle = angle + 0.03125;
  case '2' angle = angle + 0.0626;
  case '3' angle = angle + 0.09375;
  case '4' angle = angle + 0.125;
  case '5' angle = angle + 0.15625;
  case '6' angle = angle + 0.1875;
  case '7' angle = angle + 0.21875;
  case '8' angle = angle + 0.25;
  case '9' angle = angle + 0.28125;
  case 'A' angle = angle + 0.3125;
  case 'B' angle = angle + 0.34375;
  case 'C' angle = angle + 0.375;
  case 'D' angle = angle + 0.40625;
  case 'E' angle = angle + 0.4375;
  case 'F' angle = angle + 0.46875;
  otherwise disp("Error: illegal character in transmitted data");
endswitch
```

```
switch(data(3:3))                    % decode 3rd hex-char
  case '0' angle = angle + 0;
  case '1' angle = angle + 0.001953125;
  case '2' angle = angle + 0.00390625;
  case '3' angle = angle + 0.005859375;
  case '4' angle = angle + 0.0078125;
  case '5' angle = angle + 0.009765625;
  case '6' angle = angle + 0.01171875;
  case '7' angle = angle + 0.013671875;
  case '8' angle = angle + 0.015625;
  case '9' angle = angle + 0.017578125;
  case 'A' angle = angle + 0.01953125;
  case 'B' angle = angle + 0.021484375;
  case 'C' angle = angle + 0.0234375;
  case 'D' angle = angle + 0.025390625;
  case 'E' angle = angle + 0.02734375;
  case 'F' angle = angle + 0.029296875;
  otherwise disp("Error: illegal character in transmitted data");
endswitch

x = 0:1:1;
y = [0 angle];

figure(4);                           % display calculated angle
polar (y, x, '-');
hlines = findall(gcf,'Type','line');

for i = 1:length(hlines)
    set(hlines(i),'LineWidth',2);
end
title ("Berechneter Winkel");


endfunction
```

C Cadence-Synthesereports

Quellcode C.1: ATAN2_syn_cell.rep

```
============================================================
  Generated by:            Genus(TM) Synthesis Solution 17.11-s014_1
  Generated on:            Aug 03 2018  02:27:11 pm
  Module:                  ATAN2
  Technology libraries:    c35_CORELIB_TYP 3.02
                           physical_cells
  Operating conditions:    _nominal_
  Interconnect mode:       global
  Area mode:               physical library
============================================================


  Gate    Instances     Area         Library
---------------------------------------------------
ADD22            8    1164.800    c35_CORELIB_TYP
ADD32           21    5733.000    c35_CORELIB_TYP
AOI211          44    3203.200    c35_CORELIB_TYP
AOI2111         31    2821.000    c35_CORELIB_TYP
AOI221          85    7735.000    c35_CORELIB_TYP
AOI311           4     364.000    c35_CORELIB_TYP
BUFE2           43    6260.800    c35_CORELIB_TYP
CLKIN3          23     837.200    c35_CORELIB_TYP
DF3             40   10920.000    c35_CORELIB_TYP
DFC3            10    3094.000    c35_CORELIB_TYP
DFE1            47   15397.200    c35_CORELIB_TYP
DFE3             1     345.800    c35_CORELIB_TYP
IMAJ31           1     109.200    c35_CORELIB_TYP
IMUX21          39    3549.000    c35_CORELIB_TYP
IMUX30           1     182.000    c35_CORELIB_TYP
INV2            71    2584.400    c35_CORELIB_TYP
INV3            77    2802.800    c35_CORELIB_TYP
JK3              1     345.800    c35_CORELIB_TYP
MAJ31            2     218.400    c35_CORELIB_TYP
MUX22            9     982.800    c35_CORELIB_TYP
NAND22         150    8190.000    c35_CORELIB_TYP
NAND31          13     946.400    c35_CORELIB_TYP
NAND41           4     364.000    c35_CORELIB_TYP
NOR21          126    6879.600    c35_CORELIB_TYP
NOR22            2     145.600    c35_CORELIB_TYP
NOR23            1      91.000    c35_CORELIB_TYP
NOR31            5     364.000    c35_CORELIB_TYP
NOR40            2     145.600    c35_CORELIB_TYP
OAI2111         39    3549.000    c35_CORELIB_TYP
OAI212          50    3640.000    c35_CORELIB_TYP
OAI222          53    4823.000    c35_CORELIB_TYP
OAI311          10     910.000    c35_CORELIB_TYP
XNR21           61    6661.200    c35_CORELIB_TYP
XNR31            1     200.200    c35_CORELIB_TYP
XOR21            2     254.800    c35_CORELIB_TYP
---------------------------------------------------
total         1077  105814.800
```

```
    Type        Instances      Area      Area %
--------------------------------------------------
sequential              99   30102.800    28.4
inverter               171    6224.400     5.9
tristate                43    6260.800     5.9
logic                  764   63226.800    59.8
physical_cells           0       0.000     0.0
--------------------------------------------------
total                 1077  105814.800   100.0
```

Quellcode C.2: ATAN2_syn_power.rep

```
==========================================================================
  Generated by:             Genus(TM) Synthesis Solution 17.11-s014_1
  Generated on:             Aug 03 2018  02:27:11 pm
  Module:                   ATAN2
  Technology libraries:     c35_CORELIB_TYP 3.02
                            physical_cells
  Operating conditions:     _nominal_
  Interconnect mode:        global
  Area mode:                physical library
==========================================================================

                   Leakage     Dynamic       Total
Instance  Cells  Power(nW)    Power(nW)     Power(nW)
-------------------------------------------------------------
ATAN2      1077      0.859  2541820.332  2541821.191
```

Quellcode C.3: ATAN2_syn_timing.rep

```
==========================================================================
  Generated by:             Genus(TM) Synthesis Solution 17.11-s014_1
  Generated on:             Aug 03 2018  02:27:11 pm
  Module:                   ATAN2
  Technology libraries:     c35_CORELIB_TYP 3.02
                            physical_cells
  Operating conditions:     _nominal_
  Interconnect mode:        global
  Area mode:                physical library
==========================================================================

           Pin                    Type      Fanout  Load  Slew  Delay  Arrival
                                                    (fF)  (ps)  (ps)    (ps)
------------------------------------------------------------------------------------
(clock clk)                     launch                                     0 R
STATE_reg[1]/C                                             400             0 R
STATE_reg[1]/QN                 DFC3           22  528.1  2342  +1751   1751 R
g25661__6083/B                                                    +0    1752
g25661__6083/Q                  NAND22         15  353.7  1293  +618    2370 F
g25628__8780/B                                                    +0    2370
g25628__8780/Q                  NOR21           7  175.6  2331  +1141   3511 R
g25597__8780/A                                                    +0    3511
g25597__8780/Q                  NAND31          1   31.8   719  +166    3678 F
g25561__7114/D                                                    +0    3678
g25561__7114/Q                  OAI2111         1   35.8   942  +526    4204 R
sub004712_Y_sub004774_Y_...                                       +0    4205
sub004712_Y_sub004774_Y_...     XNR21           1   51.8   501  +561    4766 F
sub004712_Y_sub004774_Y_...                                       +1    4766
sub004712_Y_sub004774_Y_...     ADD32           1   51.8   338  +444    5210 F
```

```
sub004712_Y_sub004774_Y_...                                         +1      5211
sub004712_Y_sub004774_Y_...    ADD32          1   51.8   336     +418      5628  F
sub004712_Y_sub004774_Y_...                                         +1      5629
sub004712_Y_sub004774_Y_...    ADD32          1   51.8   336     +417      6046  F
sub004712_Y_sub004774_Y_...                                         +1      6047
sub004712_Y_sub004774_Y_...    ADD32          1   51.8   336     +417      6464  F
sub004712_Y_sub004774_Y_...                                         +1      6465
sub004712_Y_sub004774_Y_...    ADD32          1   51.8   336     +417      6882  F
sub004712_Y_sub004774_Y_...                                         +1      6883
sub004712_Y_sub004774_Y_...    ADD32          1   51.8   336     +417      7300  F
sub004712_Y_sub004774_Y_...                                         +1      7301
sub004712_Y_sub004774_Y_...    ADD32          1   51.8   336     +417      7718  F
sub004712_Y_sub004774_Y_...                                         +1      7719
sub004712_Y_sub004774_Y_...    ADD32          1   51.8   336     +417      8136  F
sub004712_Y_sub004774_Y_...                                         +1      8136
sub004712_Y_sub004774_Y_...    ADD32          1   51.8   336     +417      8554  F
sub004712_Y_sub004774_Y_...                                         +1      8554
sub004712_Y_sub004774_Y_...    ADD32          1   51.8   336     +417      8972  F
sub004712_Y_sub004774_Y_...                                         +1      8972
sub004712_Y_sub004774_Y_...    ADD32          1   35.8   282     +383      9355  F
g36486/B                                                            +0      9355
g36486/Q                       XNR21          1   29.8   556     +367      9722  R
g35852__2391/A                                                      +0      9723
g35852__2391/Q                 AOI221         1   36.8   860     +253      9976  F
g35847/A                                                            +0      9976
g35847/Q                       INV3           1   27.8   316     +205     10182  R
Y_reg[11]/D                    DF3                                  +0     10182
Y_reg[11]/C                    setup                     400       +11     10193  R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clk)                    capture                                     31250  F
------------------------------------------------------------------------------------
Timing slack :    21057ps
Start-point  : STATE_reg[1]/C
End-point    : Y_reg[11]/D
```

Quellcode C.4: MEMORY_CONTROL_syn_cell.rep

```
================================================================
  Generated by:            Genus(TM) Synthesis Solution 17.11-s014_1
  Generated on:            Aug 03 2018  02:09:40 pm
  Module:                  MEMORY_CONTROL
  Technology libraries:    c35_CORELIB_TYP 3.02
                           physical_cells
  Operating conditions:    _nominal_
  Interconnect mode:       global
  Area mode:               physical library
================================================================


 Gate   Instances    Area          Library
---------------------------------------------------
AOI211          3     218.400    c35_CORELIB_TYP
AOI221         80    7280.000    c35_CORELIB_TYP
BUFE2           3     436.800    c35_CORELIB_TYP
BUFT2          42    6115.200    c35_CORELIB_TYP
CLKIN3         37    1346.800    c35_CORELIB_TYP
DFC3           74   22895.600    c35_CORELIB_TYP
DFEC1          10    3458.000    c35_CORELIB_TYP
INV3           38    1383.200    c35_CORELIB_TYP
INV6            1      54.600    c35_CORELIB_TYP
NAND22         15     819.000    c35_CORELIB_TYP
NAND23          1      91.000    c35_CORELIB_TYP
NOR21          13     709.800    c35_CORELIB_TYP
NOR23           1      91.000    c35_CORELIB_TYP
NOR31           1      72.800    c35_CORELIB_TYP
OAI212          2     145.600    c35_CORELIB_TYP
---------------------------------------------------
total         321   45117.800


      Type       Instances      Area    Area %
---------------------------------------------
sequential              84 26353.600    58.4
inverter                76  2784.600     6.2
tristate                45  6552.000    14.5
logic                  116  9427.600    20.9
physical_cells           0     0.000     0.0
---------------------------------------------
total                  321 45117.800   100.0
```

Quellcode C.5: MEMORY_CONTROL_syn_power.rep

```
================================================================
  Generated by:            Genus(TM) Synthesis Solution 17.11-s014_1
  Generated on:            Aug 03 2018  02:09:40 pm
  Module:                  MEMORY_CONTROL
  Technology libraries:    c35_CORELIB_TYP 3.02
                           physical_cells
  Operating conditions:    _nominal_
  Interconnect mode:       global
  Area mode:               physical library
================================================================

                          Leakage     Dynamic        Total
   Instance       Cells  Power(nW)   Power(nW)     Power(nW)
-------------------------------------------------------------
MEMORY_CONTROL     321       0.381  1434381.644  1434382.025
```

Quellcode C.6: MEMORY_CONTROL_syn_timing.rep

```
============================================================================
  Generated by:            Genus(TM) Synthesis Solution 17.11-s014_1
  Generated on:            Aug 03 2018  02:09:40 pm
  Module:                  MEMORY_CONTROL
  Technology libraries:    c35_CORELIB_TYP 3.02
                           physical_cells
  Operating conditions:    _nominal_
  Interconnect mode:       global
  Area mode:               physical library
============================================================================

     Pin              Type       Fanout  Load  Slew  Delay  Arrival
                                         (fF)  (ps)  (ps)    (ps)
--------------------------------------------------------------------------------
(clock clk)          launch                                     0 R
(in_del_1)           ext delay                        +250    250 F
CTRL_EXT_EN          in port          4 122.7     0     +0    250 F
g4288/A                                                 +0    250
g4288/Q              INV6            46 960.3  2108   +869   1119 R
g4278__7118/B                                           +0   1119
g4278__7118/Q        NOR21            2  47.5   761   +414   1533 F
g4273__5795/A                                           +0   1534
g4273__5795/Q        NAND22           4  93.7   755   +438   1972 R
g4251__2391/B                                           +0   1972
g4251__2391/Q        NOR21            8 160.3  1325   +697   2669 F
g4195__5019/C                                           +0   2670
g4195__5019/Q        AOI221           1  36.8   766   +487   3156 R
g4128__3772/B                                           +0   3157
g4128__3772/Q        NAND22           1  27.8   270    +73   3230 F
DOUT_reg[2]/D <<<    DFC3                               +0   3230
DOUT_reg[2]/C        setup                      400     +1   3231 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clk)          capture                                31250 F
--------------------------------------------------------------------------------
Timing slack :   28019ps
Start-point  : CTRL_EXT_EN
End-point    : DOUT_reg[2]/D
```

Quellcode C.7: MODULE_CONTROL_syn_cell.rep

```
===========================================================================
  Generated by:          Genus(TM) Synthesis Solution 17.11-s014_1
  Generated on:          Aug 03 2018  02:33:23 pm
  Module:                MODULE_CONTROL
  Technology libraries:  c35_CORELIB_TYP 3.02
                         physical_cells
  Operating conditions:  _nominal_
  Interconnect mode:     global
  Area mode:             physical library
===========================================================================


 Gate   Instances    Area          Library
---------------------------------------------
ADD22           2   291.200    c35_CORELIB_TYP
BUFT2           3   436.800    c35_CORELIB_TYP
DFC3            1   309.400    c35_CORELIB_TYP
JKC3            1   364.000    c35_CORELIB_TYP
---------------------------------------------
total           7  1401.400


     Type       Instances   Area    Area %
-----------------------------------------
sequential              2  673.400   48.1
tristate                3  436.800   31.2
logic                   2  291.200   20.8
physical_cells          0    0.000    0.0
-----------------------------------------
total                   7 1401.400  100.0
```

Quellcode C.8: MODULE_CONTROL_syn_power.rep

```
===========================================================================
  Generated by:          Genus(TM) Synthesis Solution 17.11-s014_1
  Generated on:          Aug 03 2018  02:33:23 pm
  Module:                MODULE_CONTROL
  Technology libraries:  c35_CORELIB_TYP 3.02
                         physical_cells
  Operating conditions:  _nominal_
  Interconnect mode:     global
  Area mode:             physical library
===========================================================================

                       Leakage    Dynamic     Total
   Instance     Cells Power(nW) Power(nW)  Power(nW)
---------------------------------------------------
MODULE_CONTROL      7     0.011 37221.930 37221.941
```

Quellcode C.9: MODULE_CONTROL_syn_timing.rep

```
============================================================
  Generated by:            Genus(TM) Synthesis Solution 17.11-s014_1
  Generated on:            Aug 03 2018  02:33:23 pm
  Module:                  MODULE_CONTROL
  Technology libraries:    c35_CORELIB_TYP 3.02
                           physical_cells
  Operating conditions:    _nominal_
  Interconnect mode:       global
  Area mode:               physical library
============================================================

     Pin                Type        Fanout Load Slew Delay Arrival
                                           (fF) (ps)  (ps)   (ps)
-------------------------------------------------------------------
(clock clk)           launch                                  0 R
STATE_reg[1]/C                                   400           0 R
STATE_reg[1]/Q        JKC3               1 41.8  242  +764   764 F
g206__3772/B                                           +0    765
g206__3772/S          ADD22              1 27.8  286  +477  1242 R
g6__4296/A                                             +0   1242
g6__4296/Q            BUFT2              1 29.8  276  +274  1516 R
MOD_SEL[1]      <<<   interconnect               276    +0  1517 R
                      out port                          +0  1517 R
(ou_del_1)            ext delay                       +250  1767 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clk)           capture                              62500 R
-------------------------------------------------------------------
Timing slack :   60733ps
Start-point  : STATE_reg[1]/C
End-point    : MOD_SEL[1]
```

D Kurzbeschreibung zur Nutzung der Xilinx-Umgebung im Zusammenspiel mit der Cadence-Umgebung

Im folgenden wird beschrieben, wie die Vivado Design Suite der Firma Xilinx in dieser Arbeit zum Testen des entwickelten VHDL-Codes auf dem ZedBoard verwendet wird. Die Simulation des entwickelten VHDL-Code erfolgt im Vorfeld der Nutzung von Vivado mit dem Cadence-Tool NC-Sim. Zwar umfasst die Vivado-Umgebung auch einen VHDL-Simulatior, das Cadence-Tool NC-Sim ist jedoch im ISAR-Projekt weiter verbreitet und wird als sehr guter VHDL-Simulator erachtet. Nach der Simulation sollte der VHDL-Code im wesentlichen von syntaktischen Fehlern korrigiert sein. Es kann jedoch auch bei scheinbar problemloser Simulation zu Fehlern bei der VHDL-Synthese bzw. Implemenation in Vivado kommen. Im folgenden werden die Schritte von der Erstellung eines Vivado-Projekts bis zum Testen des Programms auf dem FPGA beschrieben:

1. Nach dem erstmaligen Starten der Vivado-Umgebung wird ein neues Projekt erstellt. Durch Betätigen der Schaltfläche `Create Projekt` im Startfenster öffnet sich der ensprechende Wizard. In diesem sind Projektname und Speicherort des Projektes anzugeben.

2. Als Projekttyp wird `RTL Projekt` gewählt. Die Aktivierung des Feldes `Do not specify sources at this time` erlaub ein Hinzufügen der Quellen zu späterem Zeitpunkt.

3. Als Plattform wird unter `Boards` das Zedboard `xc7z020clg484-1` ausgewält. Dazu wird in den Filteroptionen als Vendor `em.avnet.com` gewählt und als Board Revision `Latest`.

4. Im erstellten Projekt lassen sich mit dem `Projekt Manager` Quellen hinzufügen. Dies geschieht durch Betätigung der Schaltfläche `Add Sources`. In dem sich öffnenden Fenster lassen sich unter dem Menüpunkt `Add or creade design sources` die entwickelten VHDL-Datein hinzufügen. Außerdem wird eine Constraints-Datei benötigt, die unter dem Menüpunkt `Add or create Constraints` eingebunden wird. In dieser Arbeit wurde die Datei `Zedboard Master XDC Rev C/D v3` verwendet, welche auf der Avnet-Homepage http://zedboard.org/support/documentation/1521 zum Download zur Verfügung steht.

5. Es ist darauf zu achten, dass sich alle Packages, die im VHDL-Code definiert wurden, im Verzeichnis `work` befinden. Die Packages lassen sich im `Projekt Manager` im Fenster `Sources` unter dem Tab `Libraries` verschieben.
6. Im Flow Navigator lässt sich unter dem Punkt `RTL ANALYSIS` durch Anwahl des Menüpuntes `open elabored Design` ein Schematic auf Register Transfer Level für den entwickelten VHDL-Code erstellen.
7. Als nächstes wird das Projekt im Flow Navigator unter `SYNTHESIS` im Menüpunkt `Run Synthesis` synthetisiert. Im Log-Fester werden Informationen über die Synthese und den Syntheseaufwand aufgelistet.
8. Nach erfolgreicher Synthese erfolgt die Implementation über den Menüpunkt `IMPLEMENTATION` → `Run Implementation` im Flow Navigator.
9. War die Synthese erfolgreich, ist der nächste Schritt das Erzeugen des Bitstreams zum Laden des Programms auf die FPGA-Plattform. Dazu wird im Flow Navigator der Menüpunkt `PROGRAM AND DEBUG` → `Generate Bitstream` ausgewählt.
10. Unter dem Menüpunkt `PROGRAM AND DEBUG` → `Open Hardware Manager` lässt sich der Hardwaremanager öffnen. Über den Menüpunkt `Open Target` → `Auto Connect` wird das Zedboard erkannt und steht zum Laden des Programms zur Verfügung. Dazu muss das Zedboard über das Netzteil mit Spannung versorgt sein, sowie über die JTAG-Schnittstelle an den PC angeschlossen und eingeschaltet sein.
11. Im `Hardware Manager` lässt sich nun über die Option `Program Device` der Zynq `xc7z020_1` auf dem Zedboard auswählen. Durch Betätigen der Schaltfläche `Program` wird der Bitstream auf den FPGA geladen.
12. Anschließend lässt sich das Programm auf dem FPGA testen. In dieser Arbeit werden dazu die I/O-Pins des ZedBoards über einen Mikrocontroller angesteuert.

Nach erfolgreichem Testen auf der FPGA-Plattform ist der nächste Schritt in der Arbeit die Synthese des VHDL-Codes mit den Cadence Tools Genus und Innovus für eine Abschätzung des Syntheseaufwands der entwickelten Module für ein Chipdesign.

E CD

Auf der beigefügten CD befinden sich die Programme und die Synthesereports, die im Rahmen dieser Arbeit erstellt wurden. Abbildung E.1 zeigt die Ordnerstruktur, wie sie auf der CD vorhanden ist.

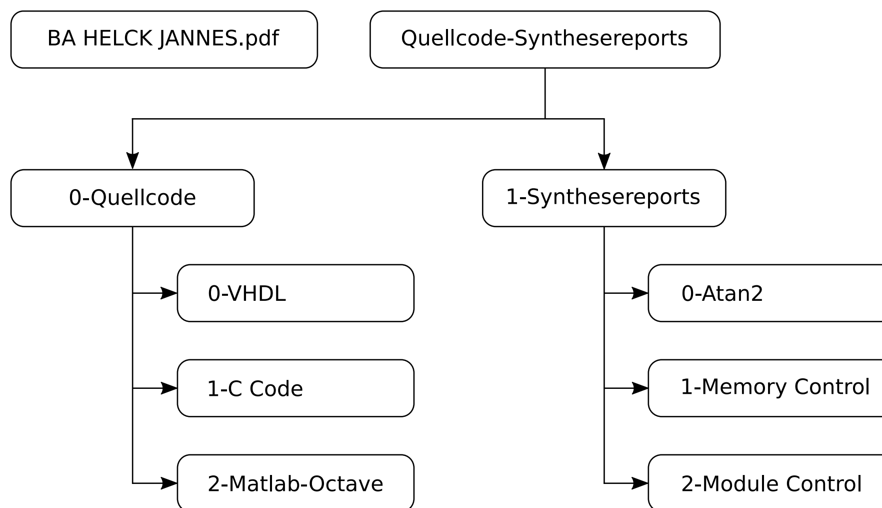


Abbildung E.1: Ordnerstruktur der beigefügten CD

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 23. August 2018	
Ort, Datum	Unterschrift