

Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (<http://www.ub.tuwien.ac.at>).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (<http://www.ub.tuwien.ac.at/englweb/>).

## Diplomarbeit

# Speicherinterface für einen schnellen A/D-Wandler

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs unter der Leitung von

Dipl.-Ing. Dr. Holger Arthaber  
Univ. Prof. Dipl.-Ing. Dr. Gottfried Magerl  
Institut für Elektrische Mess- und Schaltungstechnik  
Inst. Nr. E354



eingereicht an der Technischen Universität Wien  
Fakultät für Elektrotechnik und Informationstechnik

von

**Manfred Balik**  
Matr. Nr. 8825130  
Wienerstrasse 1/3/5  
2120 Wolkersdorf

Wien, im April 2004

# Danksagung

Ich möchte allen Mitarbeitern des Instituts für Elektrische Mess- und Schaltungstechnik an der Technischen Universität Wien danken, insbesondere meinen Betreuern Dipl.-Ing. Dr. Holger Arthaber und Univ. Prof. Dipl.-Ing. Dr. Gottfried Magerl.

Danken möchte ich auch meinen Eltern für ihre Unterstützung und die langjährige Finanzierung meines Studiums.

Besonderer Dank gebührt meiner Lebensgefährtin Claudia Krammer für ihre Geduld und Hilfe während meiner Studienzeit.

# Kurzfassung

Ziel dieser Diplomarbeit ist die Entwicklung und der Aufbau einer Schaltung (im folgenden Speicherinterface genannt), welche die Messdaten eines schnellen Analog/Digital-Wandlers in einen Speicher schreibt. Das Auslesen der Daten aus dem Speicher ist über eine einfache Schnittstelle vorzusehen. Gefordert waren eine hohe Speichertiefe, d.h. eine lange Messzeit, und niedrige Kosten des Speichers. Außerdem sollen mehrere Speicherinterfaces parallel betrieben werden können, um eine Mehrkanalmessung zu ermöglichen.

Zur Realisierung des Speicherinterfaces wurde wegen der hohen Flexibilität bezüglich der Programmierung bei geringen Kosten ein FPGA-Baustein verwendet. Die Programmierung des FPGAs erfolgte in einer Kombination aus einer graphischen Eingabe von Design-Blöcken und der Hardwarebeschreibungssprache VHDL. Die Messdaten des A/D-Wandlers sollten in ein handelsübliches SDRAM-Modul, wie es auch in Personal Computern eingesetzt wird, geschrieben werden. Das Problem dabei war, dass die Daten vom ausgewählten 14 Bit-A/D-Wandler mit der hohen Datenrate von 80 MHz kontinuierlich ausgegeben werden, das Schreiben ins SDRAM aber wegen der notwendigen Refresh-Vorgänge nicht kontinuierlich erfolgen konnte. Deshalb mussten die Messwerte vom A/D-Wandler in einem FIFO im FPGA-Baustein zwischengespeichert werden. Um die hohe Datenrate der Messwerte vom A/D-Wandler zu verringern und die volle Datenbusbreite des SDRAMs auszunützen, wurden unmittelbar am Eingang des FPGAs jeweils vier Messwerte zusammengefasst und als ein Datenwort weiterverarbeitet. Im FPGA wurden weiters eine Eingangs-Schnittstelle zum A/D-Wandler, eine Ausgangs-Schnittstelle zum Auslesen der Messwerte, ein SDRAM-Controller, der die Ansteuerung des SDRAM-Moduls durchführt, und eine Steuerung, welche die Koordination der einzelnen Komponenten steuert, realisiert. Nachdem die Funktion des FPGAs durch Simulationen überprüft war, wurde eine Schaltung entworfen, welche die Programmierung des FPGAs ermöglicht, eine Eingangs-Schnittstelle zum A/D-Wandler und eine Ausgangs-Schnittstelle zum Auslesen der Daten besitzt. Für diese Schaltung musste eine Leiterplatte entwickelt werden. Mithilfe eines einfachen Messsystems wurde die korrekte Funktion des Speicherinterfaces überprüft.

In Kapitel 1 wird die Aufgabenstellung im Detail beschrieben und in Kapitel 2 das Konzept für die Realisierung mit einem FPGA-Baustein präsentiert. Es wird die Auswahl des A/D-Wandlers, des Speichers und des FPGA-Typs beschrieben. Kapitel 3 befasst sich mit der Realisierung der Programmierung des FPGAs. Die Simulation der Funktion des FPGAs wird in Kapitel 4 besprochen. In Kapitel 5 wird auf die Schaltung und den Entwurf des Layouts eingegangen. Zum Schluss werden noch in Kapitel 6 die fertige Schaltung und die damit durchgeführten Tests beschrieben. Im Anhang wird die Bedienung des Speicherinterfaces erläutert. Den Abschluss bilden eine detaillierte Beschreibung der Steckerbelegungen, der Schaltplan mit einer Bauteilliste und das Layout mit Bestückungsplänen.

# Abstract

The goal of this diploma thesis is the development and the construction of a circuit (called in the following Memory-Interface), which writes the measured data of an Analog/Digital-Converter into a memory. The readout of the data from the memory shall be provided by a low-end interface. Low costs of the memory and a large memory depth for a long measurement period were required. To allow a measurement of multiple channels several Memory-Interfaces should be able to operate in parallel.

To realise the Memory-Interface a FPGA-Device was used, because of its flexibility for programming at low costs. The measured data of the A/D-Converter should be stored in a commercial SDRAM-Module as used in personal computers. The data flow from the selected 14 Bit-A/D-Converter is continuous with a high data rate of 80 MHz. However, writing to the SDRAM could not be continuous because of the required refresh-cycles. Therefore, the measuring data of the A/D-Converter must be stored in a FIFO built in the FPGA-Device. To reduce the high bandwidth of the measuring data of the A/D-Converter and to utilize the whole width of the SDRAMs data-bus, four data values of the A/D-Converter are combined to a data word immediately at the input of the FPGA. An Input-Interface to the A/D-Converter, an Output-Interface to read out the measured data, a SDRAM-Controller to control the communication with the SDRAM-Module und a Controlling-Unit to coordinate the several components are realised in the FPGA. After checking the functionality of the FPGA by simulating it in the design-software a schematic of the FPGA-Board was designed. The Board has to allow the programming of the FPGA and must have an Input-Interface to the A/D-Converter and an Output-Interface to read out the measured data. A printed circuit board was developed from this schematic. The correct operation of the Memory-Interface was verified by a simple measuring system.

In Chapter 1 the problem of this diploma thesis is specified in detail and in Chapter 2 the idea of the realisation by a FPGA-Device is presented. The choice of the A/D-Converter, the memory and the FPGA type is specified. Chapter 3 shows the realisation of the FPGA-programming. The simulation of the function realized in the FPGA is described in Chapter 4. In Chapter 5 the design of the schematic and printed circuit board is shown. The Appendix describes the operation of the Memory-Interface. A detailed description of the pin assignment of the connectors, the schematic, a parts list, the layout and a specification of the component placement can be found in the appendix of this diploma thesis.

# Inhaltsverzeichnis

<b>1</b>	<b>Aufgabenstellung</b>	<b>1</b>
<b>2</b>	<b>Konzept</b>	<b>3</b>
2.1	A/D-Wandler . . . . .	4
2.2	SDRAM-Modul . . . . .	6
2.3	Speicherinterface . . . . .	8
2.3.1	Auswahl des FPGA-Typs . . . . .	8
2.3.2	Auswahl der Design-Software . . . . .	8
<b>3</b>	<b>Design des FPGA</b>	<b>10</b>
3.1	A/D-Wandler Schnittstelle . . . . .	14
3.2	FIFO . . . . .	17
3.3	SDRAM-Controller . . . . .	18
3.4	Ausgangs-Schnittstelle . . . . .	24
3.5	Steuerung . . . . .	27
3.6	Einstellungen in Quartus II . . . . .	30
<b>4</b>	<b>Simulation des FPGA</b>	<b>32</b>
4.1	Ergebnisse der Simulation mit Quartus II . . . . .	32
4.2	Ergebnisse der Simulation mit ModelSim . . . . .	37
<b>5</b>	<b>Entwurf der Hardware</b>	<b>41</b>
5.1	Entwurf der Schaltung . . . . .	41
5.2	Entwurf des Layouts . . . . .	43
<b>6</b>	<b>Aufbau des Speicherinterface</b>	<b>45</b>
6.1	Aufbau der Speicherinterface-Platine . . . . .	45
6.2	Aufbau eines Messsystems . . . . .	47
<b>7</b>	<b>Zusammenfassung</b>	<b>50</b>
<b>A</b>	<b>Bedienung des Speicherinterface</b>	<b>53</b>
A.1	Programmierung des FPGAs über die ByteBlaster-Schnittstelle . . . . .	53
A.2	Programmierung des Configuration Devices und des FPGAs über die JTAG-Schnittstelle	53
A.3	Messen von Daten . . . . .	54
<b>B</b>	<b>Steckerbelegung</b>	<b>56</b>
B.1	Stecker CON1 . . . . .	57
B.2	Stecker CON2 . . . . .	58
B.3	Stecker CON3 . . . . .	59
B.4	Stecker CON4 . . . . .	59
B.5	Stecker CON5 . . . . .	60
B.6	Stecker CON6 . . . . .	61
B.7	Stecker CON7 . . . . .	62
B.8	Stecker CON8 . . . . .	63

B.9 Stecker CON9 . . . . .	64
<b>C Schaltplan</b>	<b>65</b>
<b>D Bauteilliste</b>	<b>69</b>
<b>E Layout</b>	<b>71</b>
E.1 Bauteilseite . . . . .	71
E.2 Lötseite . . . . .	72
<b>F Bestückungsplan</b>	<b>73</b>
F.1 Bauteilseite . . . . .	73
F.2 Lötseite . . . . .	74
<b>Literaturverzeichnis</b>	<b>75</b>

# Abbildungsverzeichnis

1.1	Blockschaltbild der Messschaltung . . . . .	1
1.2	Blockschaltbild des Messsystems . . . . .	2
2.1	Blockschaltbild des A/D-Wandler mit Speicherinterface . . . . .	3
2.2	Blockschaltbild des A/D-Wandlers AD6645 . . . . .	4
2.3	Timing Diagramm des A/D-Wandlers . . . . .	5
2.4	A/D-Wandler Evaluation Board . . . . .	5
2.5	Timing Diagramm des A/D-Wandler Evaluation Boards . . . . .	6
2.6	512MB SDRAM-Modul . . . . .	7
2.7	Ablauf des FPGA-Designs mit Quartus II . . . . .	9
3.1	Blockschaltbild der im FPGA realisierten Funktion . . . . .	10
3.2	Im Blockschaltbild der im FPGA realisierten Funktion verwendete Taktraten . . . . .	12
3.3	Quartus II Block Design File der im FPGA realisierten Funktion . . . . .	13
3.4	Blockschaltbild der A/D-Wandler Schnittstelle . . . . .	14
3.5	Quartus II Block Design File der A/D-Wandler Schnittstelle . . . . .	15
3.6	Altera Megafunction LPM_FF . . . . .	15
3.7	Platzierung der Flip Flops im FPGA . . . . .	16
3.8	Altera Megafunction FIFO Schritt1 . . . . .	17
3.9	Altera Megafunction FIFO Schritt 3 . . . . .	17
3.10	Blockschaltbild des SDRAM-Controller . . . . .	19
3.11	Quartus II Block Design File des SDRAM-Controllers . . . . .	20
3.12	Altera Megafunction LPM_BUSTRI . . . . .	20
3.13	Flussdiagramm des Refresh Timers . . . . .	21
3.14	Flussdiagramm der Command Engine . . . . .	22
3.15	Blockschaltbild der Ausgangs-Schnittstelle . . . . .	25
3.16	Quartus II Block Design File der Ausgangs-Schnittstelle . . . . .	26
3.17	Flussdiagramm der Steuerung . . . . .	27
4.1	Simulation mit Quartus II - Reset . . . . .	33
4.2	Simulation mit Quartus II - Initialisierung des SDRAM . . . . .	34
4.3	Simulation mit Quartus II - Refresh des SDRAM . . . . .	34
4.4	Simulation mit Quartus II - SDRAM-Write . . . . .	35
4.5	Simulation mit Quartus II - SDRAM-Write mit Refresh . . . . .	36
4.6	Simulation mit Quartus II - Ausgabe des Adresszählers . . . . .	36
4.7	Simulation mit Quartus II - SDRAM-Read . . . . .	37
4.8	Blockschaltbild der Simulation mit ModelSim . . . . .	38
4.9	Blockschaltbild des 512 MB SDRAM-Moduls . . . . .	39
4.10	Simulation mit ModelSim . . . . .	40
5.1	Blockschaltbild des Schaltplans . . . . .	42
6.1	Speicherinterface-Platine mit FPGA bestückt . . . . .	45
6.2	Speicherinterface-Platine Bauteilseite . . . . .	46
6.3	Speicherinterface-Platine Lötseite . . . . .	46

6.4	Blockschaltbild des Messsystems zur Überprüfung der Funktion . . . . .	47
6.5	Messsystems zur Überprüfung der Funktion . . . . .	48
6.6	Detail des Messsystems zur Überprüfung der Funktion . . . . .	49
A.1	Blockschaltbild des Messsystems . . . . .	54
B.1	Stecker des Speicherinterfaces . . . . .	56
C.1	Schaltplan Teil1 links . . . . .	66
C.2	Schaltplan Teil 1 rechts . . . . .	67
C.3	Schaltplan Teil 2 . . . . .	68
E.1	Layout der Bauteilseite . . . . .	71
E.2	Layout der Lötseite . . . . .	72
F.1	Bestückungsplan der Bauteilseite . . . . .	73
F.2	Bestückungsplan der Lötseite . . . . .	74

# Tabellenverzeichnis

3.1	Aufteilung der Adress Bits auf die SDRAM Steuerleitungen . . . . .	22
3.2	Zustände der Steuerleitungen für die SDRAM-Befehle . . . . .	23
3.3	Bedeutung der Status-Leitungen . . . . .	28
4.1	Ergebnisse des Quartus II Timing Analyzers . . . . .	32
B.1	Stecker CON1 . . . . .	57
B.2	Stecker CON2 . . . . .	58
B.3	Stecker CON3 . . . . .	59
B.4	Stecker CON4 . . . . .	59
B.5	Stecker CON5 . . . . .	60
B.6	Stecker CON6 . . . . .	61
B.7	Stecker CON7 . . . . .	62
B.8	Stecker CON8 . . . . .	63
B.9	Stecker CON9 . . . . .	64
D.1	Bauteilliste Teil 1 . . . . .	69
D.2	Bauteilliste Teil 2 . . . . .	70

# Abkürzungen

<b>A/D-Wandler</b>	Analog/Digital-Wandler
<b>A</b>	Address
<b>BA</b>	Bank Address
<b>BDF</b>	Block Design File
<b>CAS</b>	Column Address Select
<b>CMOS</b>	Complementary Metal Oxide Semiconductor
<b>CPLD</b>	Complex Programmable Logic Device
<b>CS</b>	Chip Select
<b>D/A-Wandler</b>	Digital/Analog-Wandler
<b>DDR</b>	Double Data Rate
<b>DIMM</b>	Dual Inline Memory Module
<b>DIP</b>	Dual Inline Package
<b>DRAM</b>	Dynamic Random Access Memory
<b>DSP</b>	Digitaler Signal Prozessor
<b>DUT</b>	Device Under Test
<b>EEPROM</b>	Electrical Erasable Programmable Read Only Memory
<b>EPP</b>	Enhanced Parallel Port
<b>FPGA</b>	Field Programmable Gate Array
<b>IEEE</b>	Institute of Electrical and Electronics Engineers
<b>IF</b>	Intermediate Frequency
<b>I/O-Karte</b>	Input/Output-Karte
<b>JTAG</b>	Joint Test Action Group
<b>LSB</b>	Least Significant Bit
<b>LVTTL</b>	Low Voltage Transistor Transistor Logic
<b>MSB</b>	Most Significant Bit
<b>NOP</b>	No Operation
<b>PLCC</b>	Plastic J-Lead Chip Carrier
<b>PLL</b>	Phase Locked Loop
<b>PQFP</b>	Plastic Quad Flat Pack
<b>RAS</b>	Row Address Select
<b>S/H</b>	Sample and Hold
<b>SDR</b>	Single Data Rate
<b>SDRAM</b>	Synchronous Dynamic Random Access Memory
<b>SFDR</b>	Spurious-Free Dynamic Range
<b>SNR</b>	Signal to Noise Ratio
<b>SRAM</b>	Static Random Access Memory
<b>TTL</b>	Transistor Transistor Logic
<b>VHDL</b>	VHSIC Hardware Description Language
<b>VHSIC</b>	Very High Speed Integrated Circuit
<b>WE</b>	Write Enable

# Kapitel 1

## Aufgabenstellung

Ziel dieser Diplomarbeit ist die Entwicklung und der Aufbau einer Schaltung (im folgenden Speicherinterface genannt), welche die Messdaten eines schnellen A/D-Wandlers zwischenspeichert und dazu in einen Speicher schreibt. Das Auslesen der Daten aus diesem Speicher ist über eine einfache Schnittstelle im Handshake-Verfahren vorzusehen. Gefordert waren eine hohe Speichertiefe, d.h. eine lange Messzeit, und niedrige Kosten des Speichers. Die Messdaten des A/D-Wandlers sollten deshalb in ein handelsübliches 512 MByte SDRAM-Modul, wie es auch in Personal Computern eingesetzt wird, geschrieben werden. Weil die Daten vom ausgewählten A/D-Wandler mit der hohen Datenrate von 80 MHz kontinuierlich ausgegeben werden, das Schreiben ins SDRAM aber wegen der Ansteuerung des SDRAMs und zur Erhaltung von Datenwerten im SDRAM notwendiger Refresh-Vorgänge, nicht kontinuierlich erfolgen kann, mussten die Messwerte vom A/D-Wandler in einem FIFO zwischengespeichert werden. Außerdem sollen mehrere Speicherinterfaces parallel geschaltet betrieben werden können, um eine Mehrkanalmessung zu ermöglichen.

Benötigt wird eine solche Schaltung z. B. zur Messung der Verzerrungen von Mikrowellenverstärkern bei hohen Signalbandbreiten. Eine solche Messschaltung ist in Abbildung 1.1 skizziert. Ein Signal  $s(t)$  wird auf eine Trägerfrequenz  $\omega_0$  aufmoduliert und vom zu messenden Verstärker (DUT) verstärkt. Die Ergebnisse der Demodulation in Inphase- und Quadraturkomponente sollen gemessen werden.

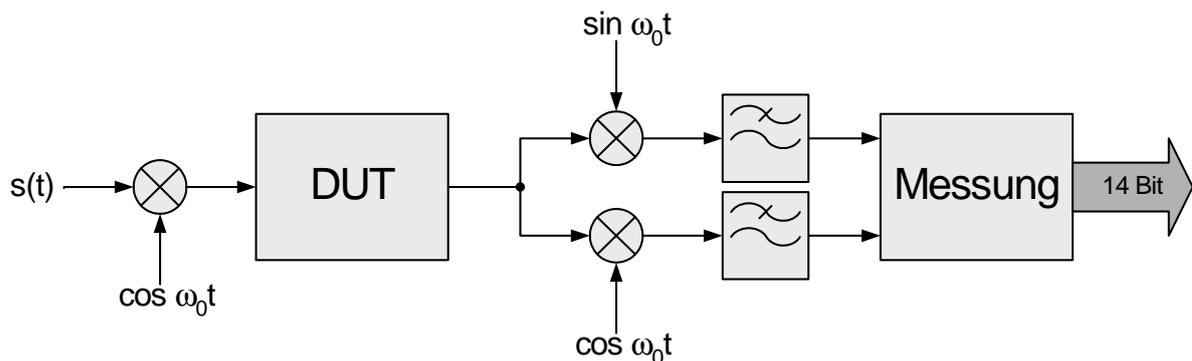


Abbildung 1.1: Blockschaltbild der Messschaltung

Der in Abbildung 1.1 dargestellte Block **Messung** ist in Abbildung 1.2 detailliert dargestellt. Man erkennt zwei parallel geschaltete A/D-Wandler mit Speicherinterface für Inphase- und Quadraturkomponente. Beide werden vom selben Takt angesteuert, d.h. die Abtastung der beiden analogen Eingangssignale erfolgt synchron. Ausgangsseitig liegen diese beiden A/D-Wandler mit Speicherinterface an einem gemeinsamen Bus. Vom Interface für Messdaten werden die Daten aus den zwei Speichern nacheinander ausgelesen.

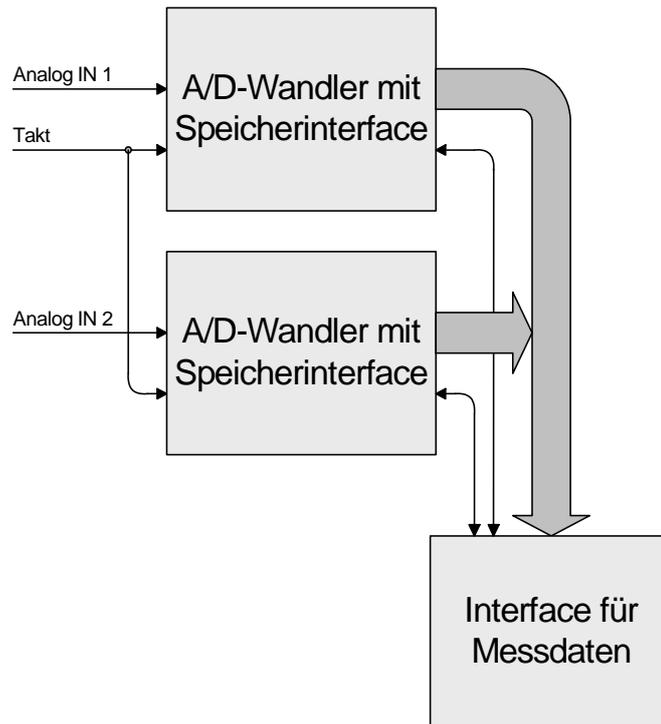


Abbildung 1.2: Blockschaltbild des Messsystems

Als Ergebnis der Diplomarbeit sollte ein A/D-Wandler mit Speicherinterface, wie in Abbildung 1.2 dargestellt, entwickelt werden. Aus der dafür entworfenen Schaltung musste zur Fertigung des Speicherinterfaces ein Layout entwickelt werden. An der fertig aufgebauten Schaltung soll die korrekte Funktion des Speicherinterfaces überprüft werden.

Das Interface zum Auslesen der Messdaten war nicht Inhalt der Diplomarbeit.

# Kapitel 2

## Konzept

Zur Realisierung des A/D-Wandlers mit Speicherinterface, wie in Abbildung 1.2 beschrieben, wurde das in Abbildung 2.1 dargestellte Konzept verwendet. Die Messdaten des A/D-Wandlers werden vom Speicherinterface übernommen und dort zwischengespeichert. Im Speicherinterface werden vier Messwerte zusammengefasst, um die Datenbusbreite des SDRAM-Moduls auszunutzen und die hohe Taktrate von 80 MHz der Messwerte des gewählten A/D-Wandlers zu reduzieren. Die Messdaten sollen in ein handelsübliches SDRAM-Modul, wie es auch in Personal Computern eingesetzt wird, geschrieben werden. Damit die Daten im SDRAM-Modul erhalten bleiben, sind wiederholte Refresh-Zyklen vorzusehen. Da diese Refresh-Zyklen auch während des Schreibens von Daten ins SDRAM notwendig sind, ist kein kontinuierliches Schreiben ins SDRAM möglich. Deshalb ist im Speicherinterface ein FIFO zur Zwischenspeicherung der Daten notwendig. Nach Ende der Messung können die Daten aus dem SDRAM ausgelesen werden und wieder als einzelne Messwerte an die Ausgangsschnittstelle ausgegeben werden.

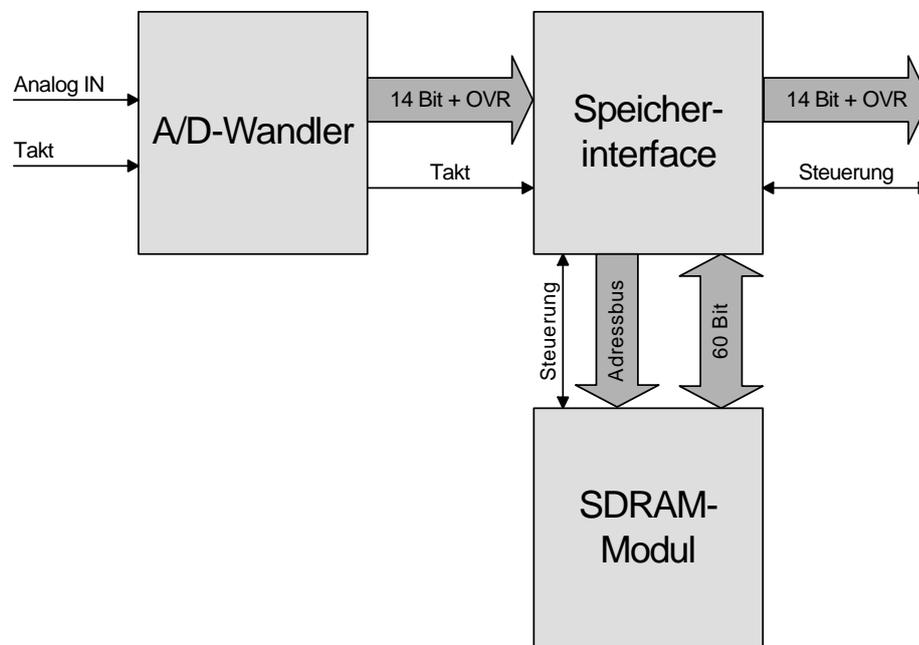


Abbildung 2.1: Blockschaltbild des A/D-Wandlers mit Speicherinterface

## 2.1 A/D-Wandler

Als A/D-Wandler wurde wegen seiner hohen Geschwindigkeit von 80 MSamples/s und der hohen Auflösung von 14 Bit der AD6645 der Firma Analog Devices ausgewählt.

Der AD6645 ist ein 14 Bit Wandler, der als 3-stufiger Kaskadenkonverter aufgebaut ist (siehe Abbildung 2.2). Das differentielle Eingangssignal  $A_{IN}$  und  $\overline{A_{IN}}$  darf einen Maximalpegel von  $2,2 V_{PP}$  um die Referenzspannung  $V_{REF}$  von 2,4 V haben. Mit einer steigenden Flanke am differentiellen Eingang  $ENCODE$  und  $\overline{ENCODE}$  wird die erste S/H-Stufe TH1 in den Halte-Modus gesetzt und so das analoge Eingangssignal gespeichert. Diese Spannung wird von dem 5-Bit A/D-Wandler ADC1 digitalisiert. Dieser Wert wird vom D/A-Wandler DAC1 wieder in einen analogen Wert umgesetzt und vom gespeicherten Eingangssignal abgezogen. Die S/H-Stufe TH2 dient dazu, die Verzögerung des ADC2 im analogen Pfad auszugleichen. Die zweite Stufe des Kaskadenkonverters bestehend aus TH3, ADC2, DAC2 und TH4 funktioniert genauso wie die erste Stufe und liefert ebenfalls einen Digitalwert mit 5 Bit. Die dritte Stufe besteht nur mehr aus dem Sample and Hold TH5 und dem 6-Bit A/D-Wandler ADC3. Die von den drei A/D-Wandlern gelieferten Werte werden in einer Logik-Schaltung zusammengefasst und auf die endgültigen Ausgangsdaten korrigiert.

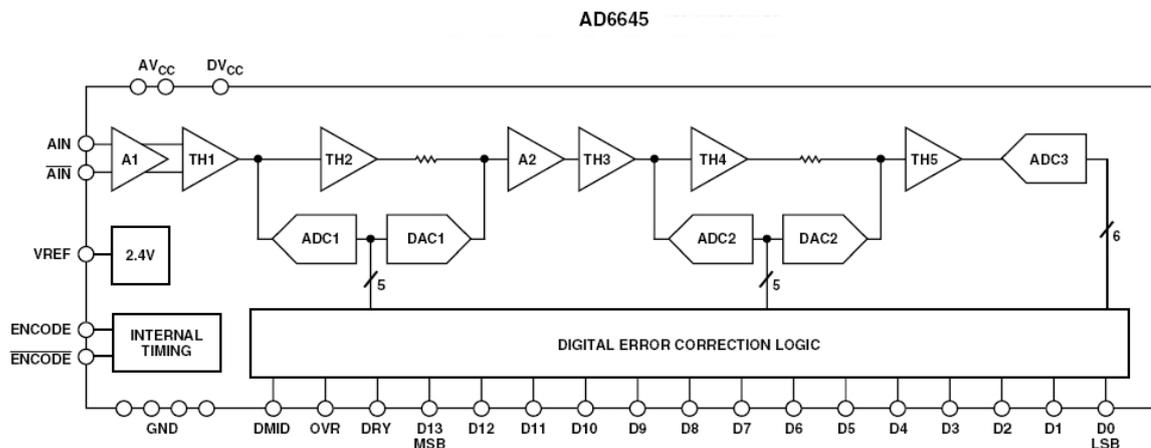


Abbildung 2.2: Blockschaltbild des A/D-Wandlers AD6645

Das digitale Ausgangssignal D0 (LSB) bis D13 (MSB) wird in der Zweierkomplement-Darstellung als 14-Bit Wert in CMOS kompatiblen Pegeln ausgegeben. Mit der steigenden Flanke des Signals DRY (Data Ready) wird angezeigt, wann gültige Daten am Ausgang anliegen (die Setup- und Hold-Time der Daten sind im Timing Diagramm Abbildung 2.3 zu sehen). Ein HIGH-Pegel am Signal OVR (OverRange) zeigt an, dass das Eingangssignal den maximalen Aussteuerbereich überschritten hat und die Ausgangsdaten daher nicht gültig sind. Das Signal DMID (DataMIDpoint) liefert die Mittenspannung der Pegel der Ausgangsdaten, die etwa die Hälfte der Versorgungsspannung der digitalen Ausgangsstufe beträgt.

Mit diesem Aufbau des A/D-Wandlers wird ein Signal zu Rauschabstand von  $SNR=75\text{dB}$  bei einer Eingangsfrequenz von  $f_{in}=15\text{ MHz}$  und 80 MSamples/s erreicht.

Dieser A/D-Wandler kann ein IF Sampling bis zu einer Frequenz von 200 MHz, d.h. auch Unterabtastung des Eingangssignals ist möglich.

Der störungsfreie Aussteuerbereich SFDR beträgt 89dBc bei einer Eingangsfrequenz von  $f_{in}=69,1\text{ MHz}$  und 80 MSamples/s.

Die kompletten Spezifikationen des AD6645 sind im Datenblatt [1] nachzulesen.

Um das bezüglich Schirmung kritische Layout des A/D-Wandlers auf der Leiterplatte des Speicherinterfaces zu vermeiden, wurde das von Analog Devices angebotene Evaluation Board für den AD6645

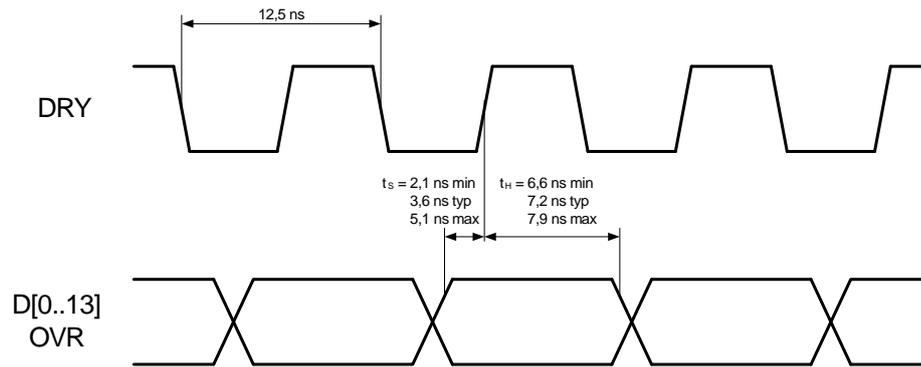


Abbildung 2.3: Timing Diagramm des A/D-Wandlers

verwendet. Außerdem wandeln die Treiber der Ausgangssignale die CMOS-Pegel des A/D-Wandlers in 3,3V-Pegel am Ausgang um. Auf dem Foto des Boards (Abbildung 2.4) ist auf der linken Seite, mit AIN markiert, der Stecker J5 für das Eingangssignal zu erkennen. Je nach Bestückung des Boards kann das Eingangssignal differentiell oder unipolar sein. Das Taktsignal zur A/D-Wandlung kann entweder an dem mit ENC gekennzeichneten Stecker J4 angelegt werden (ebenfalls je nach Bestückung des Boards differentiell oder unipolar) oder wird von einem 80 MHz Oszillator Y1 selbständig erzeugt. Auf der rechten Seite befindet sich der Stecker J2 an dem die digitalen Messdaten in 3,3V-Logik-Pegeln an das Speicherinterface ausgegeben werden. Die Steckerbelegung dieses Steckers befindet sich im Anhang in der Tabelle B.1.

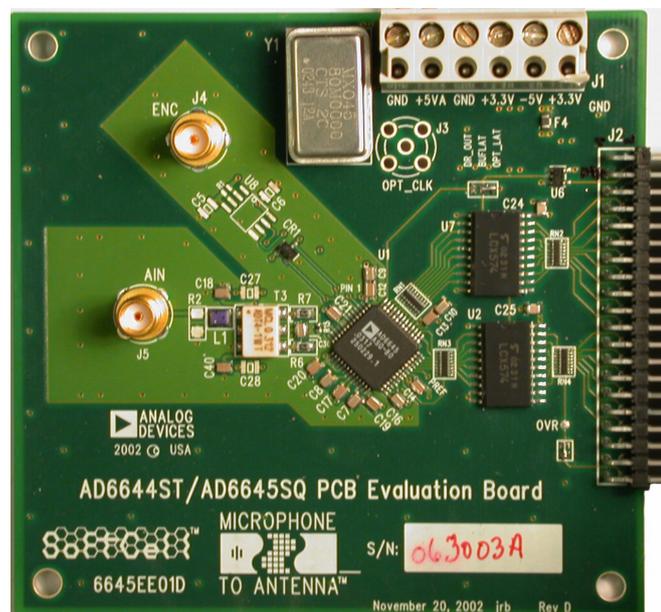


Abbildung 2.4: A/D-Wandler Evaluation Board

Durch unterschiedliche Treiber am Evaluation Board, die verschiedene Verzögerungszeiten für die Ausgangssignale DRY und D[0..13] haben, ergibt sich ein gegenüber dem A/D-Wandler geändertes Timingverhalten am Ausgang, das in Abbildung 2.5 dargestellt ist. Die Daten-Signale werden am Evaluation Board um etwa 12,5 ns verzögert, die Taktsignale jedoch nur um etwa 1,5 bis 2,5 ns. Dadurch ist der Zusammenhang zwischen der positiven Taktflanke von DRY und den dazugehörigen Abtastwert nicht mehr gegeben. Die positive Flanke von DRY kann aber zur Übernahme des vorherigen Abtastwertes D[0..13] verwendet werden. Es ist dazu zu bemerken, dass dieses Timingverhalten bei der maximalen

Sample-Frequenz des A/D-Wandlers von 80 MHz gemessen wurde und für andere Sample-Frequenzen nicht garantiert werden kann.

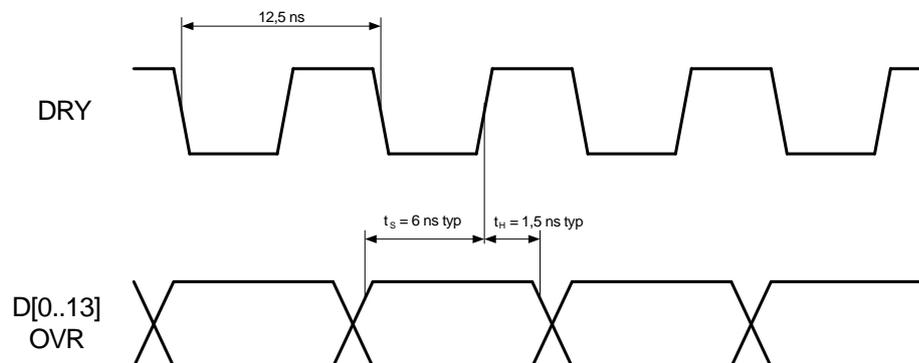


Abbildung 2.5: Timing Diagramm des A/D-Wandler Evaluation Boards

Als Anregung, wie die Datentübernahme vom A/D-Wandler und die Speicherung der Messwerte erfolgen kann, wurde das Datenblatt [2] des Analog Devices FIFO Evaluation Kits herangezogen, in dem 32 kByte große Blöcke von Messwerten in einem FIFO gespeichert werden und über eine Parallel-Schnittstelle wieder ausgelesen werden können. Da mit diesem Konzept jedoch nur kleine Datenmengen gespeichert werden können, da FIFOs mit der gewünschten großen Speicherkapazität nicht gebaut werden und eine Vergrößerung des FIFOs außerdem mit hohen Kosten verbunden wäre, wurde diesem Design keine weitere Beachtung geschenkt.

## 2.2 SDRAM-Modul

Zum Speichern der Messdaten sollte aus Kostengründen ein handelsübliches SDRAM-Modul, wie es auch in Personal Computern eingesetzt wird, verwendet werden. Es gibt eine Vielzahl an verschiedenen SDRAM-Modulen auf dem Markt, die sich bezüglich Speichergröße, Geschwindigkeit und Bauart unterscheiden. Daraus resultierend gibt es viele verschiedene Ansteuerungsarten für SDRAMs. Da der Aufwand der Ansteuerung sehr groß gewesen wäre, wenn alle möglichen Ansteuerungsvarianten in der Speicherinterface ausprogrammiert worden wären, wurde ein bestimmter SDRAM-Typ ausgewählt und nur dessen Ansteuerung implementiert.

Bei einem SDRAM handelt es sich um ein synchrones DRAM, d.h. es hat, um von einem Mikroprozessor angesteuert werden zu können, ein synchrones Interface, das vom Systemtakt gesteuert wird. Dabei werden die zur Adressierung des SDRAMs notwendigen Signale zu bestimmten Zeitpunkten vom Prozessor an das SDRAM übergeben, und dort in Latches zwischengespeichert. Der Vorteil ist, dass der Prozessor nicht während eines ganzen Schreib- oder Lesezyklus die Steuerleitungen kontrollieren muss und dadurch gebremst wird, sondern bis z.B. das SDRAM einen Datenwert ausgibt andere Aufgaben erledigen kann. Da es sich bei SDRAMs um dynamische Speicher handelt, bei dem eine Speicherzelle aus sehr kleinen Kapazitäten besteht, die eine Ladung nur für kurze Zeit halten können, müssen die gespeicherten Daten laufend erneuert werden, damit sie nicht verloren gehen (refresh). Die aktuellen SDRAM-Module bieten dafür einen Auto-Refresh und einen Self-Refresh-Modus an. Beim Auto-Refresh wird dem SDRAM nur der Refresh-Befehl gegeben und das SDRAM verwaltet selbständig, welcher Teil des Speichers gerade refreshet werden muß. Der gesamte Speicher muß innerhalb von 64 ms einmal komplett refreshet werden. Der Kennwert *8k-Refresh* bzw. *4k-Refresh* gibt an, wieviele Refresh-Zyklen das SDRAM braucht, um den Speicher einmal komplett zu refreshen. Im Self-Refresh-Modus erledigt das SDRAM komplett selbstständig den nötigen Refresh, während dieser Zeit ist jedoch kein Schreib- oder Lesezugriff auf das SDRAM möglich. Dieser Modus ist vorgesehen, um die gespeicherten Daten zu halten, falls sich das ganze System zum Beispiel in einem Power-Save-Modus befindet.

Die Bauform der SDRAM-Module ist ein 168-Pin DIMM Modul, das zwei Codiernasen besitzt (siehe Abbildung 2.6). Die Lage der linken Codiernase, bezeichnet als *DRAM Key*, beschreibt, ob das SDRAM *buffered* (Codiernase mittig) oder *unbuffered* (Codiernase nach rechts verschoben) aufgebaut ist. Bei buffered SDRAMs sind alle Signale außer RAS und den Datenbits gepuffert, bei unbuffered gibt es keine Puffer. Unbuffered SDRAMs werden auch als Registered SDRAMs bezeichnet.

Mit der Lage der zweiten Codiernase, bezeichnet als *Voltage Key*, wird die benötigte Versorgungsspannung angegeben. Liegt die Codiernase nach links verschoben, so werden 5V benötigt, liegt die Codiernase mittig, so ist das SDRAM-Modul mit 3,3V zu versorgen. Die Versorgungsspannung legt auch fest, welche Spannungspegel an die Eingänge des SDRAM-Moduls angelegt werden dürfen.

SDRAM-Module unterscheiden sich weiters in ihrer Bestückung mit SDRAM-Chips. Je nach der zur Zeit kostengünstigsten Speichergröße der SDRAM-Chips werden die Module einseitig, d.h. mit 8 Speicherchips, oder beidseitig, d.h. mit 16 Speicherchips, bestückt. Dadurch kommt es, abhängig von der Bestückung, zu einer unterschiedlichen Adressierung gleich großer SDRAM-Module.

Neben der Speichergröße eines SDRAM-Moduls ist die Geschwindigkeit ein weiteres wichtiges Auswahlkriterium. Das erste Kriterium dabei ist die Datenrate, wobei man *SDR* und *DDR*, bei der die Daten mit der doppelten Taktrate gespeichert und gelesen werden können, unterscheidet. Weiters unterscheidet man *PC100* und *PC133*-Typen, die für eine Taktfrequenz von 100 oder 133 MHz geeignet sind. Außerdem wird noch angegeben, wieviel Taktzyklen ein SDRAM braucht, bis nach dem Anlegen der Spalten-Adresse (Column Address) Daten ausgelesen werden können. Dieser Wert wird *CAS Latency (CL)* genannt und kann 2 oder 3 Taktzyklen betragen.

Die Datenbusbreite von SDRAM-Modulen beträgt 64 Bit. Falls eine Paritätskontrolle der Daten im SDRAM vorgesehen ist, in der acht Datenbits zu einem Paritätsbit zusammengefasst werden, erhöht sich die Anzahl der Datenbits um acht auf 72 Bit.

Der Aufbau und die Funktion eines SDRAMs und die zur Ansteuerung notwendigen Timingdiagramme können im Buch [3] genau nachgelesen werden.



Abbildung 2.6: 512MB SDRAM-Modul

Die Wahl des Speichers fiel auf die Type MT16LSDT6464AG-13E der Firma Micron (siehe Abbildung 2.6), der folgende Eigenschaften hat:

- 512 MByte
- 64 Bit Datenbusbreite
- 3,3V Versorgung (rechte Codiernase mittig)
- unbuffered (linke Codiernase nach rechts verschoben)
- 2 Bank-Aufbau (vorne und hinten mit je acht Stück 32MBitx8 SDRAM-Chips bestückt)
- Geschwindigkeit SDR, PC133, CL2
- 8k-Refresh
- Bauform 168-Pin DIMM

Um eine genügend lange Messzeit zu erhalten, wurde ein Speicher mit 512 MByte ausgewählt. Damit die Datenbusbreite von 64 Bit ausgenutzt werden kann, werden 4 Messwerte zu je 15 Bit (14 Datenbits und Overrange) zusammengefasst und in einem Schreibzyklus gespeichert. In ein 512 MByte-Speichermodul können somit 268.435.456 Messwerte gespeichert werden, was eine Messzeit von 3,36 Sekunden bei 80 MSamples/s ergibt. Durch das Zusammenfassen von 4 Messwerten wird die Taktrate der

Messwerte von 80 auf 20 MSamples/s reduziert. Deshalb muss das SDRAM nicht mehr mit der vollen Geschwindigkeit betrieben werden, sondern es reicht eine Taktfrequenz von 33 MHz.

## 2.3 Speicherinterface

Zur Realisierung des Speicherinterfaces wird eine logische Funktion benötigt, die mit einzelnen Logikgatter-Bausteinen nicht mehr realisierbar wäre. Deshalb wurden ein digitaler Signalprozessor DSP oder programmierbare Logikbausteine wie ein CPLD bzw. FPGA in Betracht gezogen. Die Entscheidung fiel wegen der hohen Flexibilität bezüglich der Programmierung bei geringen Kosten auf einen FPGA.

Die Erklärung der Funktionsweise von CPLDs ist [4] zu entnehmen. Weiterführende Erläuterungen zum Verständnis des Aufbaus und der Funktionsweise von FPGAs und den Ablauf eines FPGA-Designs sind in [5] enthalten.

### 2.3.1 Auswahl des FPGA-Typs

Als FPGA-Baustein wurde unter Zuhilfenahme des “Altera Component Selector Guide” [6], in dem die wichtigsten Eigenschaften und Parameter der FPGAs zusammengefasst sind, die Type ACEX 1K, ein Low-Density FPGA der Firma Altera, ausgewählt. Dieser FPGA wird in vier Größen erzeugt, mit 10.000 bis 100.000 Logik-Gattern für allgemeine logische Funktionen, und 12.288 bis 49.152 RAM-Bits. Das RAM in diesem FPGA ist als Dual Port-RAM ausgeführt und kann somit auch als FIFO-Speicher verwendet werden. Weiters gibt es diesen FPGA mit drei verschiedenen Geschwindigkeiten der Logik-Gatter (Speedgrade), wobei der Speedgrade 1 der schnellste und Speedgrade 3 der langsamste FPGA ist. Als Vergleich der Leistungsfähigkeit der verschiedenen Speedgrades ist in den Datenblättern als Beispiel ein “16-zu-1 Multiplexer” angeführt, der für Speedgrade 3 eine Ausführungszeit von 6,6 ns und für Speedgrade 1 nur mehr 3,5 ns benötigt. Je nach Größe und Gehäuse-Bauform des FPGAs gibt es 136 bis 333 I/O-Pins, die kompatibel zu 2,5 V, 3,3 V und 5,0 V Logik-Pegeln sind und einzeln als Ein- oder Ausgänge oder Tri-State geschaltet werden können. Diese I/O-Pins sind intern speziell geschützt und können auch vor oder während der Konfiguration des FPGAs an Eingangsspannungen geschaltet werden. Die Architektur des ACEX 1K basiert auf CMOS SRAM-Elementen die beliebig oft neu programmiert werden können, um das Testen der Programmierung zu vereinfachen. Zur Programmierung ist im FPGA bereits ein Interface für die IEEE-genormte JTAG-Schnittstelle und die von Altera festgelegte ByteBlaster-Schnittstelle implementiert.

Die genaue Type des FPGAs bezüglich Größe und Geschwindigkeit kann im vorhinein nicht festgelegt werden, sondern ergibt sich am Ende des Designprozesses.

Die kompletten Spezifikationen des ACEX 1K sind in den Datenblättern [7] und [8] nachzulesen.

### 2.3.2 Auswahl der Design-Software

Zum Entwurf des FPGAs sollte eine Design-Software der Firma Altera verwendet werden. Aufgrund der Beschreibungen der Software im “Altera Design Software Selector Guides” [9] wurde die neuere Design-Software Quartus II dem älteren MAX+PLUS II vorgezogen, da Quartus II eine gegenüber MAX+PLUS II verbesserte graphische Design-Eingabe anbietet, wegen neuerer Algorithmen schnellere Compile-Ergebnisse liefert und eine verbesserte Timing-Analyse des fertigen Designs ermöglicht. Quartus II bietet eine komplette Design-Umgebung von der Design-Eingabe bis hin zur Programmierung des FPGAs.

Der Ablauf des Designs eines FPGAs mit Quartus II ist in Abbildung 2.7 zu sehen.

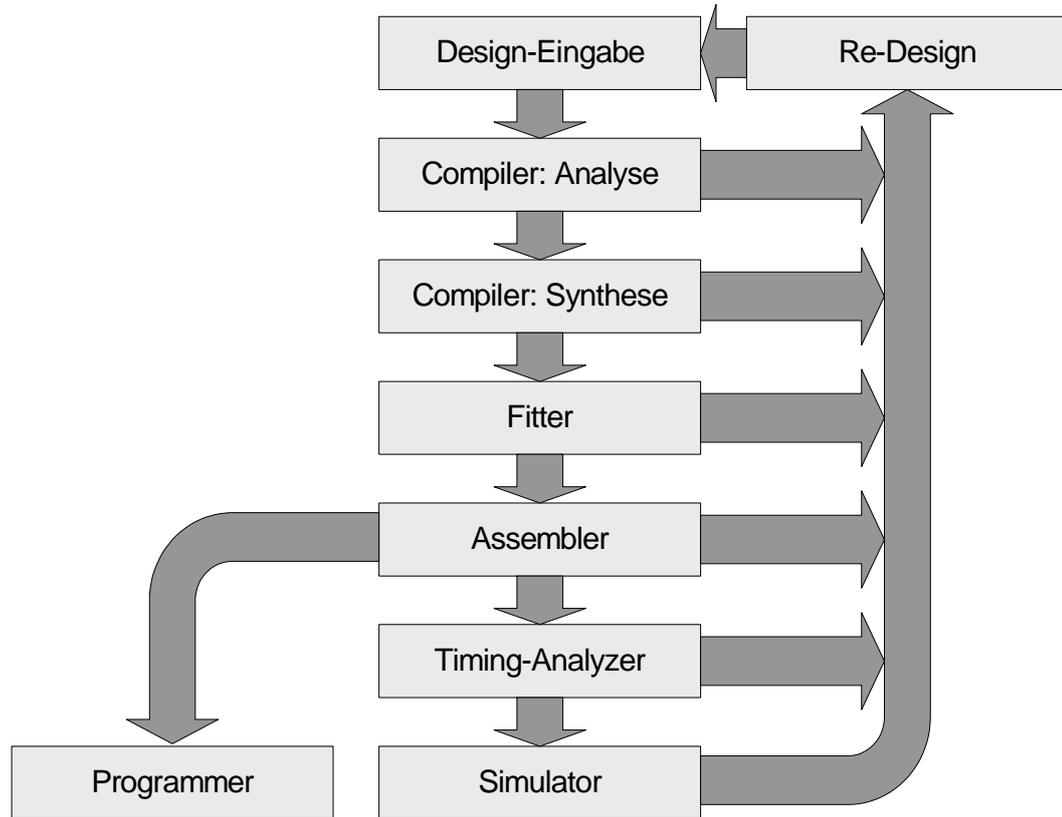


Abbildung 2.7: Ablauf des FPGA-Designs mit Quartus II

Die *Design-Eingabe* kann in graphischer Weise als Schaltplan, als Block-Design mit von Altera bereitgestellten Megafunctions, in einer Hardwarebeschreibungssprache wie VHDL, Verilog oder Altera HDL oder als beliebige Kombination dieser Möglichkeiten erfolgen. Der *Compiler* von Quartus II analysiert das eingegebene Design und synthetisiert daraus logische Gleichungen, die der *Fitter* im ausgewählten FPGA zu realisieren versucht. Vom *Assembler* wird daraus ein Ausgabe-File erzeugt, das mit dem *Programmer* in den FPGA geladen werden kann. Das zeitliche Verhalten des Designs, wie zum Beispiel benötigte Setup- und Hold-Zeiten sowie maximale Clock-Frequenzen, werden vom *Timing-Analyser* angezeigt. Die korrekte Funktion des Designs kann mit den ermittelten Gatterverzögerungszeiten und realen Laufzeiten der Signale im FPGA mit dem *Simulator* überprüft werden.

Die Realisierung des Speicherinterfaces ist in den Kapiteln 3 bis 6 detailliert beschrieben.

## Kapitel 3

# Design des FPGA

In Abbildung 3.1 ist die Funktion, die im FPGA realisiert wurde, als Blockschaltbild dargestellt.

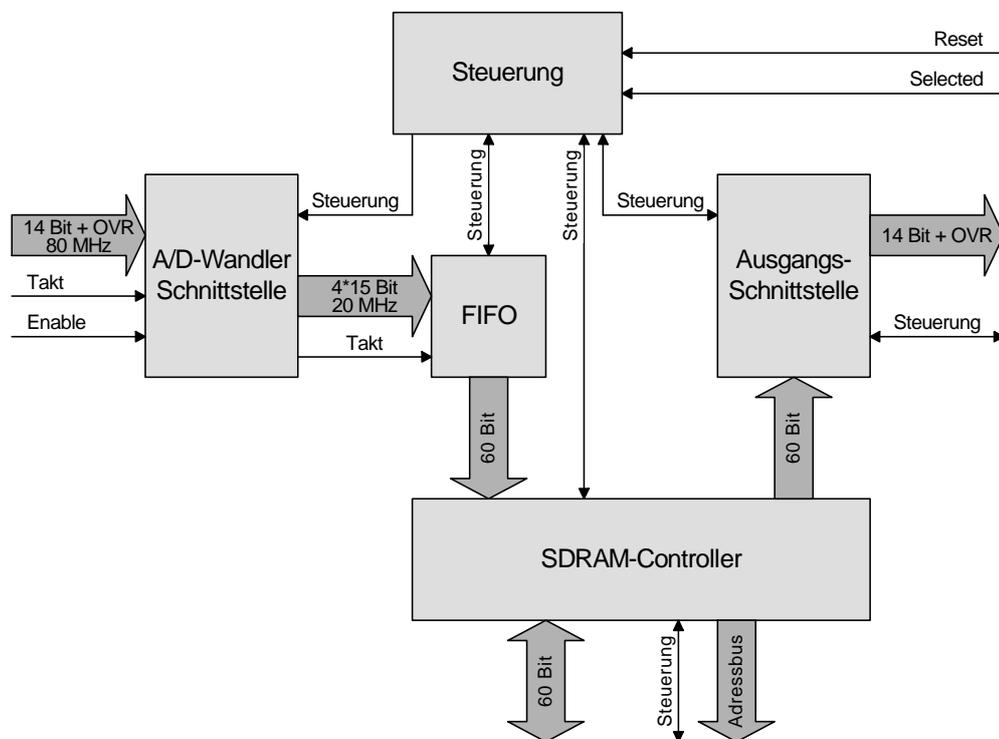


Abbildung 3.1: Blockschaltbild der im FPGA realisierten Funktion

Den zentralen Teil der Funktion bildet die **Steuerung**. Sie koordiniert das Zusammenspiel aller anderen Funktionseinheiten. Auf der linken Seite des Blockschaltbildes befindet sich der Eingang für die Messdaten. Der A/D-Wandler liefert Messwerte mit 15 Bit (14 Datenbits und OVR) und ein Taktsignal das anzeigt, wann die Daten übernommen werden können. Mit **Enable** kann zusätzlich gesteuert werden, wann Messwerte vom Speicherinterface aufgezeichnet werden sollen, falls der A/D-Wandler laufend misst, aber nur ein bestimmter Zeitpunkt aufgenommen werden soll. Vier Messwerte werden in der A/D-Wandler Schnittstelle zu einem 60 Bit Datenwort zusammengefasst, das dadurch nur mehr eine Datenrate von 20 MHz besitzt. Dieses Datenwort wird in einem FIFO-Speicher abgelegt, der als Zwischenspeicher notwendig ist, da die Daten vom A/D-Wandler kontinuierlich ausgegeben werden, aber das Schreiben in

das SDRAM, den endgültigen Datenspeicher, aufgrund der Ansteuerung des Speichers und zwischendurch notwendiger Refresh-Zyklen nicht kontinuierlich erfolgen kann. Die Kommunikation des Speicherinterfaces mit dem SDRAM wird vom SDRAM-Controller durchgeführt. Der SDRAM-Controller liest die Messdaten aus dem FIFO aus und speichert sie im SDRAM und generiert selbständig die notwendigen Refresh-Zyklen damit die Daten nicht verloren gehen. Wenn die Messung beendet ist, bleiben die Daten im SDRAM gespeichert. Als Ende der Messung wurde das Ausbleiben von Messwerten für 100 Taktzyklen definiert. Wird das Speicherinterface mit dem Selected-Signal ausgewählt, liest der SDRAM-Controller die gespeicherten Datenworte aus dem SDRAM und liefert die Daten an die Ausgangs-Schnittstelle. In der Ausgangs-Schnittstelle wird das Datenwort wieder in die vier einzelnen Messwerte aufgeteilt, die im Handshake-Verfahren nacheinander am Ausgang auf der rechten Seite des Blockschaltbildes ausgegeben werden. Die Ausgabe der Messergebnisse erfolgt in der Reihenfolge, in der sie ins SDRAM geschrieben wurde, d.h. es ist beim Auslesen keine explizite Adressierung eines Messwerts vorgesehen. Die Daten können nur einmal aus dem Speicher ausgelesen werden, danach kann die ganze Schaltung mit dem Reset-Signal wieder in den Ausgangszustand gesetzt werden.

Damit der FIFO-Speicher nicht überläuft, muss schneller in das SDRAM geschrieben werden, als Daten vom A/D-Wandler geliefert werden. Zum Schreiben ins SDRAM wird der Burst 8-Modus verwendet, d.h. es werden mit einer Adressierung des SDRAMs acht Schreibzyklen hintereinander durchgeführt (siehe Kapitel 3.3). Vom A/D-Wandler werden acht mal vier Datenwerte in der Zeit von  $\frac{8 * 4}{80 \text{ MHz}} = 400 \text{ ns}$  ausgegeben. Da ein Burst 8-Schreibzyklus elf Taktzyklen benötigt, muss die Taktfrequenz mindestens  $\frac{11}{400 \text{ ns}} = 27,5 \text{ MHz}$  betragen. Außerdem muss noch die Zeit der Refresh-Zyklen, die das kontinuierliche Schreiben ins SDRAM unterbrechen, aufgeholt werden. Als Takt für die Steuerung und den SDRAM-Controller wurden deshalb 33,3333 MHz gewählt. Die Dauer eines Burst 8-Schreibzyklus ergibt sich daher mit  $\frac{11}{33,3333 \text{ MHz}} = 330 \text{ ns}$ , die eines Refresh-Zyklus, der für diese Taktfrequenz drei Taktzyklen benötigt, mit  $\frac{3}{33,3333 \text{ MHz}} = 90 \text{ ns}$ . Man erkennt, dass bei einer Taktfrequenz von 33,3333 MHz die im FIFO während eines Refresh-Zyklus zwischengespeicherten Daten, bereits innerhalb von zwei Burst 8-Schreibzyklen, d.h. innerhalb von 800 ns, durch das schnellere Schreiben ins SDRAM aufgeholt werden.

In Abbildung 3.2 sind die im FPGA verwendeten Taktraten dargestellt.

Die A/D-Wandler Schnittstelle wird mit dem 80 MHz-Takt der Daten vom A/D-Wandler betrieben. Das Schreiben der von der A/D-Wandler Schnittstelle zusammengefassten Daten ins FIFO erfolgt mit einem aus dem 80 MHz-Takt abgeleiteten 20 MHz-Takt. Das Auslesen der Daten aus dem FIFO und die restlichen Funktionen im FPGA werden vom 33,3333 MHz Systemtakt betrieben. Durch das asynchrone Schreiben und Lesen des FIFOs wird der Übergang zwischen den beiden verschiedenen Taktraten durchgeführt. Es ist zu beachten, dass alle Gatter im FPGA, auch die durch VHDL-Programcode erzeugten, nur mit der positiven Taktflanke schalten.

Zum Auslesen der Daten aus dem Speicherinterface wird an die Ausgangs-Schnittstelle eine I/O-Karte der Firma ADLINK Technology Inc. angeschlossen. Zur Kommunikation wird ein einfaches Handshakeverfahren verwendet, in dem das Speicherinterface ein Request an die I/O-Karte ausgibt, wenn die Ausgangsdaten bereitstehen. Die I/O-Karte bestätigt mit einem Acknowledge die Übernahme der Daten. Das Timingverhalten des Speicherinterfaces wurde an die Geschwindigkeit der PCI-7200 und PCI-7300A Karten der Firma ADLINK angepasst. Die kompletten Spezifikationen der I/O-Karten sind in den Datenblättern [10] und [11] nachzulesen.

Realisiert wurde das Blockschaltbild Abbildung 3.1 in Quartus II in einem Block Design File (BDF), das in Abbildung 3.3 zu sehen ist.

Im Block Design File wurden weitgehend genau die selben Funktionsblöcke wie im bereits beschriebenen Blockschaltbild realisiert. Dazugekommen sind nur der Block `altclock0 (inst51)` und vier D-Flip Flops (`inst6`, `inst7`, `inst12` und `inst13`).

Der Block `altclock0 (inst5)` ist eine Altera Megafunktion, die eine PLL für den Eingangstakt CLKIN realisiert. Megafunktionen sind von Altera zur Verfügung gestellte Funktionen, bei denen noch Werte, wie

<sup>1</sup>Eine Instance ist die Verwendung einer logischen Funktion in einem Design-File. Im Block Design File befindet sich der Instance-Name in der linken unteren Ecke des jeweiligen Blocks.

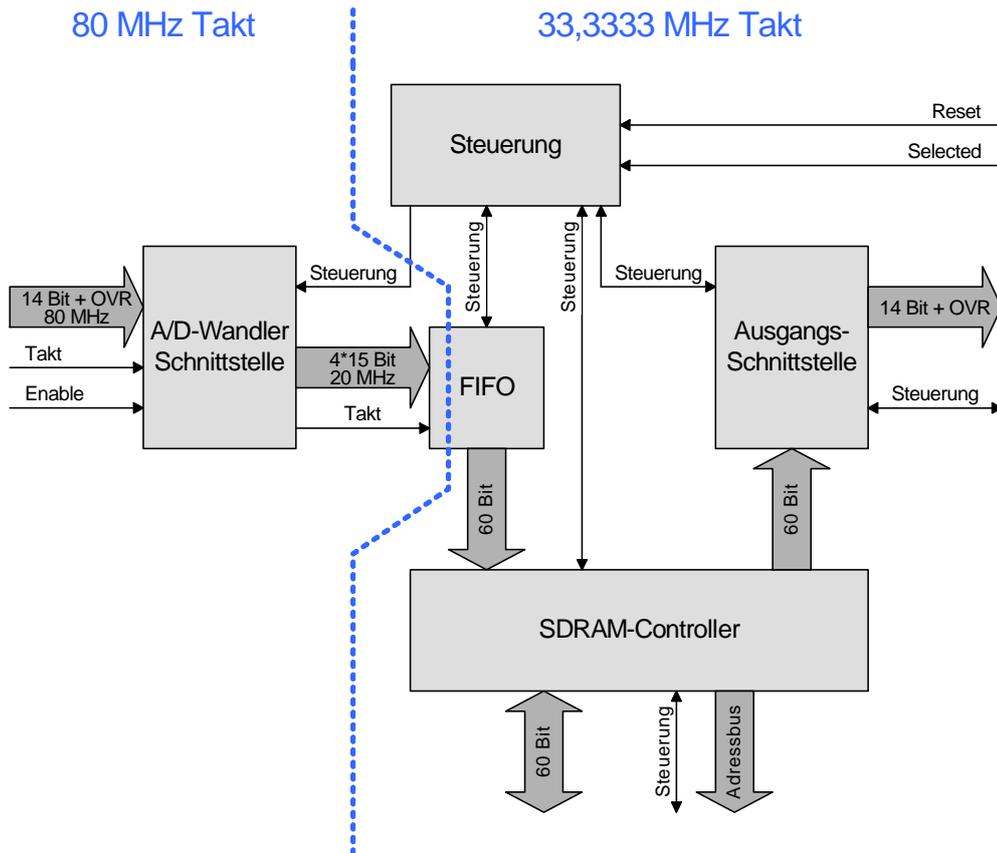


Abbildung 3.2: Im Blockschaltbild der im FPGA realisierten Funktion verwendete Taktraten

zum Beispiel die benötigte Taktfrequenz, eingestellt werden können und die Hardware des verwendeten FPGAs optimal ausgenutzt wird, d.h. in diesem Fall wird die PLL nicht in Logikzellen realisiert, sondern die im verwendeten FPGA ACEX 1K vorhandene PLL verwendet. Der Eingangstakt von 33,3333 MHz liegt an einem speziellen Pin, der als “Dedicated Clock Pin for Global\_CLK” bezeichnet wird, da er eine besonders kurze Laufzeit zur PLL hat. Diese PLL für den Takt zu verwenden, hat den Vorteil, dass die PLL einen Takt mit exaktem 50% Duty-Cycle realisiert und das Taktsignal in ein im FPGA globales Clock-Signalnetz einspeist. Dadurch werden die Laufzeiten des Clock-Signals CLK zu allen Gattern im FPGA genau gleich, d.h. alle Gatter schalten exakt synchron.

Die vier D-Flip Flops (*inst6*, *inst7*, *inst12* und *inst13*) dienen dazu, die beiden asynchronen Eingangssignale **RESETIN** und **SELECTED** mit dem internen Takt zu synchronisieren. Dies ist notwendig, da Zustandswechsel von Eingangssignalen, die genau zum Zeitpunkt einer Taktflanke des internen Takts auftreten, unvorhersehbare Zustände von Gattern in FPGA hervorrufen können. Zu beachten ist dabei jedoch, dass die Eingangssignale dadurch erst mit zwei Taktperioden Verzögerung an die nachfolgende Logik gelangen.

Im Block Design File sind außerdem noch zehn heller markierte Ausgangspins zu sehen. Sie werden zur Funktion des Speicherinterfaces nicht benötigt, sondern sind nur interne Signale des Designs, die zur Kontrolle der Funktion auf Pins geführt wurden.

Auf die genaue Bedeutung der einzelnen Steuerleitungen zwischen den Blöcken wird bei der Beschreibung des jeweiligen Blocks in den Kapiteln 3.1 bis 3.5 genauer eingegangen.

Die Programmierung der Funktion einiger Blöcke erfolgte in der Hardwarebeschreibungssprache VHDL. Zum Erlernen dieser Programmiersprache wurden [12] bis [21] verwendet.



### 3.1 A/D-Wandler Schnittstelle

Die A/D-Wandler Schnittstelle ist der zeitkritischste Teil des ganzen FPGA-Designs, da der kontinuierliche Strom der Messdaten schnell übernommen werden muss. Abbildung 3.4 zeigt ein Blockschaltbild dieser Funktion.

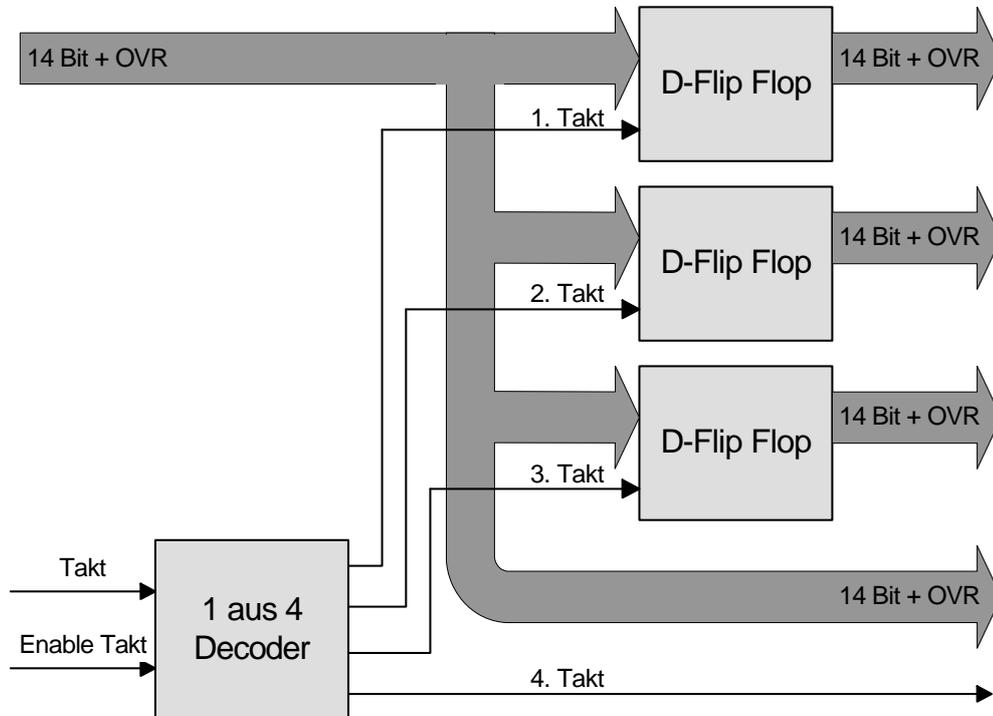


Abbildung 3.4: Blockschaltbild der A/D-Wandler Schnittstelle

Um vier 15 Bit Messwerte zusammenfassen zu können, müssen drei Messwerte zwischengespeichert und gemeinsam mit dem vierten Messwert als 60 Bit Wert an den Ausgang ausgegeben werden. Die Speicherung der Messwerte erfolgt jeweils in einer Bank aus 15 D-Flip Flops. Vom 1 aus 4 Decoder wird mit den ersten drei Takten jeweils eine Flip Flop-Bank getriggert, die dann die Messwerte übernimmt und speichert. Mit dem vierten Takt wird ein Steuersignal an die nächste Schaltungsstufe ausgegeben, dass der 60 Bit Wert übernommen werden muss. Mit dem nächsten Takt geht der 1 aus 4 Decoder wieder in den Zustand des ersten Takts und ein neuer Speicher-Zyklus fängt an. Der 1 aus 4 Decoder kann außerdem mit dem Enable Signal für den Takt ein- und ausgeschaltet werden.

Realisiert wurde das Blockschaltbild Abbildung 3.4 in Quartus II in einem Block Design File (BDF), das in Abbildung 3.5 zu sehen ist.

Der Eingang für die Messdaten vom A/D-Wandler befindet sich im Block Design File links oben. Die Eingangsdaten `DIN[0..13]` und `OVR` werden im Block `zusammenfassen_in1` (`inst1`) zu einem gemeinsamen 15 Bit breiten Datenstrom `data[0..14]` zusammengefasst und bilden die Eingangsdaten der Flip Flop-Bänke `LPM_FlipFlop` (`inst3`, `inst4` und `inst5`). Die Ausgänge `q1[0..14]`, `q2[0..14]` und `q3[0..14]` der Flip Flop-Bänke und der zusammengefasste Eingangs-Datenstrom `data[0..14]` werden im Block `zusammenfassen_in2` (`inst2`) zum 60 Bit breiten Ausgangsdatenwert `FIFOIN[0..59]` zusammengefasst. Die beiden Blöcke, die nur verschiedene Signale zusammenfassen, wurden in VHDL programmiert. Die Flip Flop-Bänke wurden mit Hilfe der Altera Megafunktion `LPM_FF` (siehe Abbildung 3.6) realisiert. Der Clock-Eingang dieser Flip-Flops wird vom 80 MHz Takt `DRY` angesteuert, der Enable-Eingang vom 1 aus 4 Decoder. An den

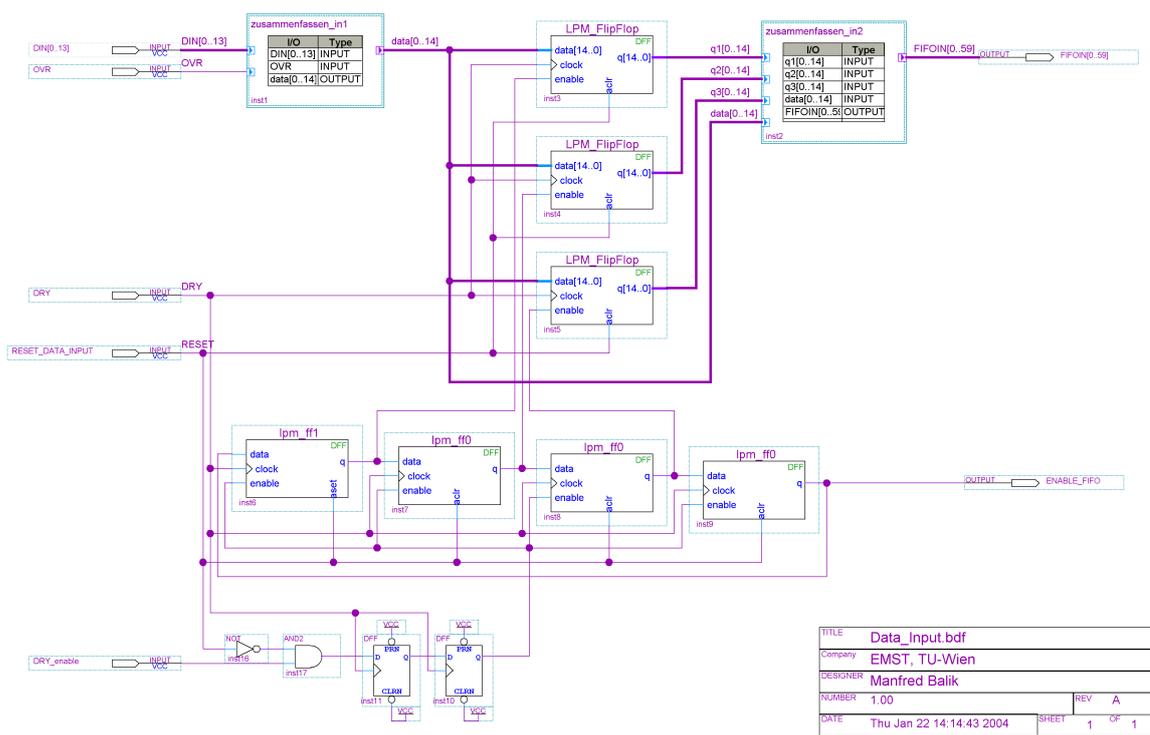


Abbildung 3.5: Quartus II Block Design File der A/D-Wandler Schnittstelle

asynchronen Clear-Eingängen (aclr) werden die Flip Flops im Reset-Fall von der zentralen Steuerung (siehe Kapitel 3.5) mit dem Signal RESET\_DATA\_INPUT auf null gesetzt.

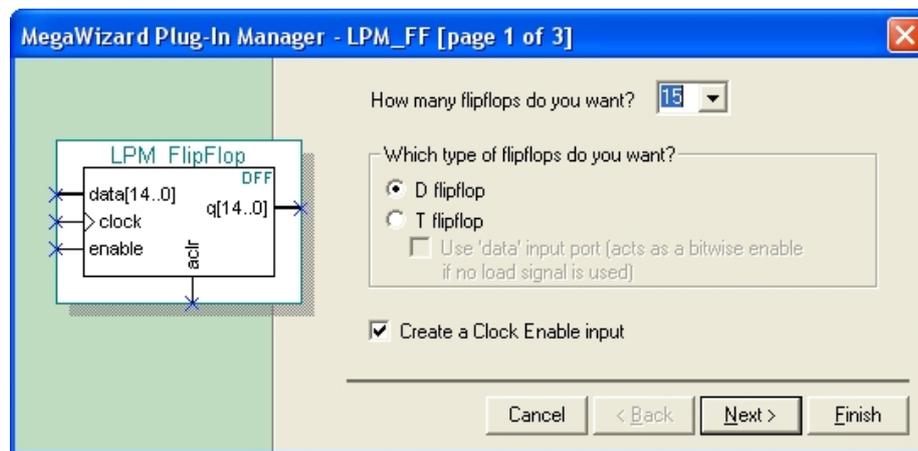


Abbildung 3.6: Altera Megafunktion LPM\_FF

Es wurde zuerst versucht den 1 aus n Decoder direkt mit einer Altera Megafunktion zu realisieren, was aber zu keiner korrekten Ansteuerung der D-Flip Flops führte, da bei den Zustandswechseln immer wieder Spikes auftraten. Deshalb wurde der Decoder als rückgekoppelte Kette aus vier D-Flip Flops lpm\_ff1 und lpm\_ff0 (inst6, inst7, inst8 und inst9) mit der Megafunktion LPM\_FF realisiert. Die Ausgangssignale der ersten drei Flip Flops bilden die Enable-Signale der Flip Flop-Bänke zur Zwischenspeicherung der Daten. Das Ausgangssignal des vierten Flip Flops wird an den Eingang des ersten Flip Flops zurückgeführt und

dient außerdem als Ausgangssignal `ENABLE_FIFO` und gibt an, dass an `FIFOIN[0..59]` die Ausgangsdaten zur Übernahme für den nachfolgenden FIFO-Speicher (siehe Kapitel 3.2) bereit stehen. Der Clock-Eingang dieser Flip Flop-Kette wird vom Takt `DRY` angesteuert. Der Enable-Eingang wird vom Eingangssignal `DRY_enable` angesteuert, dass mit Hilfe zweier Flip Flops (`inst10` und `inst11`) an den Eingangstakt `DRY` synchronisiert wurde. Die UND-Verknüpfung (`inst17`) von `DRY` und dem mit einem NOT-Gatter (`inst16`) invertierten Reset-Signal `RESET_DATA_INPUT` stellen sicher, dass keine Daten vom Speicherinterface aufgenommen werden können, solange die zentrale Steuerung (siehe Kapitel 3.5) ein Reset-Signal liefert. Durch das Reset-Signal wird die Flip Flop-Kette in einen Anfangszustand gesetzt, der für das erste Flip Flop `lpm_ff1` (`inst6`) 1 und für die anderen Flip Flops `lpm_ff0` (`inst7`, `inst8` und `inst9`) 0 ist.

Um möglichst kurze Laufzeiten der Eingangsdaten zu haben, wurde beim Platzieren der Flip Flops im FPGA dem Fitter von Quartus II, der die optimale Platzierung der benötigten Gatter im FPGA vornimmt, nicht freie Hand gelassen. Obwohl die Optimierungsmethode des Fitters auf "Geschwindigkeit" statt auf "Fläche" eingestellt wurde, konnte keine optimale Platzierung und somit keine kurzen Laufzeiten der Eingangs-Signale zu den Flip Flops erreicht werden. Deshalb wurde dem Fitter die Lage der Flip Flops genau vorgegeben. Abbildung 3.7 zeigt die von Quartus II dargestellte geometrische Lage der Pins und Gatter im FPGA. Wie in der Abbildung zu sehen ist, wurden sie direkt zwischen den Eingangspins an der Oberseite und den FIFO-Zellen, die senkrecht in der Mitte des FPGAs untergebracht sind, platziert. Durch diese Platzierung konnten die Messwerte, mit den am Ausgang des A/D-Wandler Evaluation Boards gemessenen Zeiten Hold-time  $t_h=1,5\text{ns}$  und Setup-time  $t_{su}=6\text{ns}$ , korrekt übernommen werden.

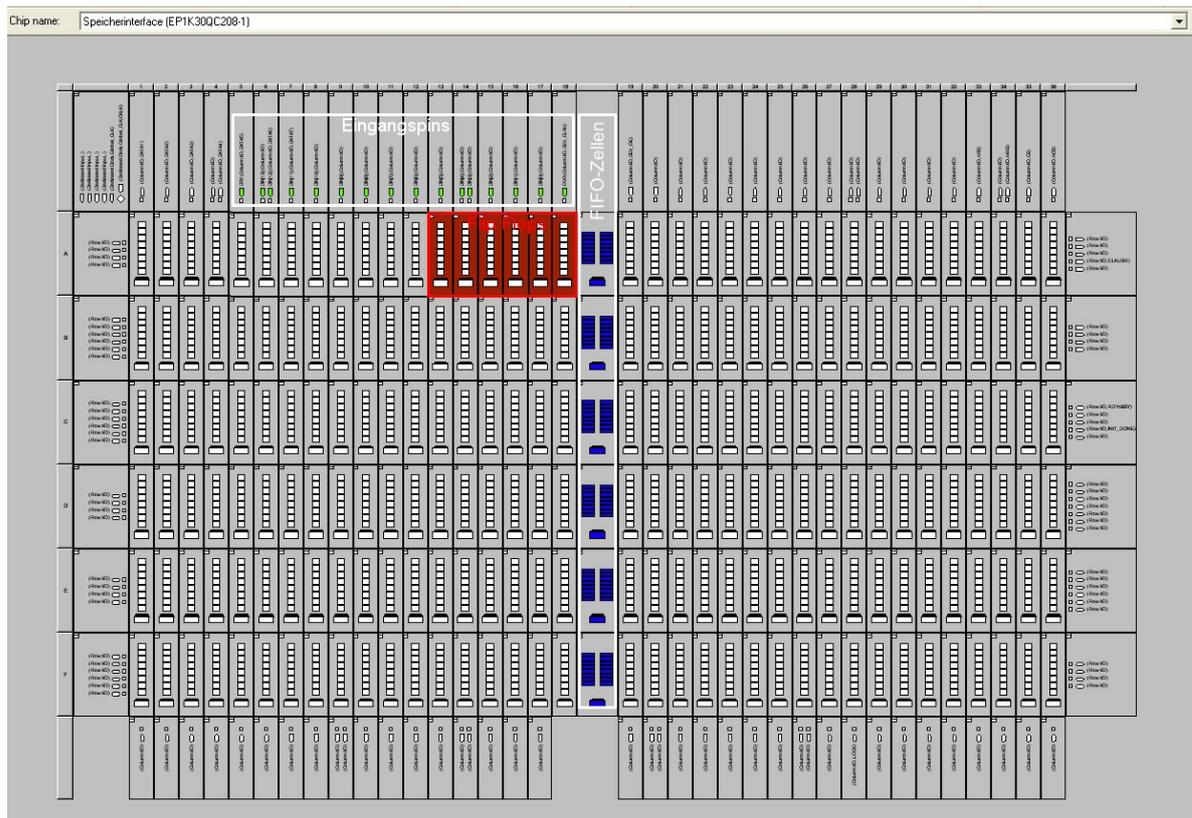


Abbildung 3.7: Platzierung der Flip Flops im FPGA

## 3.2 FIFO

Das FIFO-Speicher wurde durch die Altera Megafunktion DCFIFO realisiert, die dafür nicht Logik-Zellen, sondern das im ACEX 1K-Baustein vorhandene Dual Port-RAM verwendet. Das DC im Namen bedeutet Dual Clock, d.h. dass mit verschiedenen, nicht synchronen Schreib-Takt wrclk und Lese-Takt rdclk gearbeitet werden kann. In dieser Stufe erfolgt somit die Synchronisierung der Messdaten an den Systemtakt CLK von 33,3333 MHz. Die wichtigsten Einstellungen der Megafunktion sind in den Abbildungen 3.8 und 3.9 zu sehen. Die komplette Beschreibung der Megafunktion ist unter [22] nachzulesen.

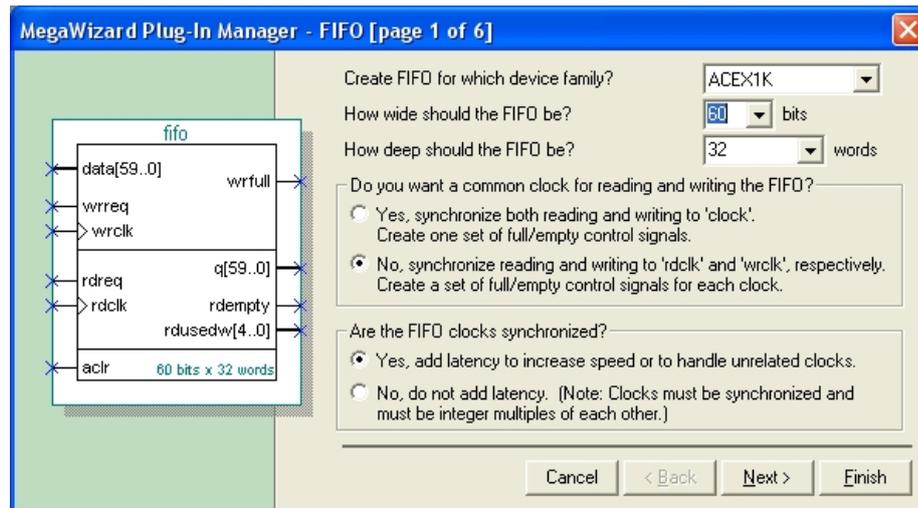


Abbildung 3.8: Altera Megafunktion FIFO Schritt 1

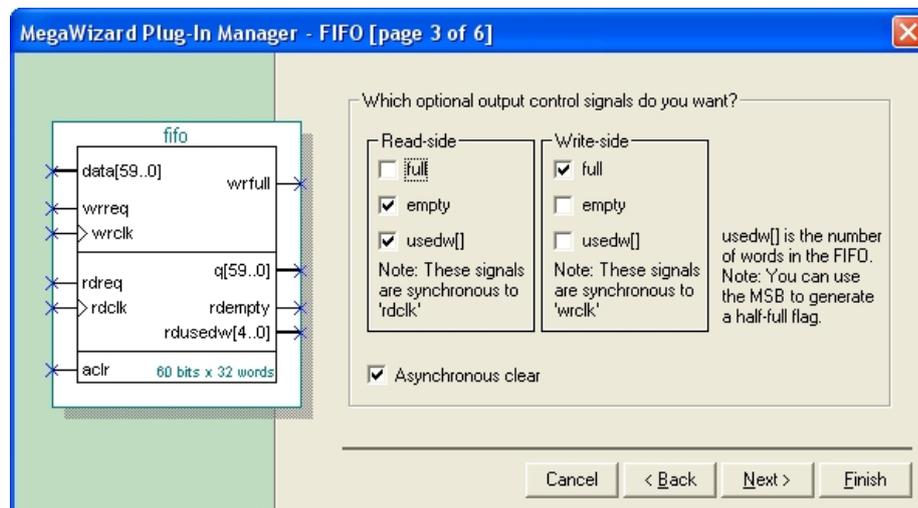


Abbildung 3.9: Altera Megafunktion FIFO Schritt 3

Die *Breite* des FIFOs ist durch das Zusammenfassen von vier 15 Bit-Messwerten mit *60 Bit* vorgegeben.

Die *Tiefe* des Speichers wird durch das Timing des SDRAMs, in das die Daten geschrieben werden, bestimmt. In das SDRAM wird mit einem Burst 8-Schreibzyklus geschrieben, d.h. es werden mit einer Adressierung des SDRAMs acht Schreibzyklen hintereinander durchgeführt. Als Refresh-Art des SDRAMs

wurde Auto Refresh gewählt, bei dem alle  $7,8125 \mu\text{s}$  ein Refresh-Zyklus mit der vom SDRAM benötigten minimalen Dauer  $t_{RC}=70 \text{ ns}$  notwendig ist, um die gespeicherten Daten zu erhalten. Bei dem gewählten Systemtakt von  $33,3333 \text{ MHz}$  entspricht das einer Dauer von drei Systemtakt, d.h.  $90 \text{ ns}$ . In der Zeit von  $90 \text{ ns}$  werden vom A/D-Wandler sieben bis acht  $15 \text{ Bit}$  Messwerte mit  $80 \text{ MHz}$  geliefert, die in der A/D-Wandler Schnittstelle (siehe Kapitel 3.1) auf zwei  $60 \text{ Bit}$  Datenworte zusammengefasst werden. Im für das FIFO schlechtesten Fall sind gerade acht Datenworte gespeichert und es wird ein Refresh durchgeführt, d.h. es müssen noch etwa zwei weitere Datenworte gespeichert werden. Zusammengenommen müsste das FIFO also etwa zehn Datenworte tief sein. Es wurde eine FIFO-Tiefe von *32 Worten* gewählt, um bei den nachfolgenden Simulationen eine bessere Analyse des Fehlverhaltens, im Falle einer Fehlfunktion bei der Datenübergabe vom FIFO an den SDRAM-Controller (siehe Kapitel 3.3), durchführen zu können.

Die Eingangsseite des FIFOs besteht aus den Datenleitungen `data[59..0]` von der A/D-Wandler Schnittstelle, dem Schreibtakt `wrclk`, der vom Takt der Eingangsdaten DRY angesteuert wird, und der Steuerleitung `wrreq`, durch `ENABLE_FIFO` von der A/D-Wandler Schnittstelle angesteuert, die angibt, wann die Daten ins FIFO übernommen werden sollen.

Ausgangsseitig arbeitet das FIFO mit dem  $33,3333 \text{ MHz}$  Systemtakt CLK an `rdclk`. Der Befehl zum Auslesen des FIFOs wird von der zentralen Steuerung (siehe Kapitel 3.5) an `rdreq` angelegt. Die Daten werden daraufhin an `q[59..0]` ausgegeben.

Damit die Steuerung (siehe Kapitel 3.5) die Kommunikation zwischen dem FIFO und dem SDRAM-Controller (siehe Kapitel 3.3) koordinieren kann, wurden als Zustandsmeldungen in der Megafunktion des FIFOs die Ausgangssignale `wrfull`, `rdempty` und `rdusedw[4..0]` eingestellt. Die ersten beiden Buchstaben dieser Signale geben an, von welchem Takt die Signale gesteuert werden (`rd` für den Lesetak `rdclk`, `wr` für den Schreibtakt `wrclk`). Das Signal `wrfull` zeigt an, dass das FIFO voll geschrieben wurde, ein Zustand der im normalen Betrieb nicht auftreten sollte. `rdempty` zeigt an, dass das FIFO komplett ausgelesen wurde und der  $5 \text{ Bit}$ -Wert `rdusedw[4..0]` wieviele Datenwerte sich im Moment im FIFO befinden.

An den asynchronen Clear-Eingängen (`aclr`) kann das FIFO von der zentralen Steuerung (siehe Kapitel 3.5) gemeinsam mit der A/D-Wandler Schnittstelle (siehe Kapitel 3.1) mit dem Signal `RESET_DATA_INPUT` in einen Anfangszustand gesetzt werden.

### 3.3 SDRAM-Controller

Die Kommunikation des Speicherinterfaces mit dem SDRAM-Modul wird vom SDRAM-Controller durchgeführt, dessen Blockschaltbild in Abbildung 3.10 zu sehen ist.

Den zentralen Teil des SDRAM-Controllers bildet die Command Engine. Diese bekommt von der Steuerung (siehe Kapitel 3.5) den Befehl, Daten an eine bestimmte Adresse im SDRAM zu schreiben oder von einer bestimmten Adresse aus dem SDRAM auszulesen, und setzt diese Befehle in die korrekten Steuersignale mit dem richtigen Timingverhalten für das SDRAM um.

Wenn Daten ins SDRAM geschrieben werden sollen, wird von der Command Engine der Tristate-Buffer durchgeschaltet und die vom FIFO (siehe Kapitel 3.2) gelieferten Daten werden an das SDRAM zur Übernahme geschickt. Da der Datenbus am SDRAM bidirektional ist, muss dieser Buffer im Lesefall hochohmig sein. Zum Schreiben wird der Burst 8-Modus verwendet, d.h. es werden mit einer Adressierung des SDRAMs acht Schreibzyklen hintereinander durchgeführt.

Werden Daten aus dem SDRAM gelesen, so liefert das SDRAM sie zu einem durch die CAS-Latency bestimmten Zeitpunkt. Damit die Daten für die weitere Verarbeitung in der Ausgangs-Schnittstelle erhalten bleiben, werden sie in einer Bank aus  $60 \text{ D-Flip Flops}$  gespeichert. Zum Lesen wird der Burst 1-Modus verwendet, d.h. es wird mit einer Adressierung des SDRAMs nur ein Lesezyklus durchgeführt.

Damit die Daten im SDRAM erhalten bleiben, liefert der Refresh-Timer alle  $7,8125 \mu\text{s}$  ein Request Signal an die Command Engine, die nach Beendigung eines laufenden Schreib- oder Lesezyklus einen Auto-Refresh Befehl an das SDRAM schickt und dem Refresh-Timer mit Acknowledge eine Rückmeldung liefert, damit der Refresh-Timer das Request Signal wieder abschalten kann.

Realisiert wurde das Blockschaltbild Abbildung 3.10 in Quartus II in einem Block Design File (BDF), das in Abbildung 3.11 zu sehen ist.

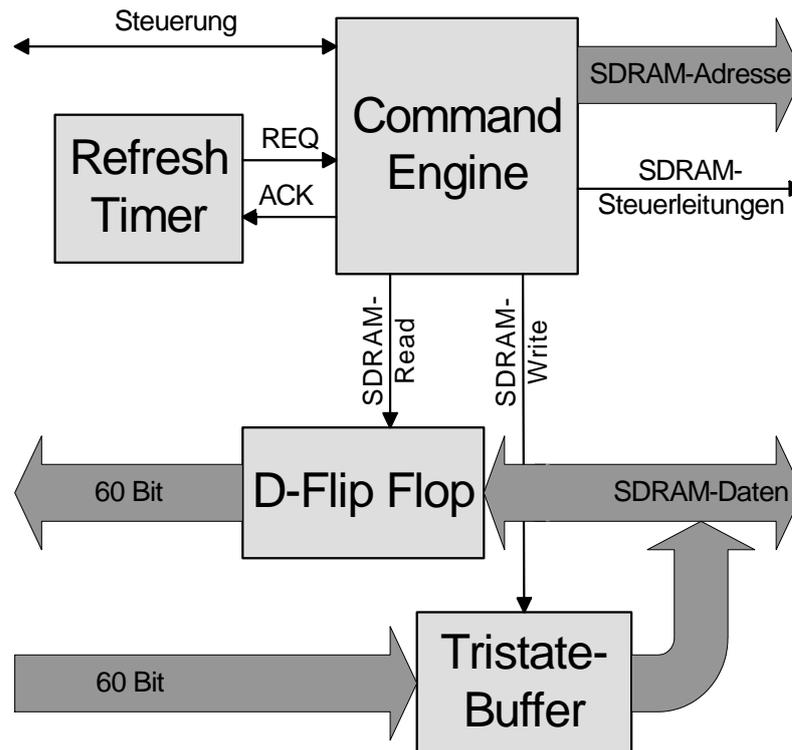


Abbildung 3.10: Blockschaltbild des SDRAM-Controller

An der linken oberen Seite des Block Design Files befinden sich die Eingänge, an der rechten oberen Seite die Ausgänge zur Kommunikation mit der Steuerung (siehe Kapitel 3.5). Mit dem Signal `SDRAM_READY=1` wird der Steuerung angezeigt, dass der SDRAM-Controller zum Schreiben oder Lesen bereit ist. Das Signal `SDRAM_READY` ist 0, wenn nach einem `RESET_SDRAM` die Initialisierung des SDRAMs noch nicht abgeschlossen ist, oder gerade ein Auto-Refresh des SDRAMs durchgeführt wird.

Falls der SDRAM-Controller bereit ist, kann die Steuerung mit `SDRAM_WRREQ=1` einen Burst 8-Schreibzyklus beginnen, der von der `SDRAM_command_engine` mit `SDRAM_WRACK=1` bestätigt wird. Durch dieses Handshake wird sichergestellt, dass die Steuerung und der SDRAM-Controller synchron arbeiten. Die Datenwerte werden an die durch `SDRAM_ADR[0..25]` angegebene Adresse und die sieben nachfolgenden Adressen geschrieben.

Mit `SDRAM_RDREQ=1` kann die Steuerung einen Burst 1-Lesezyklus starten, der von der `SDRAM_command_engine` mit `SDRAM_RDACK=1` bestätigt wird. Dadurch wird wieder sicher gestellt, dass die Steuerung und der SDRAM-Controller synchron arbeiten. Der Datenwert wird an die durch `SDRAM_ADR[0..25]` angegebene Adresse geschrieben.

Der Tristate-Buffer `lpm_bustri1 (inst)` wurde mit der Altera Megafunction `LPM_BUSTRI` realisiert. In Abbildung 3.12 sind die Einstellungen für einen unidirektionalen 60 Bit-Buffer zu sehen. Die Bank aus 60 D-Flip Flops `lpm_FlipFlop2 (inst2)` wurde wie in der A/D-Wandler Schnittstelle (siehe Kapitel 3.1) mit der Altera Megafunction `LPM_FF` realisiert.

Die beiden Blöcke `SDRAM_refresh_timer (inst0)` und `SDRAM_command_engine (inst1)` wurden komplett in VHDL programmiert.

Die Funktion des `SDRAM_refresh_timer (inst0)` wird anhand des Flussdiagrammes Abbildung 3.13 beschrieben.

Die im Flussdiagramm dargestellte Funktion wird durch eine Zustandsänderung der Signale `RESET` oder `CLK` ausgelöst.

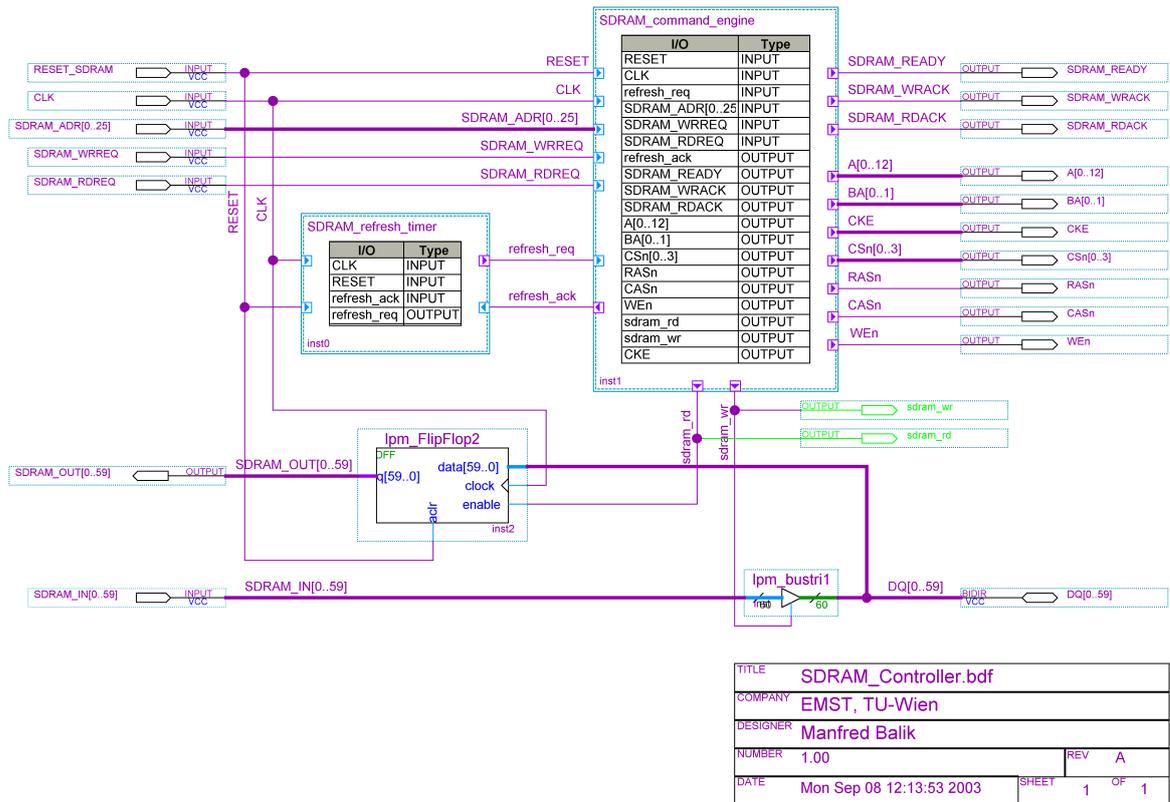


Abbildung 3.11: Quartus II Block Design File des SDRAM-Controllers

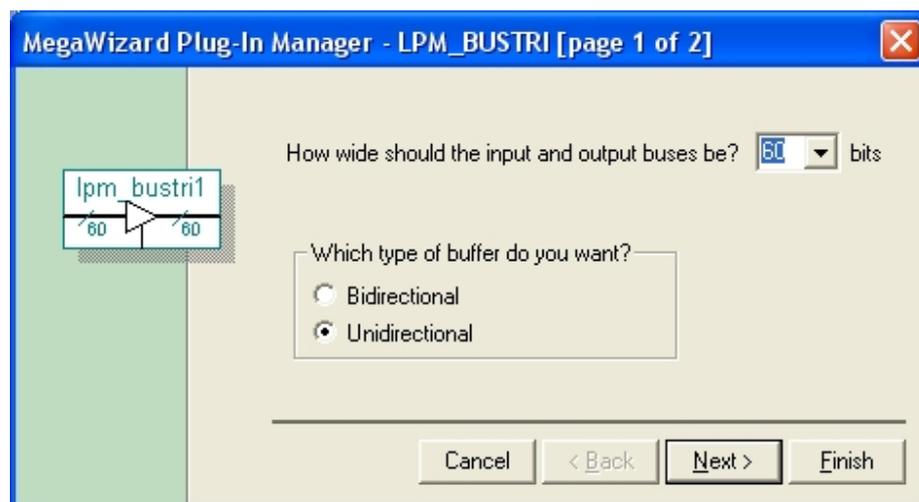


Abbildung 3.12: Altera Megafunction LPM\_BUSTRI

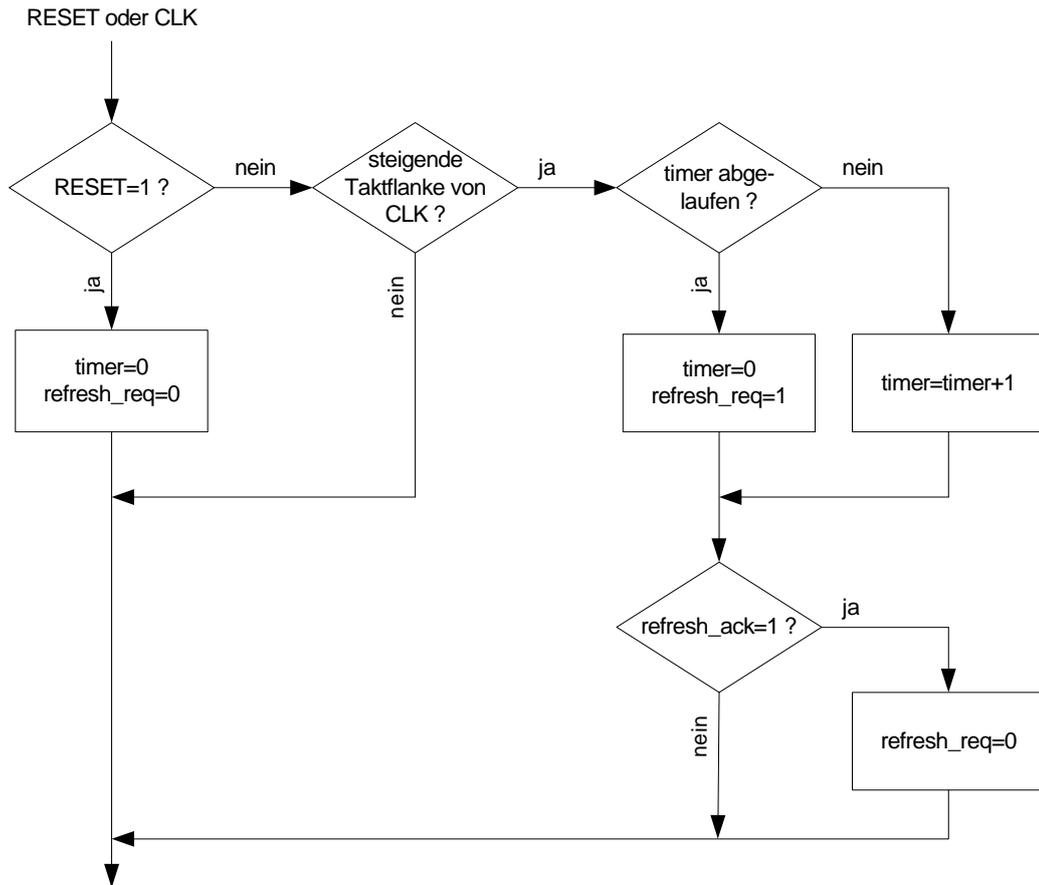


Abbildung 3.13: Flussdiagramm des Refresh Timers

Wird mit `RESET=1` ein Reset ausgelöst, so wird der Timer, der die Taktzyklen bis zum nächsten Refresh zählt, und das Signal `refresh_req` an die `SDRAM_command_engine` auf 0 gesetzt.

Ändert sich das `CLK`-Signal, so muss zwischen steigender und fallender Taktflanke unterschieden werden. Bei einer fallenden Taktflanke wird keine Funktion ausgeführt, bei einer steigenden Taktflanke wird überprüft, ob der Timer abgelaufen ist, d.h. die  $7,8125 \mu\text{s}$  bis zum nächsten Refresh schon vergangen sind. Ist dies der Fall, so wird mit `refresh_req=1` eine Refresh-Aufforderung an die `SDRAM_command_engine` ausgegeben und der Timer wieder auf null gesetzt. Andernfalls wird der Timer um eins erhöht. Bestätigt die `SDRAM_command_engine` mit `refresh_ack=1`, dass sie die Refresh-Aufforderung bekommen hat, so wird `refresh_req` wieder auf 0 gesetzt.

Die Funktion des `SDRAM_command_engine (inst1)` wird anhand des Flussdiagrammes Abbildung 3.14 beschrieben.

Die im Flussdiagramm beschriebene Funktion wird durch eine Zustandsänderung der Signale `RESET` oder `CLK` ausgelöst.

Wird ein Reset mit `RESET=1` ausgelöst, so werden alle Variablen und Ausgangssignale initialisiert und die Variable `state`, die den Zustand der State-Machine, die die Funktionsabläufe beschreibt, auf `SDRAM_NOP` gesetzt.

Bei einer fallenden Flanke des `CLK`-Signals wird keine Funktion ausgeführt, bei einer steigenden Taktflanke wird abhängig vom aktuellen Zustand der State-Machine fortgefahren. In den einzelnen ausprogrammierten Befehlen wurden jeweils die zur Ansteuerung des `SDRAM`-Moduls nötigen Steuerleitungen gesetzt und das notwendige Timingverhalten realisiert.

Bevor auf die Ausprogrammierung der einzelnen Befehle näher eingegangen wird, folgen einige zum

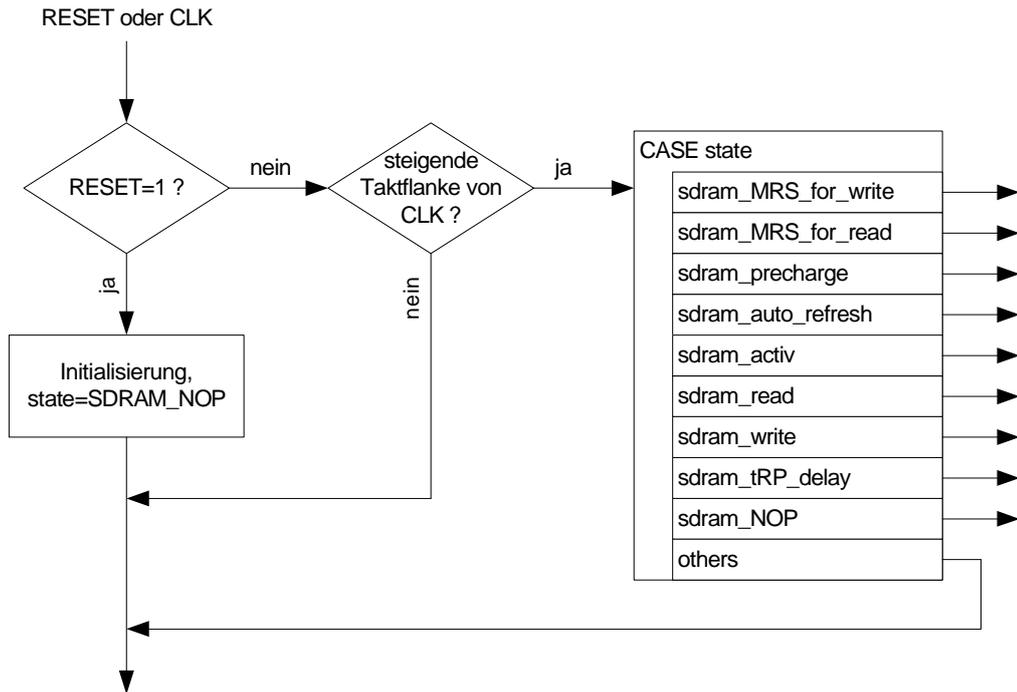


Abbildung 3.14: Flussdiagramm der Command Engine

Verständnis des SDRAMs notwendige Erklärungen.

Der Aufbau und die Funktion eines SDRAMs und die zur Ansteuerung notwendigen Steuerleitungen und Timingdiagramme können im Buch [3] genau nachgelesen werden. Zum Verständnis des Timingverhaltens sind weiters die Datenblätter [23] und [24] hilfreich. Das exakte Zeitverhalten des verwendeten SDRAM-Moduls findet man in [25].

Das 512 MByte große SDRAM-Modul hat eine Datenbusbreite von 64 Bit, d.h. es werden mit einer Adressierung 8 Bytes geschrieben oder gelesen. Somit ist der Adressbereich des SDRAMs  $\frac{512 \cdot 1024^2}{8} = 67\,108\,864$  Adressen groß. Um diesen Adressbereich ansprechen zu können, benötigt man 26 Bit, welche, wie in Tabelle 3.1 zu sehen ist, auf die einzelnen Steuerleitungen des SDRAMs aufgeteilt werden.

Adress Bit	Bedeutung	SDRAM Steuerleitung
25	Chip Address	CS0 bis CS3
23 - 24	Bank Address	BA0 bis BA1
10 - 22	Row Address	A0 bis A12
0 - 9	Column Address	A0 bis A9

Tabelle 3.1: Aufteilung der Adress Bits auf die SDRAM Steuerleitungen

Aus der Vielzahl der möglichen Befehle an ein SDRAM wurden nur die zur Funktion des Speicherinterfaces benötigten ausprogrammiert. Die Zustände der einzelnen Steuerleitungen des SDRAMs für den jeweiligen Befehl sind in der Tabelle 3.2 zu sehen.

Befehl	RAS	CAS	WE	BA	A
MRS	0	0	0	0	OP-Code
Precharge all Banks	0	1	0	X	A <sub>10</sub> /AP=1
Auto Refresh	0	0	1	X	X
Row activate	0	1	1	Bank Address	Row Address
Write with Auto Precharge	1	0	0	Bank Address	Column Address, A <sub>10</sub> /AP=1
Read with Auto Precharge	1	0	1	Bank Address	Column Address, A <sub>10</sub> /AP=1
NOP	1	1	1	X	X

Tabelle 3.2: Zustände der Steuerleitungen für die SDRAM-Befehle

Die einzelnen Befehle an das SDRAM sind in der ersten Spalte zu sehen. Beim Mode Register Set *MRS* wird an die Adressleitungen A der Operation-Code OP angelegt, der den Modus des SDRAMs bestimmt. Die Bedeutung der einzelnen Bits des OP-Codes ist in [22] und [23] erklärt. Bei *Precharge all Banks* muss das Adressbit 10, das auch Auto Precharge AP genannt wird, auf 1 gesetzt werden. Es ist zu beachten, dass sich ein Write- und Read-Befehl aus zwei Teilbefehlen zusammensetzt. Im ersten Schritt eines Write- oder Read-Befehls wird die entsprechende SDRAM-Bank mit *Row activate* aktiviert und die Zeilenadresse (Row Address) über die Adressleitungen übergeben. Im zweiten Schritt wird die Spaltenadresse (Column Address) über die Adressleitungen übergeben, und durch das gesetzte Adressbit 10 wird automatisch ein Precharge nach dem Schreib- oder Lesebefehl durchgeführt.

Nach dem Einschalten der Stromversorgung benötigt ein SDRAM eine sogenannte Startupsequenz, bevor darauf zugegriffen werden kann. Diese Startupsequenz muss wie folgt aussehen:

für mindestens 200  $\mu$ s NOP  
 Precharge all Banks  
 8 Refresh Zyklen  
 Mode Register Set MRS  
 NOP

Die im Flussdiagramm Abbildung 3.14 dargestellten Befehle haben folgende Funktion:

- **s dram \_ MRS \_ for \_ write:** Durch diesen Befehl wird das SDRAM in den Burst 8 Mode gesetzt, der für Schreibzyklen verwendet wird.  
 Mit dem ersten Taktzyklus wird der MRS Befehl ausgelöst und der OP-Code über die Adressleitungen übergeben. Nach dem Abwarten der Zeit “MRS to Ready delay”  $t_{MRD}$  wird in den State-Machine-Status `s dram _ NOP` gewechselt.
- **s dram \_ MRS \_ for \_ read:** Durch diesen Befehl wird das SDRAM in den Burst 1 Mode gesetzt, der für Lesezyklen verwendet wird.  
 Mit dem ersten Taktzyklus wird der MRS Befehl ausgelöst und der OP-Code über die Adressleitungen übergeben. Nach dem Abwarten der Zeit “MRS to Ready delay”  $t_{MRD}$  wird in den State-Machine-Status `s dram _ NOP` gewechselt.
- **s dram \_ precharge:** Durch diesen Befehl werden alle Bänke des SDRAMs geladen.  
 Mit dem ersten Taktzyklus wird der “Precharge all Banks” Befehl ausgelöst. Nach dem Abwarten der Zeit “RAS Precharge time”  $t_{RP}$  wird in den State-Machine-Status `s dram _ autorefresh` gewechselt. Dieser Befehl wird nur in der Startupsequenz benötigt, weil nach Schreib- und Lese-Befehlen ein automatisches Precharge erfolgt.
- **s dram \_ autorefresh:** Durch diesen Befehl wird ein Autorefresh im SDRAM ausgelöst.  
 Mit dem ersten Taktzyklus wird der Autorefresh Befehl ausgelöst. Nach dem Abwarten der Zeit “Row refresh cycle time”  $t_{RFC}$  wird überprüft, ob man sich in der Startupsequenz befindet und das Autorefresh wiederholt bis acht Refreshzyklen durchgeführt wurden. Danach wird in den State-Machine-Status `s dram _ MRS _ for _ write` gewechselt. Befindet man sich nicht in der Startupsequenz, so wird sofort in den Status `s dram _ NOP` gewechselt.

- **s dram \_ activate**: Mit diesem Befehl wird eine Zeile im SDRAM aktiviert. Dieser Befehl ist für Schreib- und Lese-Befehle identisch.  
Mit dem ersten Taktzyklus wird der “Row activate” Befehl ausgelöst und die Zeilen-Adresse über die Adressleitungen übergeben. Nach dem Abwarten der Zeit “RAS to CAS delay”  $t_{RCD}$  wird in den State-Machine Status `s dram _ NOP` gewechselt.
- **s dram \_ read**: Durch diesen Befehl wird ein Datenwert aus dem SDRAM ausgelesen.  
Mit dem ersten Taktzyklus wird der “Read with Auto Precharge” Befehl ausgelöst und die Spalten-Adresse über die Adressleitungen übergeben. Nach dem Abwarten der CAS-Latency wird in den State-Machine-Status `s dram _ delay _ tRP` gewechselt.
- **s dram \_ write**: Durch diesen Befehl werden acht Datenwert in das SDRAM geschrieben.  
Mit dem ersten Taktzyklus wird der “Write with Auto Precharge” Befehl ausgelöst und die Spalten-Adresse über die Adressleitungen übergeben. Nachdem die acht Datenwerte geschrieben wurden, wird in den State-Machine-Status `s dram _ delay _ tRP` gewechselt.
- **s dram \_ delay \_ tRP**: Dieser Befehl dient zur Zeitverzögerung, um das SDRAM-Timing am Ende eines Lese- oder Schreib-Zyklus einzuhalten.  
Nach dem Abwarten der Zeit “RAS Precharge time”  $t_{RP}$  wird in den State-Machine-Status `s dram _ NOP` gewechselt.
- **s dram \_ NOP**: In diesem Befehl wird nicht nur der NOP-Befehl an das SDRAM ausgegeben, sondern er stellt auch den zentralen Status der State-Machine dar, von dem aus die anderen Befehle aufgerufen werden.
  - Wenn sich das SDRAM in der Startupsequenz befindet, wird  $200 \mu s$  gewartet und in den State-Machine-Status `s dram _ precharge` gewechselt.
  - Ist vom `SDRAM _ refresh _ timer (inst0)` die Aufforderung für ein Autorefresh mit `refresh _ req=1` gekommen, so wird mit `refresh _ ack=1` bestätigt und in den State-Machine-Status `s dram _ autorefresh` gewechselt.
  - Falls die Steuerung mit `SDRAM _ WRREQ=1` einen Schreibzyklus initiiert, wird dieser mit `SDRAM _ WRACK=1` bestätigt. Aus Geschwindigkeitsgründen wird sofort die der übergebenen Adresse entsprechende Zeile im SDRAM aktiviert und in den State-Machine-Status `s dram _ activate` gewechselt.
  - Falls die Steuerung mit `SDRAM _ RDREQ=1` einen Lesezyklus initiiert, wird dieser mit `SDRAM _ RDACK=1` bestätigt. Aus Geschwindigkeitsgründen wird sofort die der übergebenen Adresse entsprechende Zeile im SDRAM aktiviert und in den State-Machine-Status `s dram _ activate` gewechselt.
  - Wird keine der angeführten Aktionen durchgeführt, so wird NOP an das SDRAM ausgegeben.

### 3.4 Ausgangs-Schnittstelle

In der Ausgangs-Schnittstelle werden die vier, im SDRAM zu einem Datenwort zusammengefassten Messwerte, hintereinander an den Ausgang ausgegeben. Abbildung 3.15 zeigt ein Blockschaltbild dieser Funktion.

Werden Daten vom SDRAM-Controller (siehe Kapitel 3.3) aus dem SDRAM gelesen, so liefert das SDRAM sie zu einem durch die CAS-Latency bestimmten Zeitpunkt. Damit die Daten für die weitere Verarbeitung in der Ausgangs-Schnittstelle erhalten bleiben, werden sie dort in einer Bank aus 60 D-Flip Flops gespeichert. In der Ausgangs-Schnittstelle sind für die Messdaten deshalb nur mehr vier voneinander getrennt ansteuerbare 15 Bit *Tristate-Buffer* notwendig. *Tristate-Buffer* müssen verwendet werden, damit mehrere Speicherinterfaces parallel an den selben Ausgabebus geschaltet werden können. Von der Steuerung (siehe Kapitel 3.5) wird bei der am Ausgang angeschlossene Ausleseschaltung (z.B. eine IO-Karte

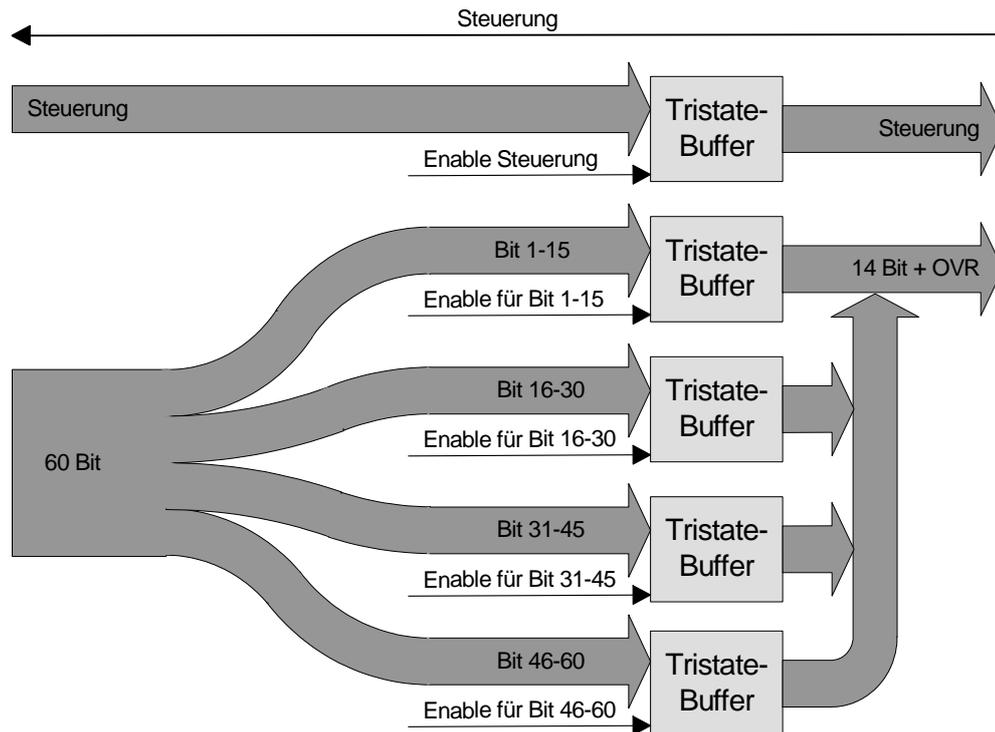


Abbildung 3.15: Blockschaltbild der Ausgangs-Schnittstelle

in einem PC) mit einem Request angefragt, ob sie zur Datenaufnahme bereit ist. Bestätigt die Ausleseschaltung, so werden die Steuerleitungen und der erste Tristate-Buffer von der Steuerung an den Ausgang durchgeschaltet. Mit jedem weiteren Ausgabezyklus wird der nächste Tristate-Buffer durchgeschaltet und der vorige wieder hochohmig geschaltet. Nach dem vierten Ausgabezyklus wird von der Steuerung ein neues Datenwort aus dem SDRAM gelesen, und dessen Ausgabe durch die Ausgangs-Schnittstelle erfolgt genauso.

Realisiert wurde das Blockschaltbild Abbildung 3.15 in Quartus II in einem Block Design File (BDF), das in Abbildung 3.16 zu sehen ist.

Die Schnittstelle zur Steuerung befindet sich in der Abbildung 3.16 auf der linken Seite, die Ausgangspins der Ausgangs-Schnittstelle auf der rechten.

Die Steuerleitungen `DATA_AVAILABLE` und `DOUT_RDREQ` sind Signale von der Steuerung an den Ausgang und können von der Steuerung mit dem Signal `ENABLE_OUT_signals` über einen Tristate-Buffer durchgeschaltet werden. Dieser Tristate-Buffer `lpm_bustri2` (`inst7`) wurde genauso wie im SDRAM-Controller mit der Altera Megafunktion `LPM_BUSTRI` realisiert. Die in VHDL programmierten Blöcke `zusammenfassen_signale` (`inst8`) und `aufspalten_signale` (`inst9`) dienen dazu, die zwei Steuerleitungen mit nur einem Tristate-Buffer gemeinsam schalten zu können. Die Steuerleitung `DOUT_RDACK` ist ein Signal von der Ausleseschaltung an die Steuerung und wird mit Hilfe zweier Flip Flops (`inst17` und `inst19`) an den Systemtakt `CLK` synchronisiert.

Im in VHDL programmierten Block `aufspalten_out` (`inst`) wird das 60 Bit Datenwort vom SDRAM-Controller in die vier 15 Bit Messwerte `data1[0..14]`, `data2[0..14]`, `data3[0..14]` und `data4[0..14]` aufgeteilt. Jeder dieser vier Messwerte kann von der Steuerung mit den Signalen `ENABLE_OUT_data1`, `ENABLE_OUT_data2`, `ENABLE_OUT_data3` und `ENABLE_OUT_data4`, die mit AND-Gattern (`inst5`, `inst6`, `inst10` und `inst11`) mit dem Signal `ENABLE_OUT_signals` UND-verknüpft sind, über einen Tristate-Buffer `lpm_bustri0` (`inst1`, `inst2`, `inst3` und `inst4`) an den Ausgang durchgeschaltet werden. Diese UND-Verknüpfung ist notwendig, damit die Steuerung mit `ENABLE_OUT_signals=0` alle Ausgangstreiber gemeinsam hochohmig schalten kann. Die Tristate-Buffer wurden wieder mit der Altera Megafunktion `LPM_BUSTRI` realisiert.

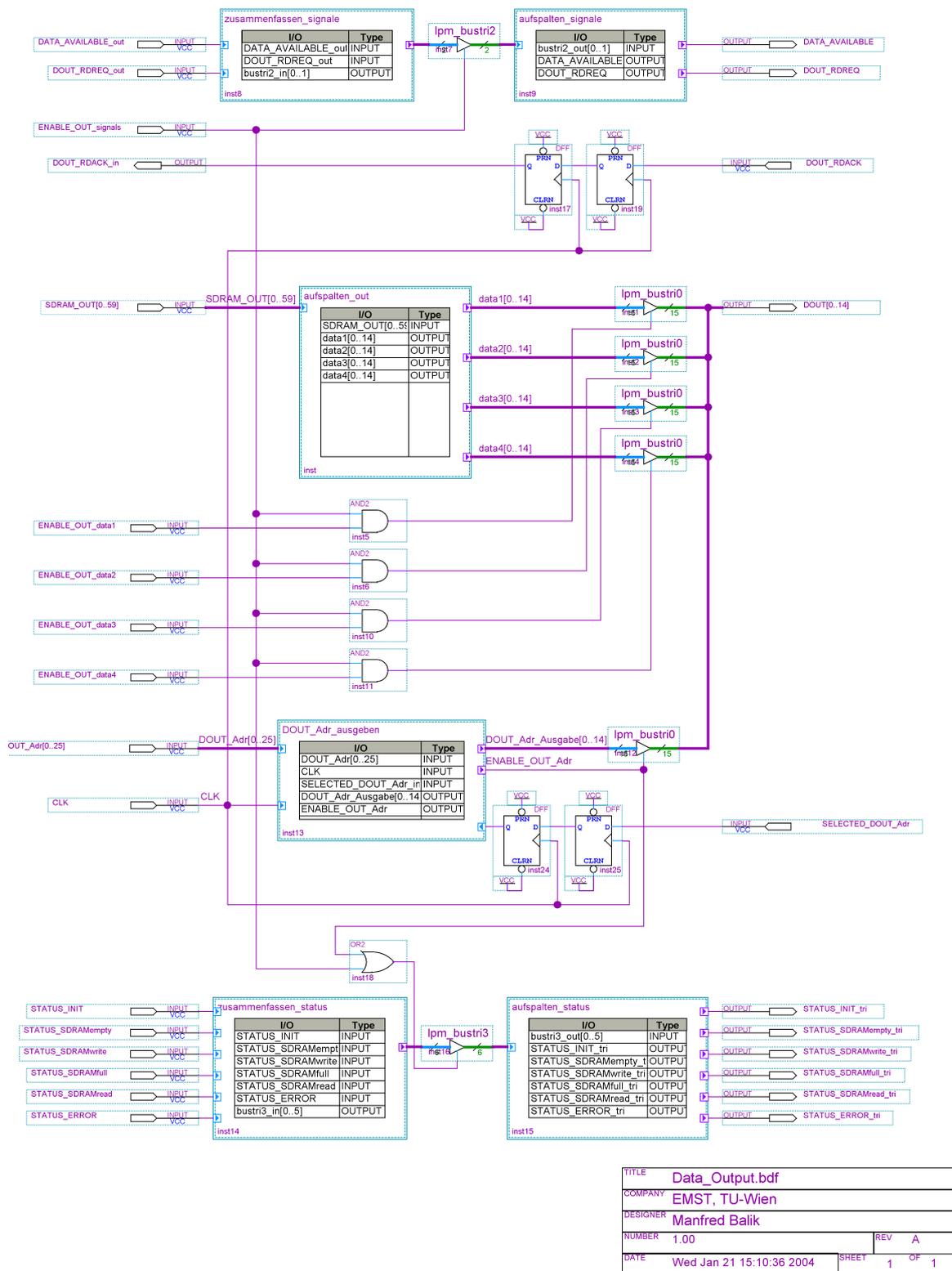


Abbildung 3.16: Quartus II Block Design File der Ausgangs-Schnittstelle

Durch das Signal `SELECTED_DOUT_Adr` vom Ausgang wird eine Zusatzfunktion des Speicherinterfaces realisiert. Damit kann, während oder nach der Aufnahme der Messwerte vom A/D-Wandler, die Anzahl der im SDRAM gespeicherten Messwerte an den Ausgang ausgegeben werden. Diese Funktion wird vom Block `DOUT_Adr_ausgeben` (`inst13`) bewerkstelligt, der vom Adresszähler für das SDRAM `OUT_Adr[0..25]`, der von der Steuerung geliefert wird, die höchstwertigen 15 Bit als `DOUT_Adr_Ausgabe[0..14]` an den Tristate-Buffer `lpm_bustri0` (`inst12`) ausgibt. Die Steuerleitung `SELECTED_DOUT_Adr` wird mit Hilfe zweier Flip Flops (`inst24` und `inst25`) an den Systemtakt `CLK` synchronisiert. Falls `SELECTED_DOUT_Adr=1` ist, wird vom Block `DOUT_Adr_ausgeben` (`inst13`) der Tristate-Buffer mit dem Enable-Signal `ENABLE_OUT_Adr` durchgeschaltet. Es ist zu beachten, dass `SELECTED_DOUT_Adr` nicht während des Auslesens der Messwerte verwendet werden darf, d.h. nicht im Status `SDRAMread` (siehe nächster Absatz bzw. Steuerung) verwenden!

Mit den Status-Leitungen `STATUS_INIT`, `STATUS_SDRAMempty`, `STATUS_SDRAMwrite`, `STATUS_SDRAMfull`, `STATUS_SDRAMread` und `STATUS_ERROR` zeigt die Steuerung den momentanen Zustand des Speicherinterfaces an (siehe Steuerung Kapitel 3.5). Die in VHDL programmierten Blöcke `zusammenfassen_status` (`inst14`) und `aufspalten_status` (`inst15`) dienen dazu, diese Status-Leitungen mit nur einem Tristate-Buffer `lpm_bustri3` (`inst14`) gemeinsam an den Ausgang schalten zu können. Der Tristate-Buffer wird durchgeschaltet, falls `ENABLE_OUT_signals` von der Steuerung oder `ENABLE_OUT_Adr` gesetzt sind. Diese ODER-Verknüpfung wird mit dem OR-Gatter (`inst18`) durchgeführt.

### 3.5 Steuerung

Die Steuerung koordiniert das Zusammenspiel aller Funktionseinheiten, die in den vorangegangenen Kapiteln 3.1 bis 3.4 erklärt wurden. Sie wurde komplett in VHDL programmiert. Die verschiedenen Zustände des Speicherinterfaces wurden als State-Machines ausprogrammiert. In Abbildung 3.17 ist ein Flussdiagramm dieser Funktion dargestellt.

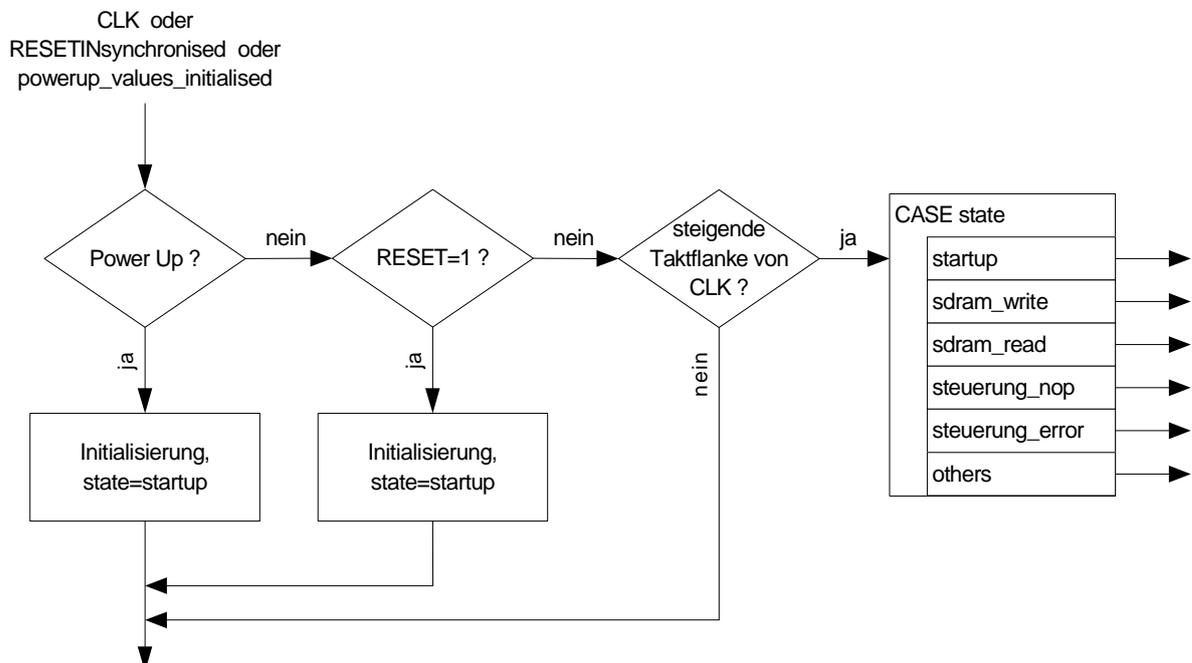


Abbildung 3.17: Flussdiagramm der Steuerung

Die im Flussdiagramm dargestellte Funktion wird durch eine Zustandsänderung der Signale `RESETinsynchronised`, `CLK` oder `powerup_values_initialised` ausgelöst.

Durch das Einschalten der Stromversorgung wird die Boolean-Variable `powerup_values_initialised` auf `false` initialisiert, weshalb beim ersten Durchlaufen das Flussdiagramm im Zweig senkrecht ganz links durchlaufen wird. Nach der Initialisierung aller Signale wird für den nächsten Durchlauf in den State-Machine-Zustand `startup` gewechselt.

Im Falle eines Resets werden ebenfalls alle Signale initialisiert und auch in den State-Machine-Zustand `startup` gewechselt.

Ändert sich das CLK-Signal, so muss zwischen steigender und fallender Taktflanke unterschieden werden. Bei einer fallenden Flanke des CLK-Signals wird keine Funktion ausgeführt, bei einer steigenden Taktflanke wird abhängig vom aktuellen Zustand der State-Machine fortgefahren.

Bevor auf die Ausprogrammierung der einzelnen Zustände der State-Machine näher eingegangen wird, folgen einige zum Verständnis notwendige Erklärungen.

Die Kommunikation mit der an der Ausgangs-Schnittstelle angeschlossenen I/O-Karte erfolgt im Handshake-Verfahren. Befinden sich Messdaten von einer abgeschlossenen Messung im SDRAM und ist das Speicherinterface mit dem Signal `SELECTED` (siehe Abbildung 3.3) ausgewählt, wird der I/O-Karte mit dem Signal `DATA_AVAILABLE` angezeigt, dass Daten zum Auslesen vorhanden sind. Weiters gibt das Speicherinterface mit dem Signal `DOUT_RDREQ` ein Request an die I/O-Karte aus, wenn die Ausgangsdaten am Ausgang `DOUT[0..14]` bereitstehen. Die I/O-Karte bestätigt mit einem Acknowledge mit dem Signal `DOUT_RDACK` die Übernahme der Daten. Dieses Handshake wird wiederholt, bis alle Daten ausgelesen sind und `DATA_AVAILABLE` von der Steuerung wieder 0 gesetzt wird. Die Datenausgabe kann jederzeit durch `SELECTED=0` unterbrochen werden und mit `SELECTED=1` an der selben Stelle fortgesetzt werden.

Die Steuerung zeigt mit den Leitungen `STATUS_INIT`, `STATUS_SDRAMempty`, `STATUS_SDRAMwrite`, `STATUS_SDRAMfull`, `STATUS_SDRAMread` und `STATUS_ERROR` den momentanen Zustand des Speicherinterfaces an. Sie werden von der State-Machine der Steuerung entsprechend gesetzt. Die Anzeige dieser Zustände erfolgt einerseits direkt an Pins des FPGAs, an die Leuchtdioden über Treiber angeschlossen sind (siehe Kapitel 5), andererseits über Tristate-Buffer in der Ausgangs-Schnittstelle, wo sie nur, wenn das Speicherinterface mit `SELECTED` oder `SELECTED_DOUT_Adr` ausgewählt ist, ausgegeben werden.

In Tabelle 3.3 sind die Bedeutungen aller möglichen Anzeigen der Status-Leitungen zusammengefasst.

STATUS_INIT	STATUS_SDRAMempty	STATUS_SDRAMwrite	STATUS_SDRAMfull	STATUS_SDRAMread	STATUS_ERROR	Bedeutung
1	0	0	0	0	0	Initialisierung des Speicherinterfaces
0	1	0	0	0	0	Speicherinterface wartet auf Messwerte vom A/D-Wandler
0	0	1	0	0	0	Messwerte vom A/D-Wandler werden ins SDRAM geschrieben
0	0	0	1	0	0	Messung beendet, Speicherinterface wartet auf Ausgabe der Messwerte
0	0	0	0	1	0	Ausgabe der Messwerte aus dem SDRAM an den Ausgang
0	0	0	0	0	1	Fehlerzustand
0	0	1	0	0	1	Fehlerzustand beim Schreiben ins SDRAM
0	0	0	0	1	1	Fehlerzustand beim Lesen aus dem SDRAM
0	0	0	0	0	0	Ausgabe der Messwerte aus dem SDRAM an den Ausgang erfolgreich beendet

Tabelle 3.3: Bedeutung der Status-Leitungen

Von den in der Tabelle erwähnten Fehlerzustände kann der allgemeine Fehlerzustand nur erreicht werden, wenn das FIFO überläuft, d.h. die vom A/D-Wandler gemessenen Daten zu langsam oder gar nicht ins SDRAM geschrieben werden, was auf eine Fehlfunktion des SDRAM-Controllers hindeutet. Die

beiden weiteren Fehlerzustände können nur durch eine Fehlfunktion der State-Machine, die zum Beispiel durch unsynchronisierte Eingangssignale ausgelöst werden kann, erreicht werden.

Für die im Flussdiagramm ganz rechts dargestellten Zustände der State-Machine werden folgende Funktionen ausgeführt. Jede dieser Funktionen wurde mit Hilfe von einer weiteren State-Machine “count” realisiert.

- **startup:** Die Status-Leitung `STATUS_INIT` wird 1 gesetzt. Damit keine Messdaten aufgenommen werden, bis das SDRAM fertig initialisiert ist, wird das Signal `SDRAM_READY` überprüft und die A/D-Wandler Schnittstelle (siehe Kapitel 3.1) von der Steuerung solange im Reset-Zustand gehalten. Ist das SDRAM betriebsbereit, wird statt `STATUS_INIT` die Status-Leitung `STATUS_SDRAMempty=1` gesetzt und in den State-Machine-Status `sdr_am_write` gewechselt.
- **sdr\_am\_write:** Mit dem ersten Zustand der State-Machine “count” dieses Befehls werden diverse Zustände überprüft:
  - Wenn das SDRAM schon voll geschrieben ist, d.h. der Adresszähler 67.108.863 erreicht hat, wird die Status-Leitung `STATUS_SDRAMfull=1` gesetzt und in den State-Machine-Status `sdr_am_read` gewechselt.
  - Wenn das FIFO voll ist, d.h. das Signal `wrfull` vom FIFO 1 gesetzt wurde, konnten die eingehenden Messwerte vom A/D-Wandler nicht schnell genug ins SDRAM geschrieben werden. Dieser Fehlerzustand wird durch das Setzen der Status-Leitungen `STATUS_SDRAMwrite` und `STATUS_ERROR` angezeigt und in den State-Machine-Status `steuerung_error` gewechselt.
  - Wenn im FIFO genug Messdaten für einen Burst 8-Schreibzyklus ins SDRAM vorhanden sind, d.h. die Anzeige für die Werte im FIFO `rdusedw` grösser als 7 ist, wird beim SDRAM-Controller mit `SDRAM_WRREQ` ein Schreibzyklus angefragt und die Speicheradresse dafür übergeben. Dieser Schreib-Zustand wird mit der Status-Leitung `STATUS_SDRAMwrite=1` angezeigt und in den nächsten Zustand für `count` gewechselt.
  - Wenn die Messung des A/D-Wandlers beendet ist, d.h. die Anzeige für die Werte im FIFO `rdusedw` sich für 100 Taktzyklen nicht geändert hat, wird der Adresszähler des SDRAMs für den späteren Auslesevorgang gespeichert, die Status-Leitung `STATUS_SDRAMfull` gesetzt und in den State-Machine-Status `sdr_am_read` gewechselt.

Im zweiten Zustand der State-Machine “count” wird auf die Antwort des SDRAM-Controllers mit `SDRAM_WRACK` gewartet und danach das Auslesen des FIFOs mit `FIFO_RDREQ=1` gestartet. Dadurch wird erreicht, dass die Steuerung und der SDRAM-Controller synchron arbeiten, damit die Datentübergabe vom FIFO an den SDRAM-Controller funktioniert.

In den weiteren Zustände der State-Machine “count” wird gewartet bis die acht Datenwerte vom FIFO ins SDRAM geschrieben werden, und dann das Auslesen des FIFOs mit `FIFO_RDREQ=0` beendet und der Adresszähler um acht erhöht. Nach dem Abwarten der Zeit, die das SDRAM benötigt, um für den nächsten Schreibbefehl bereit zu sein, wird wieder in den Anfangszustand von `sdr_am_write` gewechselt.

Falls sich die State-Machine des `sdr_am_write`-Befehls durch ein Fehlverhalten in keinem definierten Zustand befindet, wird in den State-Machine-Status des Fehlerzustandes `steuerung_error` gewechselt und die Status-Leitungen `STATUS_SDRAMwrite` und `STATUS_ERROR` werden 1.

- **sdr\_am\_read:** Wenn das Speicherinterface mit `SELECTED=1` ausgewählt wurde, müssen alle Ausgangssignale an die Ausgangs-Schnittstelle ausgegeben werden. Zum Durchschalten der Tri-State-Buffer in der Ausgangsschnittstelle wird das Signal `ENABLE_OUT_signals=1` gesetzt.

Mit der weiteren State-Machine “which\_out” wird ausgewählt, welcher der vier Messwerte aus dem 60 Bit Datenwort des SDRAMs gerade ausgegeben wird. Im ersten Zustand von “which\_out” muss ein Wert aus dem SDRAM gelesen werden:

- Mit dem ersten Zustand der State-Machine “count” dieses Befehls wird überprüft, ob schon alle Werte aus dem SDRAM ausgelesen und am Ausgang ausgegeben wurden. Ist dies der Fall, so werden alle Status-Leitungen 0 gesetzt und in den State-Machine-Status `steuerung_nop` gewechselt.

- Sind noch auszulesende Daten im SDRAM, so wird dies mit `DATA_AVAILABLE_out=1` an den Ausgang signalisiert, die Status-Leitung `STATUS_SDRAMread` gesetzt und eine Leseanfrage an den SDRAM-Controller mit `SDRAM_RDREQ` gestellt.
- Im zweiten Zustand der State-Machine “count” wird auf die Antwort des SDRAM-Controllers mit `SDRAM_RDACK` gewartet. Dadurch wird erreicht, dass die Steuerung und der SDRAM-Controller synchron arbeiten, damit die Datenübergabe vom SDRAM-Controller an die Ausgangs-Schnittstelle funktioniert.
- In den weiteren Zustände der State-Machine “count” wird nach dem Abwarten der CAS-Latency der erste Messwert des aus dem SDRAM ausgelesenen 60 Bit Datenworts mit `ENABLE_OUT_data1=1` an den Ausgang durchgeschaltet und eine Ausgabe-Anfrage mit `DOUT_RDREQ_out` an den Ausgang gestellt. Mit `DOUT_RDACK_in=1` signalisiert die nachgeschaltete Ausleseschaltung (I/O-Karte), dass sie den Messwert übernommen hat. Nach dem Abwarten der Zeit, welche die I/O-Karte benötigt bis sie wieder bereit ist, wird die State-Machine “which\_out” in den nächsten Zustand gesetzt.

Die Ausgabe des zweiten, dritten und vierten Messwerts erfolgt analog zum oben beschriebenen Vorgang für den ersten Messwert, jedoch ohne Auslesen eines Datenwertes aus dem SDRAM.

Falls sich die beiden State-Machines “which\_out” und “count” des `s dram_read`-Befehls durch ein Fehlverhalten in keinem definierten Zustand befinden, wird in den State-Machine-Status des Fehlerzustandes `steuerung_error` gewechselt und die Status-Leitungen `STATUS_SDRAMread` und `STATUS_ERROR` werden auf 1 gesetzt.

- **steuerung\_nop:** Alle Status-Leitungen werden auf 0 gesetzt. Falls das Speicherinterface mit dem Signal `SELECTED=1` ausgewählt wird, werden die Status-Leitungen an den Ausgang geschaltet. Dies ist der Zustand, nachdem alle Daten aus dem SDRAM ausgelesen wurden.
- **steuerung\_error:** Die Status-Leitung `STATUS_ERROR` wird 1 gesetzt und keine weitere Funktion mehr durchgeführt. Das Speicherinterface kann nur durch ein Reset wieder in einen definierten Zustand gebracht werden.
- **others:** Falls sich die State-Machine durch ein Fehlverhalten in keinem definierten Zustand befindet, wird in den State-Machine-Status des Fehlerzustandes `steuerung_error` gewechselt, und die Status-Leitung `STATUS_ERROR` wird auf 1 gesetzt.

### 3.6 Einstellungen in Quartus II

Um die Ergebnisse der Analyse, Synthese und vor allem des Fittings des Designs in Quartus II zu verbessern, wurden diverse Assignments festgelegt. In Kapitel 3.1 wurde bereits beschrieben, wie die Laufzeiten der Eingangsdaten zum FIFO durch das Festlegen der Platzierung der Eingangs-Flip-Flops optimiert wurden. Im weiteren werden die wichtigsten Assignments kurz erläutert:

- Zur Synchronisation von Eingangssignalen an den internen Takt wurden jeweils zwei hintereinandergeschaltete D-Flip-Flops verwendet. Damit die dadurch entstehende Verzögerung von zwei auf einen Taktzyklus verringert wird, wurden die zweiten Flip-Flops jeweils mit dem *Inverted Clock* angesteuert.
- Um die Messdaten vom A/D-Wandler-Evaluation-Board korrekt übernehmen zu können, wurden für die Eingangssignale `DIN` und `OVR` die Setup- und Hold-Zeiten, die am Evaluation-Board gemessen wurden, spezifiziert. (siehe Kapitel 2.1)
- Für die Taktsignale `CLK` (Systemtakt 33,3333 MHz) und `DRY` (Takt der Messdaten 80 MHz) wurden *Clock Settings* definiert, damit diese Signale als globale Signale im FPGA verwaltet werden.

- Zusätzlich zu den Taktsignalen wurden die beiden Reset-Leitungen von der Steuerung an den SDRAM-Controller `RESET_SDRAM` und an die A/D-Wandler-Schnittstelle `RESET_DATA_INPUT` als *globale Signale* definiert. Damit werden kurze Laufzeiten und auch gleich lange Laufzeiten zu allen angesteuerten Gattern garantiert.
- Um kurze Laufzeiten in den Signalwegen zu haben, wurde die Lage bestimmter Design-Blöcke im FPGA festgelegt. Wie bereits erwähnt, wurden die Flip-Flops der A/D-Wandler-Schnittstelle genau festgelegt, und für die Blöcke `Data_Output` und `SDRAM_Controller` Bereiche im FPGA festgelegt, in denen sie der Fitter platzieren darf.

# Kapitel 4

## Simulation des FPGA

Um die Funktion des FPGA-Designs zu testen, bevor eine Schaltung dafür in Hardware aufgebaut wird, wurden verschiedene Simulationen durchgeführt. Die Simulationen erfolgten in der FPGA-Design-Software Quartus II und mit Hilfe des reinen VHDL-Simulators ModelSim.

### 4.1 Ergebnisse der Simulation mit Quartus II

Nach dem Kompilieren und Fitten eines FPGA-Designs in den gewünschten FPGA wird vom *Timing Analyzer* überprüft, ob die angegebenen Anforderungen an das Timing erfüllt sind und die für dieses Design maximal möglichen Taktfrequenzen berechnet.

Als passender FPGA wurde ein Altera ACEX 1K mit 30.000 Logik-Gattern in der schnellsten Ausführung, d.h. mit dem Speed Grade 1, in der Bauform PQFP gewählt. Für das Speicherinterface-Design in diesem FPGA wurden die benötigten Timing-Anforderungen erreicht. Die wichtigsten Zeiten und Frequenzen sind in Tabelle 4.1 gegenübergestellt.

	<b>benötigt</b>	<b>erreicht</b>
Setup-Time $t_{su}$ der Eingangssignale DIN und OVR	6,0 ns	1,6 ns
Hold-Time $t_h$ der Eingangssignale DIN und OVR	1,5 ns	1,3 ns
Maximale Taktfrequenz von DRY	80 MHz	166,67 MHz
Maximale Taktfrequenz von CLK	33,33 MHz	50,51 MHz

Tabelle 4.1: Ergebnisse des Quartus II Timing Analyzers

Die maximale Taktfrequenz von DRY, dem Takt mit dem die Messdaten vom FPGA übernommen werden, wird dabei von der Logik, die das FIFO verwaltet, begrenzt, der Systemtakt CLK wird durch Logikfunktionen der Steuerung eingeschränkt.

Zur Simulation eines compilierten FPGA-Designs kann der Quartus II *Simulator* verwendet werden. Um die logische Funktion des Designs zu testen, kann eine *functional simulation* durchgeführt werden. Bei der *timing simulation*, wird sowohl die logische Funktion, als auch das worst-case timing für das Design im ausgewählten FPGA simuliert.

Zur Überprüfung der Funktion des Speicherinterfaces wurde ein kurzer Messvorgang komplett simuliert. In den Abbildungen 4.1 bis 4.7 sind die Simulationsergebnisse von Quartus II zu sehen. Die einzelnen Befehle an das SDRAM sind an den drei Signalen RASn, CASn und WEn laut Tabelle 3.2 zu erkennen. Die genaue Bedeutung der einzelnen Signale ist dem Kapitel 3 zu entnehmen. Um die Funktion des FPGAs

möglichst wirklichkeitsnahe zu testen, wurden *timing simulations* mit den realen Laufzeiten der Signale im FPGA durchgeführt.

Abbildung 4.1 zeigt das Einschalten des Speicherinterfaces und den Resetvorgang, um das Speicherinterface in einen definierten Zustand zu setzen. Es ist zu erkennen, dass das Speicherinterface nach dem Einschalten der Versorgungsspannung bereits korrekt zu arbeiten anfängt und den Initialisierungs-Status mit `STATUS_INIT=1` anzeigt. Nach einem Reset mit dem Signal `RESETIN=1` wird ebenfalls wieder der Initialisierungs-Status erreicht.

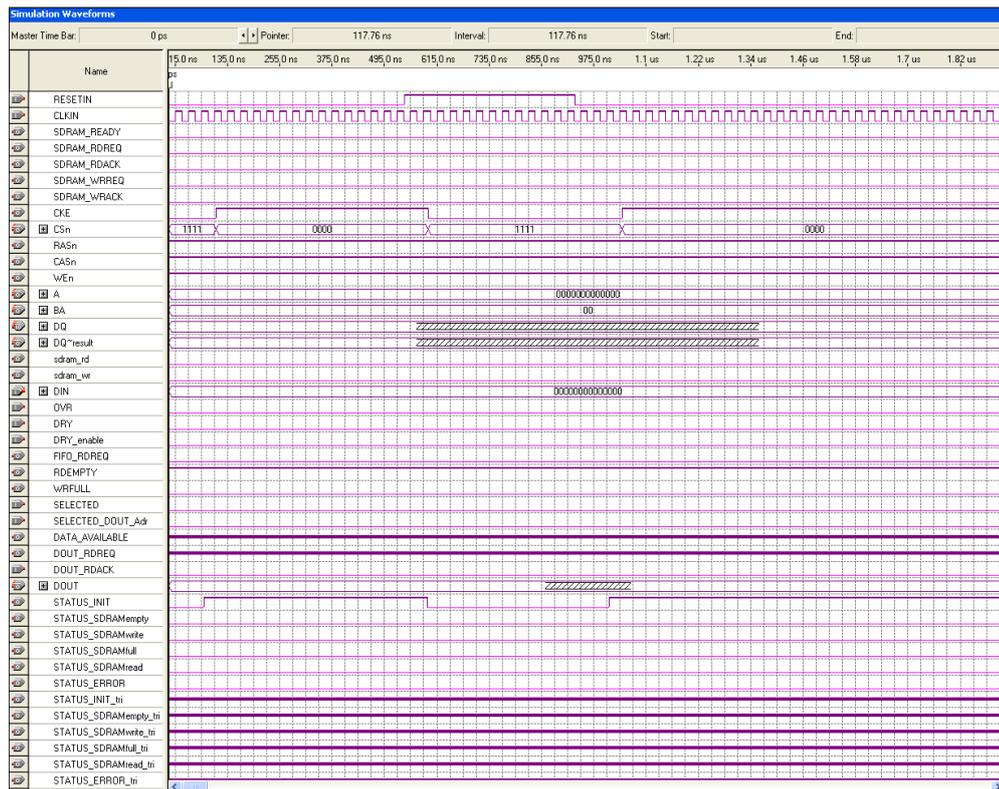


Abbildung 4.1: Simulation mit Quartus II - Reset

In Abbildung 4.2 ist die Initialisierung des SDRAMs zu sehen. Nach dem Ablauf der vom SDRAM nach dem Einschalten der Versorgungsspannung benötigten  $200\ \mu\text{s}$  wird bei  $201,045\ \mu\text{s}$  ein “Precharge all Banks”, gefolgt von neun Refresh-Zyklen, an das SDRAM ausgegeben. Mit dem “Mode Register Set” wird das SDRAM in den für das Schreiben verwendeten Burst 8-Modus gesetzt. Den Abschluss der Initialisierung erkennt man am Signal `SDRAM_READY` und an den Statusleitungen, bei denen `STATUS_SDRAMempty=1` wird.

Ein zur Erhaltung der Daten im SDRAM notwendiger Refresh-Zyklus ist in Abbildung 4.3 zu sehen. Die Dauer der Refresh-Phase ist am Signal `SDRAM_READY=0` zu erkennen.

Abbildung 4.4 zeigt den Beginn der Messwertaufnahme vom A/D-Wandler und das Schreiben der Messwerte ins SDRAM. Mit den positiven Flanken des Signals `DRY` werden die Daten ins Speicherinterface übernommen. Bei  $249,4\ \mu\text{s}$  ist zu sehen, dass mit dem Signal `DRY_enable=0` bei der Aufnahme Datenwerte ausgelassen wurden. Bei etwa  $249,9\ \mu\text{s}$  beginnt der Schreibvorgang ins SDRAM, zu erkennen durch den Wechsel der Statusleitungen auf `STATUS_SDRAMwrite=1`. In diesem Burst 8-Schreibzyklus ist zu sehen, wie zuerst die Reihe an der mit der Adresse `A` angegebenen Stelle aktiviert wird. Gleichzeitig mit der Übergabe der Spaltenadresse wird der erste der acht Datenwerte an die Datenleitungen



DQ ausgegeben. In der Abbildung ist nach wenigen Taktzyklen, die das SDRAM benötigt um das Auto Precharge aller Bänke durchzuführen, der nächste Schreibzyklus zu sehen.

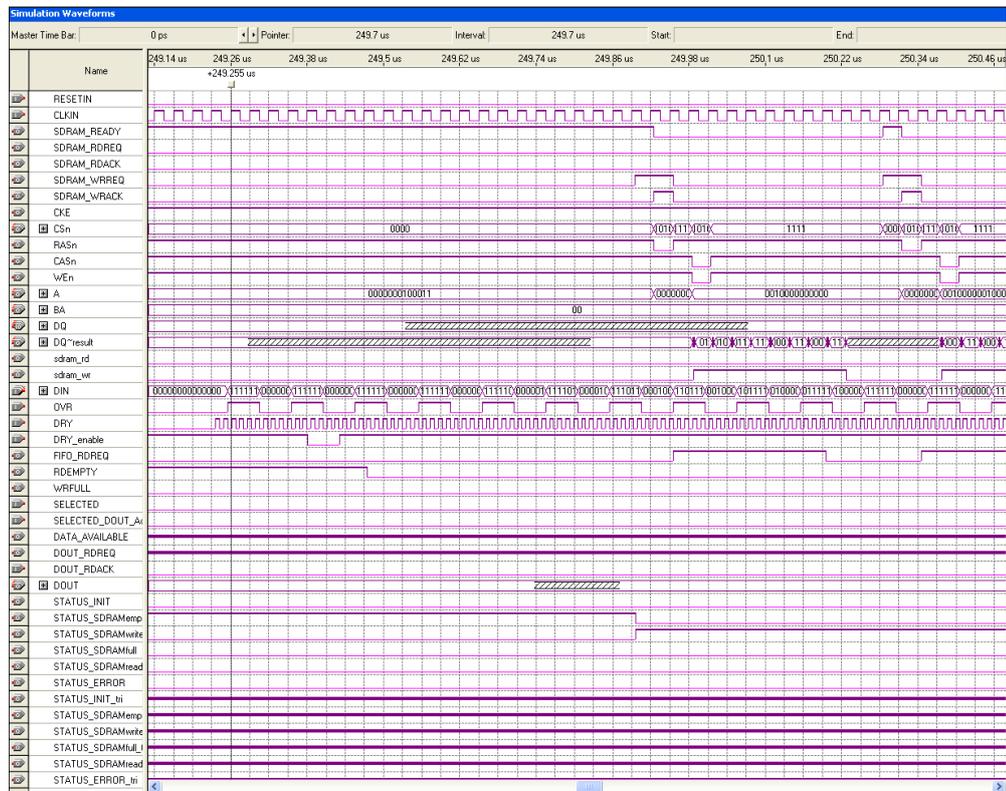


Abbildung 4.4: Simulation mit Quartus II - SDRAM-Write

Ein Refresh-Zyklus während Daten vom A/D-Wandler aufgenommen werden, ist in Abbildung 4.5 zu sehen. Bei 291,5  $\mu$ s versucht die Steuerung mit SDRAM\_WRREQ=1 einen Schreib-Zyklus zu beginnen, muss aber warten, bis der SDRAM-Controller das Refresh beendet hat und mit SDRAM\_ACK=1 den Befehl von der Steuerung annimmt.

In Abbildung 4.6 ist das Ende der Messung mit DRY=0 zu sehen. Das Speicherinterface wartet daraufhin noch 100 Taktzyklen, und wenn keine weiteren Messwerte kommen, wird vom STATUS\_SDRAMwrite in den STATUS\_SDRAMfull gewechselt. Außerdem ist in diesem Bild zu sehen, wie mit SELECTED\_DOUT\_Adr=1 die höchsten 14 Bit des 26 Bit Adresszählers des SDRAMs an den Ausgang DOUT ausgegeben werden können. Im hier simulierten Fall werden nur einige 100 Datenwerte ins SDRAM geschrieben, deshalb sind die höchsten 14 Bit des Adresszählers noch Null. Gleichzeitig zum Adresszähler werden die Statusleitungen über Tristate-Buffer an den Ausgang durchgeschaltet (im Bild die untersten sechs Signale).

Ein Auslesevorgang ist in Abbildung 4.7 zu sehen. Wird das Speicherinterface mit SELECTED=1 im STATUS\_SDRAMfull angewählt, so wird in den STATUS\_SDRAMread gewechselt und das SDRAM mit einem "Mode Register Set"-Befehl in den Burst 1-Modus gesetzt. Danach wird ein 60 Bit-Datenwort aus dem SDRAM ausgelesen. In diesem Burst 1-Lesezyklus ist zu sehen, wie zuerst die Reihe an der mit der Adresse A angegebenen Stelle im SDRAM aktiviert wird. Zwei Taktzyklen nach der Übergabe der Spaltenadresse werden Datenwerte vom SDRAM an die Datenleitungen DQ ausgegeben und vom Speicherinterface übernommen. Die Darstellung dieser Daten in diesem Timing-Diagramm ist nicht exakt, da mit keinem echten SDRAM simuliert wurde, und die Datenwerte nur zur etwa korrekten Zeit als Eingangssignale eingetragen wurden. Mit dem Hand-Shake der Signale DOUT\_RDREQ und DOUT\_RDACK werden die vier 15 Bit-Messwerte hintereinander an den Ausgang DOUT ausgegeben. Am rechten

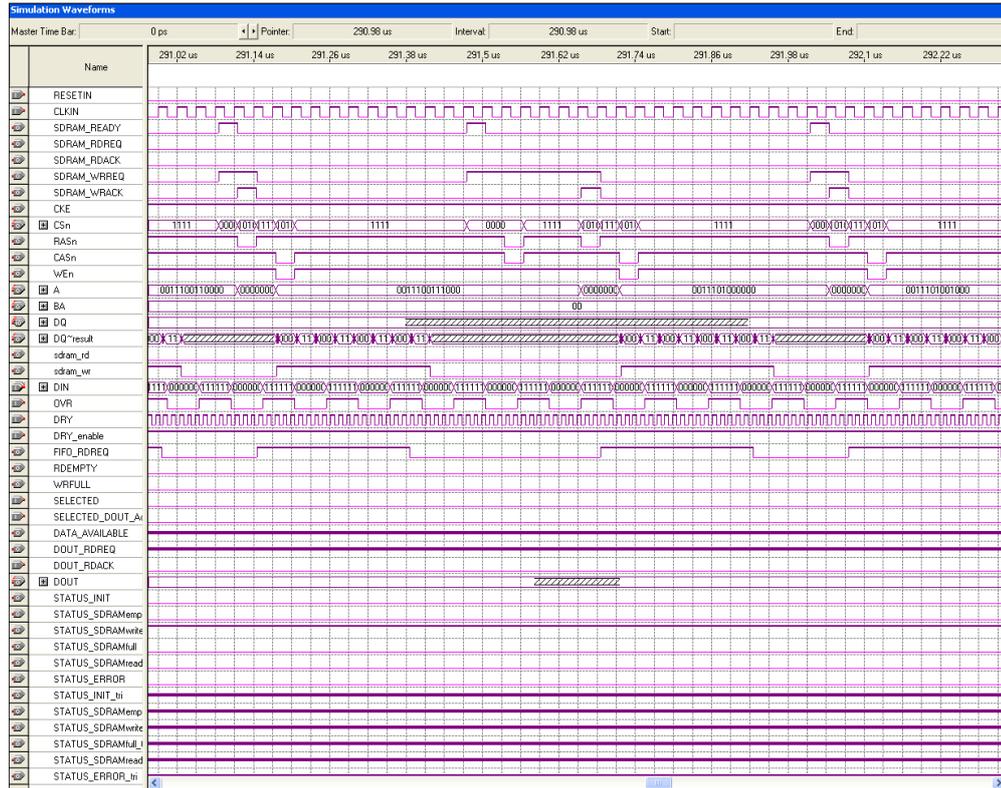


Abbildung 4.5: Simulation mit Quartus II - SDRAM-Write mit Refresh

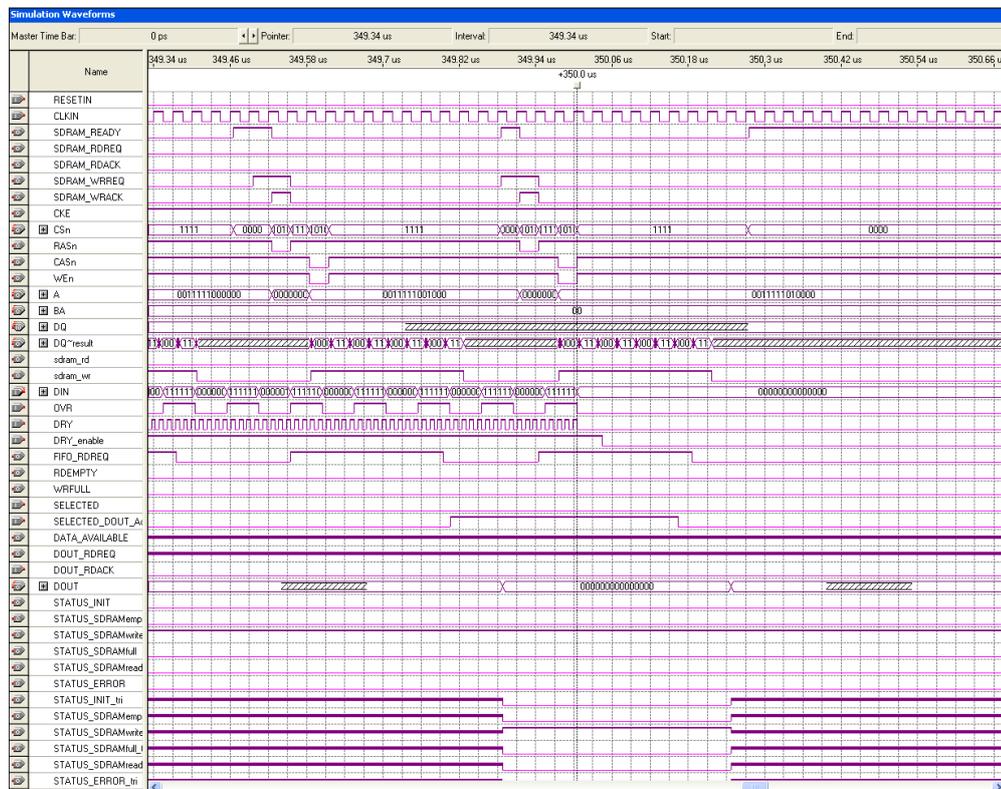


Abbildung 4.6: Simulation mit Quartus II - Ausgabe des Adresszählers

Rand des Bildes ist das Auslesen des nächsten Datenwortes aus dem SDRAM und der Beginn dessen Ausgabe zu sehen.

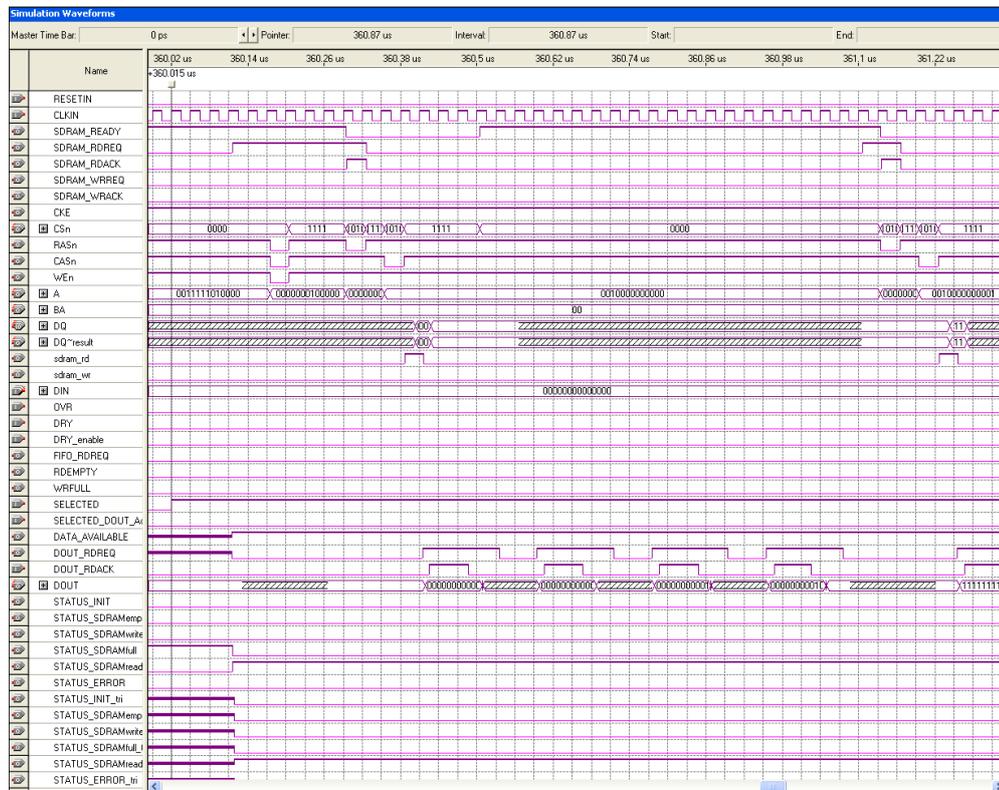


Abbildung 4.7: Simulation mit Quartus II - SDRAM-Read

## 4.2 Ergebnisse der Simulation mit ModelSim

Zur Überprüfung des korrekten Timings zwischen dem SDRAM-Controller des Speicherinterfaces und einem SDRAM-Modul wurde der VHDL-Simulator *ModelSim ALTERA 5.6a* der Firma Model Technology Incorporated verwendet. In dieser speziellen Programmversion des Simulators werden insbesondere die Altera Megafunctions, die in der Altera HDL geschrieben sind, als VHDL-Modelle unterstützt. Der Vorteil eines reinen VHDL-Simulators gegenüber der Simulation mit Quartus II besteht darin, dass die Funktion eines VHDL-Codes getestet werden kann, ohne dass er in einem FPGA realisiert werden muss. Außerdem kann in einer reinen VHDL-Simulation durch den Vergleich mit sogenannten “Test Benches” das Verhalten des VHDL-Programms überprüft werden.

Die Einbindung von ModelSim in den Design-Prozess mit Quartus II ist in der Altera Application Note [26] erklärt.

Im Blockschaltbild Abbildung 4.8 ist der Aufbau der simulierten Schaltung skizziert.

Da das Design des Speicherinterfaces in Quartus II kein reiner VHDL-Code, sondern eine Kombination aus VHDL-Files, Block Design Files und Altera Megafunctions ist, musste das ganze Projekt in VHDL-Code übersetzt werden. Block Design Files können von Quartus II als VHDL-Files exportiert werden, für die Altera Megafunctions werden von ModelSim VHDL-Modelle bereitgestellt.

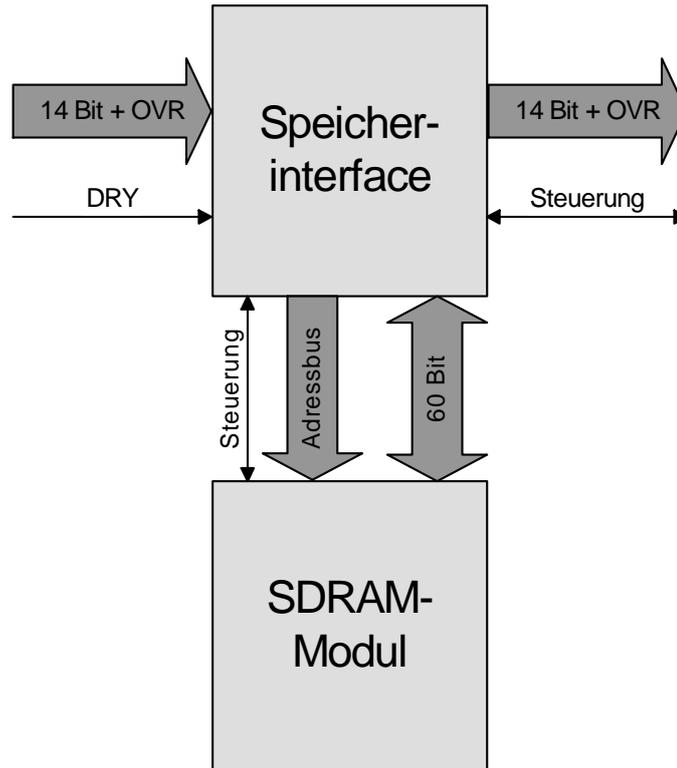


Abbildung 4.8: Blockschaltbild der Simulation mit ModelSim

Für SDRAM-Module gibt es leider keine VHDL-Modelle, sondern nur für SDRAM-Chips. Deshalb wurde das verwendete 512 MB SDRAM-Modul, wie in Abbildung 4.9 dargestellt, mit den VHDL-Modellen der verwendeten SDRAM-Chips als VHDL-Code modelliert. Als Modell dieser 32 MBitx8 SDRAM-Chips wurde das File MT48LC32M8A2.VHD der Firma Micron Semiconductor verwendet. In diesem Modell werden alle Timings und alle Befehle an das SDRAM auf ihre Plausibilität für den aktuellen SDRAM-Zustand überprüft, und, falls ein Fehler auftritt, als Fehlermeldung im ModelSim-Hauptfenster ausgegeben.

Als Test der Kommunikation zwischen SDRAM-Controller und SDRAM wurde wie bei der Simulation mit Quartus II in Kapitel 4.1 ein kurzer Messvorgang komplett simuliert. Die Simulation mit ModelSim lieferte die selben Ergebnisse wie Quartus II. Die Übergabe von Datenwerten zwischen Speicherinterface und SDRAM erfolgt korrekt und es gibt keine Fehlermeldungen wegen Verletzungen des Timing-Verhaltens des SDRAMs. In Abbildung 4.10 ist, als Beispiel der Simulationsergebnisse von ModelSim, die Initialisierung des SDRAMs zu sehen. Sie ist äquivalent der Simulation mit Quartus II in Abbildung 4.2.

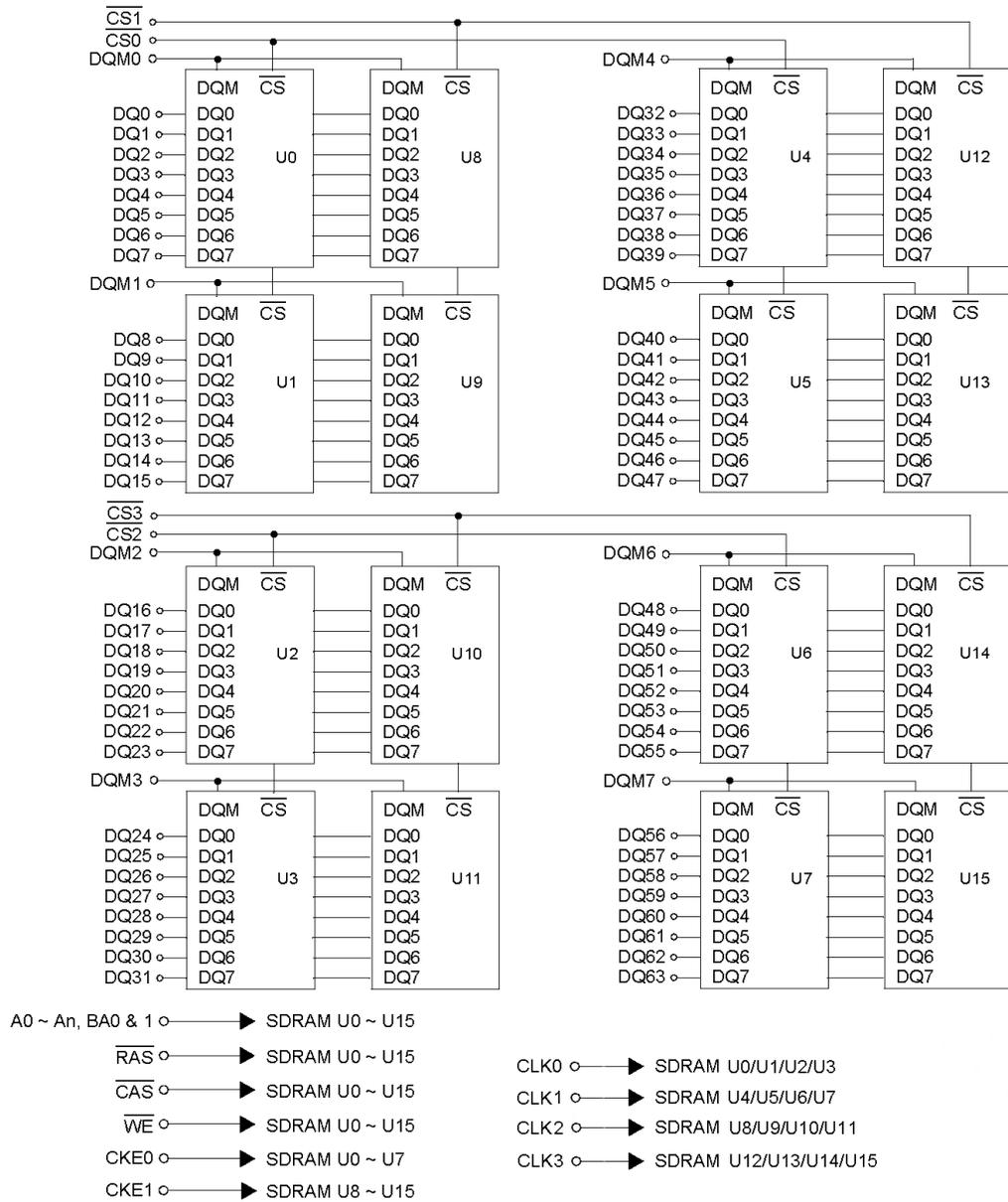


Abbildung 4.9: Blockschaltbild des 512 MB SDRAM-Moduls

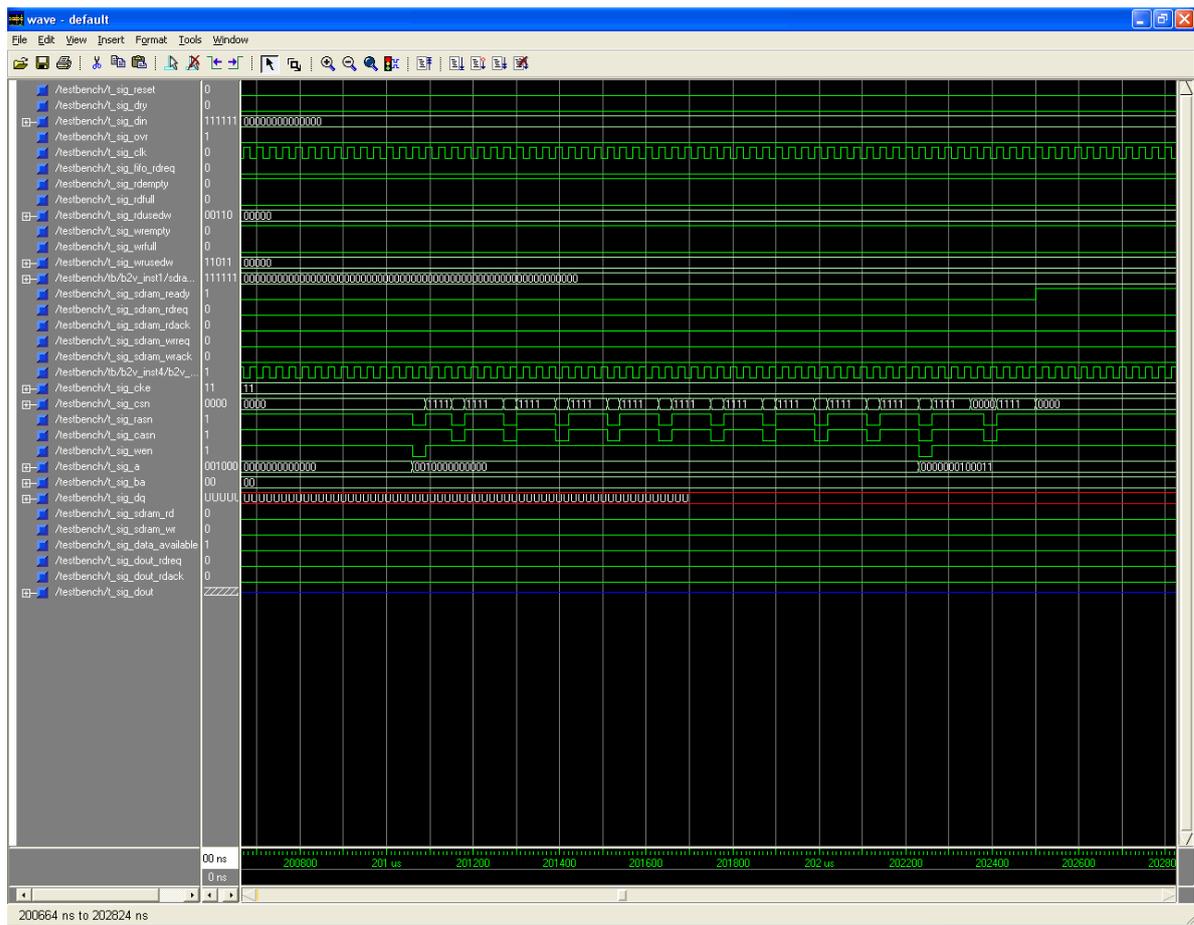


Abbildung 4.10: Simulation mit ModelSim

# Kapitel 5

## Entwurf der Hardware

Nachdem die Funktion des FPGAs fertig entworfen und simuliert war, und dadurch der benötigte FPGA und dessen Pinbelegung feststand, musste eine Schaltung entworfen werden, welche die Programmierung des FPGAs ermöglicht, eine Eingangs-Schnittstelle zum A/D-Wandler Evaluation Board und eine Ausgangs-Schnittstelle zum Auslesen der Daten besitzt. Für diese Schaltung musste zur Fertigung des Speicherinterfaces ein Layout entwickelt werden.

Als Software zur Erstellung des Schaltplans und des Layouts wurde Eagle 4.11 der Firma CadSoft verwendet. Die Funktionen und die Bedienung des Programmes sind in [27] und [28] nachzulesen.

### 5.1 Entwurf der Schaltung

In Abbildung 5.1 ist ein Blockschaltbild der zur Funktion des Speicherinterfaces notwendigen Schaltung zu sehen.

Auf der rechten Seite des Blockschaltbildes erkennt man den Stecker des *Eingangs* des Speicherinterfaces, an den das A/D-Wandler Evaluation Board angeschlossen wird. Wegen der an dieser Stelle auftretenden hohen Taktfrequenzen von 80 MHz vom A/D-Wandler, ist hier zu Verhinderung von Reflexionen besonders auf gute Anpassung an die Ausgangsimpedanz des A/D-Wandler Evaluation Boards zu achten.

Über den *Ausgangsstecker* soll das Speicherinterface zusätzlich mit den benötigten Spannungen von 2,5 V für den FPGA-Core und 3,3 V für die FPGA-IO-Treiber versorgt werden. Um keine weitere Versorgungsspannung zu benötigen und in der ganzen Schaltung die vom FPGA und SDRAM benötigten LVTTTL-Pegel verwenden zu können, wurden bei den weiteren Schaltungskomponenten Low Voltage Bauteile mit einer Versorgungsspannung von 3,3 V ausgewählt.

Zur Anzeige des Status des Speicherinterfaces sind *Leuchtdioden* vorgesehen, auf der rechten Seite des Blockschaltbildes unten zu sehen. Da die Ausgänge des FPGAs nicht genug Leistung zur Ansteuerung einer Leuchtdiode liefern können, wurde ein Treiber des Typs 74LV365 verwendet. Dieser sechsfache *buffer/line driver* arbeitet mit einer Versorgungsspannung von 3,3 V und liefert Ausgangsströme bis zu 35 mA. Der Initialisierungszustand des FPGAs wird mit einer gelben, der Fehlerzustand mit einer roten und die anderen Zustände mit einer grünen Leuchtdiode angezeigt.

Bei dem verwendeten FPGA des Types ACEX 1K handelt es sich um einen auf CMOS SRAM-Elementen basierenden Typ, d.h. eine Programmierung des FPGAs geht mit dem Ausschalten der Versorgungsspannung verloren. Zur Programmierung des FPGAs sind die *JTAG*- und die *Byte Blaster-Schnittstelle*, im Blockschaltbild links oben zu sehen, vorgesehen. Bei der JTAG-Schnittstelle handelt es sich um eine von IEEE definierte Schnittstelle, die Byte Blaster-Schnittstelle wurde von der Firma Altera festgelegt. Für die Byte Blaster-Schnittstelle wurde der Passive Serial-Mode zur Programmierung des FPGAs verwendet. Dabei handelt es sich um den einfachsten Programmier-Modus, da damit nur ein einzelnes Device angesprochen werden kann und die Daten seriell an das FPGA übertragen werden.

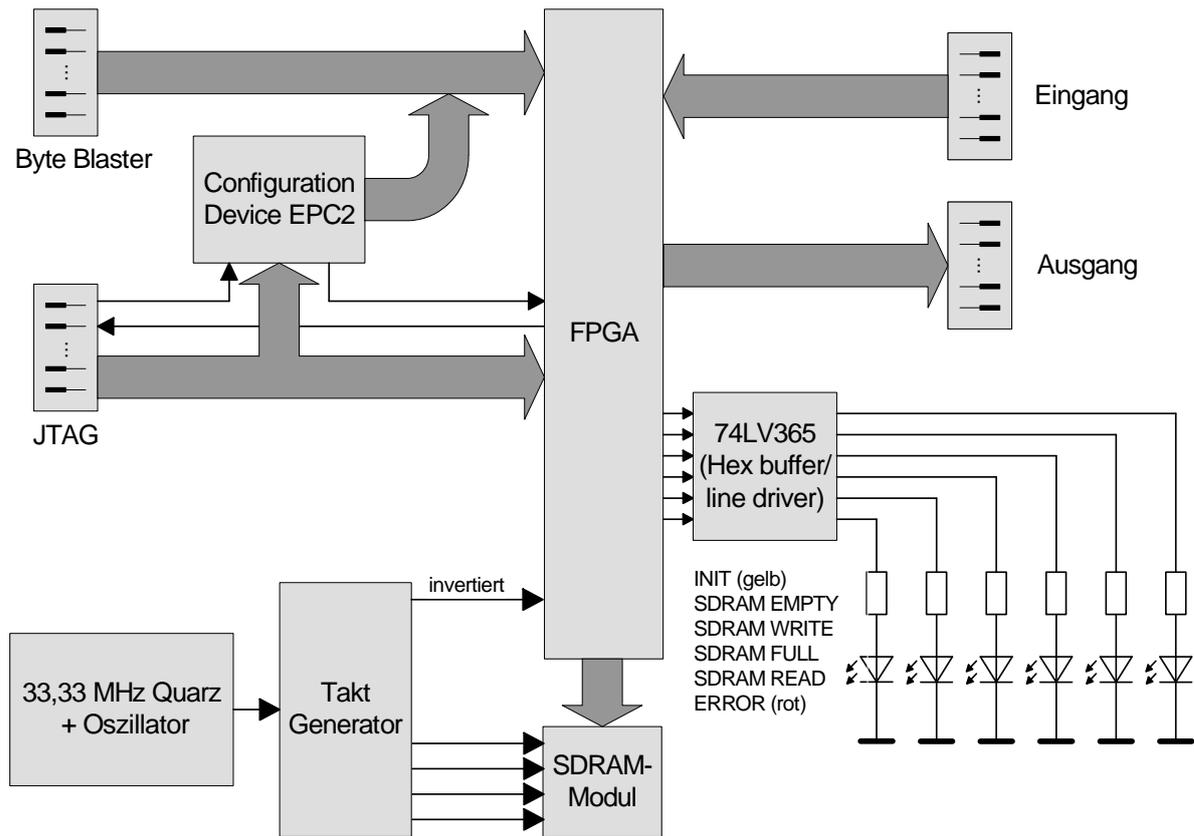


Abbildung 5.1: Blockschaltbild des Schaltplans

Um das Speicherinterface nicht jedesmal vom Computer über eine der beiden Schnittstellen programmieren zu müssen, wurde ein Configuration Device des Typs EPC2 verwendet. Dieses Configuration Device besteht aus der zur Programmierung des FPGAs notwendigen Logik und einem EEPROM, das die Programmierung des FPGAs enthält und bis zu 100 mal programmiert und gelöscht werden kann. Nach dem Einschalten der Spannungsversorgung programmiert das Configuration Device den FPGA selbständig mit der im EEPROM gespeicherten Programmierung. Die Programmierung des Configuration Devices erfolgt über die JTAG-Schnittstelle, in der das Configuration Device und der FPGA in einer sogenannten Device Chain hintereinander geschaltet sind. In einer Device Chain werden Daten, die nicht für dieses Device bestimmt sind, an das nächste Device weitergegeben. Die verschiedenen Arten der Programmierung von FPGAs und die Configuration Devices sind in den Datenblättern und Application Notes [29] bis [33] der Firma Altera genau beschrieben.

Der 33,3333 MHz Takt des Systems wird von einem *Oszillator* geliefert und von einem *Takt-Generator* für den FPGA und das SDRAM aufbereitet. Es wurden wieder Low Voltage Bauteile verwendet, um FPGA und SDRAM direkt mit LVTTTL-Pegeln ansteuern zu können. Der Takt-Generator erzeugt vier zum Signal des Oszillators synchrone Taktsignale mit einer sehr geringen Phasenverschiebung, welche die vier Clock-Leitungen des SDRAMs ansteuern, und ein zum Signal des Oszillators invertiertes Taktsignal, das den FPGA ansteuert.

Der komplette Schaltplan aus Eagle ist im Anhang in den Abbildungen C.1 bis C.3 zu sehen. Teil 1 ist in eine linke (Abbildung C.1) und rechte (Abbildung C.2) Hälfte geteilt und zeigt die zur Funktion notwendige Schaltung. Teil 2 (Abbildung C.3) zeigt die Stiftleisten CON6 - CON9, die nur zur Überprüfung der Pegel an den FPGA-Pins dienen und im Normalbetrieb der Schaltung nicht montiert werden müssen.

In Abbildung C.1 ist links oben die Schaltung zum Programmieren des FPGAs zu sehen. Die dafür verwendete Schaltung ist eine Kombination verschiedener Programmierschaltungen aus [30]. Um die Eingänge des FPGAs auf definierte Pegel zu legen, falls der JTAG- CON3 oder Byte Blaster-Stecker CON4

nicht beschaltet sind, sind die Pull-Up bzw. Pull-Down Widerstände R1 bis R8 vorhanden. Die Device Chain des Configuration Devices EPC2 IC5 und des FPGAs IC1 an der JTAG-Schnittstelle ist am Datensignal ersichtlich. Das Eingangsdatensignal TDI vom JTAG-Stecker wird an TDI des Configuration Devices geschaltet. Falls das Configuration Device nicht angesprochen wurde, wird das Datensignal über TDO an den TDI-Eingang des FPGAs weitergeleitet. Der TDO-Ausgang des FPGAs führt zum TDO des JTAG-Steckers zurück.

Die Signale des Eingangs-Steckers CON1 werden über Serienwiderstände direkt an die Pins des FPGAs geführt. Als Widerstände R19 und R20 wurden zur Anpassung an das A/D-Wandler Evaluation Board Widerstandsarrays mit  $100\ \Omega$ , dem selben Wert wie am Evaluation Board, verwendet.

Am Ausgangs-Stecker CON2 sind neben den Ausgangssignalen, die direkt mit dem FPGA verbunden sind, noch die Spannungsversorgungen zu sehen. Zur Entkopplung und Glättung der Versorgungsspannungen wurden direkt am Eingang die Ferrite L1 bis L3 in Serie geschaltet und direkt bei den Versorgungspins der ICs Entkopplungskondensatoren platziert. Um eine Rückwirkung der Versorgung des Quarzes und Oszillators auf die restliche Versorgung zu verhindern, wurde in die Spannungsversorgung der Taktgenerierungsschaltung nochmals ein Ferrit L4 platziert. Als Entkopplungskondensatoren wurden beim Treiber für die Leuchtdioden wegen der höheren Schaltströme eine Parallelschaltung von  $10\ \text{nF}$  und  $1\ \mu\text{F}$  und bei den restlichen ICs eine Parallelschaltung von  $10\ \text{nF}$  und  $150\ \text{nF}$  gewählt.

Die Ansteuerung der Leuchtdioden LED1 bis LED6 erfolgt über den "Hex buffer/line driver" 74LV365 IC2, der mit einer Versorgungsspannung von  $3,3\ \text{V}$  arbeitet. Die Treiber sind dadurch, dass die beiden Enable-Leitungen auf GND liegen, immer durchgeschaltet. Die Ausgänge können beim HIGH-Pegel der Ausgangsspannung  $35\ \text{mA}$  liefern. Für die verwendeten Leuchtdioden wurden aus dem Datenblatt der typische Arbeitspunkt mit einem Spannungsabfall von  $2,2\ \text{V}$  bei  $20\ \text{mA}$  ermittelt. Dadurch ergibt sich der Wert der Vorwiderstände R9 bis R14 mit  $\frac{3,3\ \text{V} - 2,2\ \text{V}}{20\ \text{mA}} = 55\ \Omega$ . Als nächsthöherer Wert der Widerstandsreihe E12 wurde  $56\ \Omega$  gewählt.

In Abbildung C.2 ist der Stecker des SDRAMs CON5 und dessen Verbindungen zum FPGA zu sehen. Das Symbol des Steckers ist wegen der hohen Anzahl der Pins in zwei Teilsymbole CON5-A und CON5-B aufgeteilt.

Im unteren Teil der Abbildung C.2 ist die Schaltung zur Generierung des Taktes zu sehen. Das  $33,3333\ \text{MHz}$  Taktsignal wird von dem, im Bereich von  $1$  bis  $125\ \text{MHz}$  programmierbaren, Hochfrequenz Quarzoszillator SG-8002 IC3 generiert. Die Programmierung des Quarzes erfolgte vom Distributor des ICs auf die gewünschte Frequenz. Der Takt wird vom Oszillator in  $3,3\ \text{V}$ -Pegeln mit einer Frequenzstabilität von  $\frac{\Delta f}{f} = \pm 50 \times 10^{-6}$  geliefert. Um die vier synchronen Taktsignale für die vier Taktleitungen des SDRAM-Moduls und den invertierten Takt für das FPGA zu liefern, wird ein "Low Voltage Low Skew CMOS PLL Clock Driver" MC88LV915T IC4 der Firma Motorola verwendet. Die Phasendifferenz der vier SDRAM-Takte ist kleiner als  $500\ \text{ps}$  und weniger als  $750\ \text{ps}$  zum Takt des FPGA. Mit der aus dem Datenblatt [34] entnommenen Beschaltung wird durch die Rückkopplung des Signals Q4 (Pin 28) auf den Feedbackpin (Pin 5) die PLL auf den Frequenzmultiplikationsfaktor 1 gestellt. Als externe Beschaltung des Clock-Treibers ist nur der Loop-Filter C1 bis C3 und R16 bis R18 an den Pins 8, 9 und 10 notwendig. Die üblichen Bauteilwerte für das Loop-Filter wurden ebenfalls dem Datenblatt [34] entnommen.

## 5.2 Entwurf des Layouts

Beim Entwurf des Layouts der in Kapitel 5.1 beschriebenen Schaltung wurden die Richtlinien [35] bis [37] der Firma Altera zum Bau eines FPGA-Boards besonders berücksichtigt. Insbesondere wird darin beschrieben, dass in Taktleitungen keine Durchkontaktierungen sein sollen, da das induktive Verhalten einer Durchkontaktierung die Flankensteilheit eines Taktsignales verringert. Außerdem ist bei den Versorgungsspannungen des FPGAs auf gute Entkopplung zu achten, d.h. die Kondensatoren zur Entkopplung sollen möglichst nahe bei den Versorgungspins liegen.

Das Layout konnte auf einer, wegen der niedrigen Kosten gewünschten, gewöhnlichen zweiseitigen Leiterplatte der Stärke  $1,6\ \text{mm}$  mit Kupferauflagen von  $35\ \mu\text{m}$  realisiert werden, da ein Grossteil der Entflechtungsarbeit durch Quartus II im FPGA durchgeführt wurde. Durch die richtige Wahl der Pinvergabe am FPGA konnten nämlich die Verbindungen zum Eingangs-, Ausgangs- und SDRAM-Stecker kreuzungsfrei geführt werden. Dadurch war es weitgehend möglich Signalleitungen auf der Oberseite (Bauteilseite)

und Versorgungsspannungen auf der Unterseite (Lötseite) zu führen. Dieses Vorhaben wurde nur durch das als wichtiger eingestufte Kriterium der Durchkontaktierungsfreiheit der Taktleitungen des SDRAMs im Bereich der Pins 111 bis 140 des FPGAs unterbunden. Außer für Versorgungsleitungen, die Leitungen zu den LEDs und die Leitungen vom Eingangs-Stecker zum FPGA wurde eine Leiterbahnstärke von 6 mil und ein Leiterbahnabstand von mindestens 6 mil verwendet. Auf der Unterseite der Platine wurden alle nicht durch Versorgungsspannungen oder andere Signale verwendeten Bereiche als grosse Massefläche ausgeführt.

Im Anhang ist in den Abbildungen E.1 und E.2 das fertige Layout der Schaltung zu sehen. Die Abbildungen F.1 und F.2 zeigen die Bestückung der Bauteile.

In dem Bestückungsplan der Bauteilseite ist rechts oben der Eingangs-Stecker CON1 für das A/D-Wandler Evaluation Board, und links oben der Ausgangs-Stecker CON2 zu sehen. In den Layouts ist an dessen äußersten Pins die Versorgungsspannungszuführung zu erkennen. In der Mitte der Platine befindet sich der FPGA IC1 umgeben von den Stiftleisten CON6 bis CON9, die zum Abgreifen der Signale an den FPGA-Pins zu Testzwecken dienen. In dem Bestückungsplan der Bauteilseite befindet sich unten der Stecker des SDRAMs CON5. Die Lage der Taktgenerierungsschaltung IC3 und IC4 wurde absichtlich zwischen FPGA und SDRAM gewählt, um möglichst gleiche Leitungslängen des Taktsignales zum FPGA und der vier Taktsignale zum SDRAM zu erreichen, damit keine Laufzeitdifferenzen zwischen den Signalen auftreten. Die Beschaltung des Clock-Treibers IC4 wurde möglichst nahe an dessen Pins 8 bis 10 positioniert, um parasitäre Effekte möglichst gering zu halten, da sie zu Störungen der Filterfunktion führen können. Auf der rechten Seite des Bestückungsplan der Bauteilseite befindet sich der Rest der Schaltung mit dem Byte Blaster-Stecker CON4, dem JTAG-Stecker CON3, dem Configuration Device IC5 und dem Treiber IC2 für die Leuchtdioden LED1 bis LED6. Auf der Unterseite der Platine befinden sich nur die Ferrite L1 bis L4 und Kondensatoren zur Glättung und Entkopplung der Versorgungsspannungen.

# Kapitel 6

## Aufbau des Speicherinterface

### 6.1 Aufbau der Speicherinterface-Platine

Nachdem der Schaltplan und das Layout des Speicherinterfaces fertig entworfen waren, wurde die Platine wegen der feinen Leiterstruktur nicht selbst hergestellt, sondern von der Firma Q-print electronic GmbH gefertigt. Da das verwendete PQFP-Package des FPGAs einen Pinraster von nur 0,5 mm hat, und daher händisch nicht mehr leicht einzulöten ist, wurde auch die Bestückung des FPGA der Leiterplatten-Firma in Auftrag gegeben. Die gelieferte Leiterplatte, nur mit dem FPGA bestückt, ist in Abbildung 6.1 zu sehen.

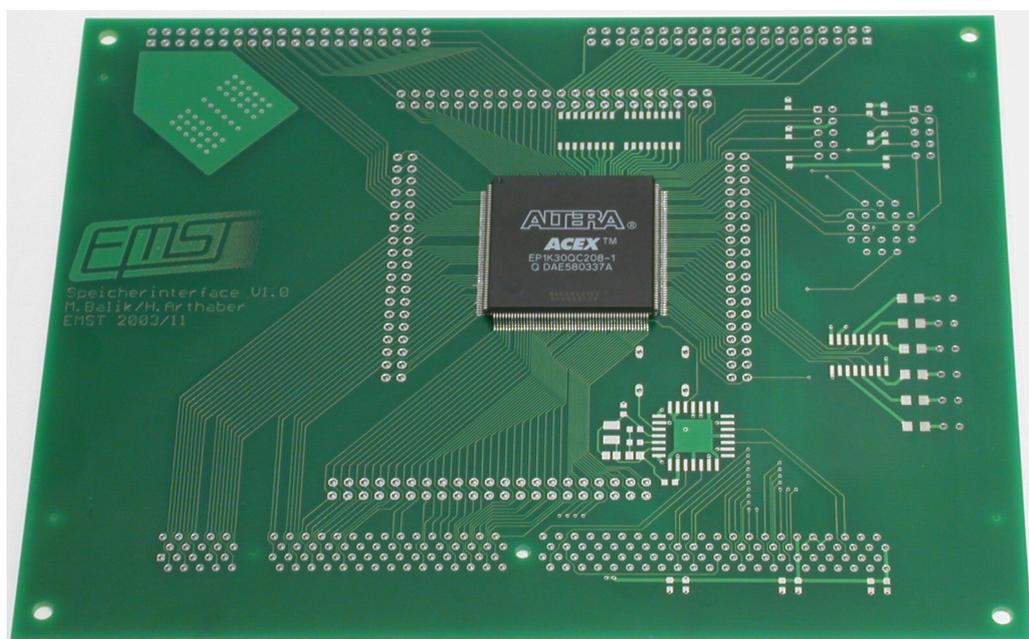


Abbildung 6.1: Speicherinterface-Platine mit FPGA bestückt

Die restlichen Bauelemente der Bauteillisten Tabelle D.1 und D.2 wurden laut der Bestückungspläne Abbildung F.1 und F.2 händisch eingelötet. Die fertig bestückte Leiterplatte ist in den Abbildungen 6.2 und 6.3 zu sehen.

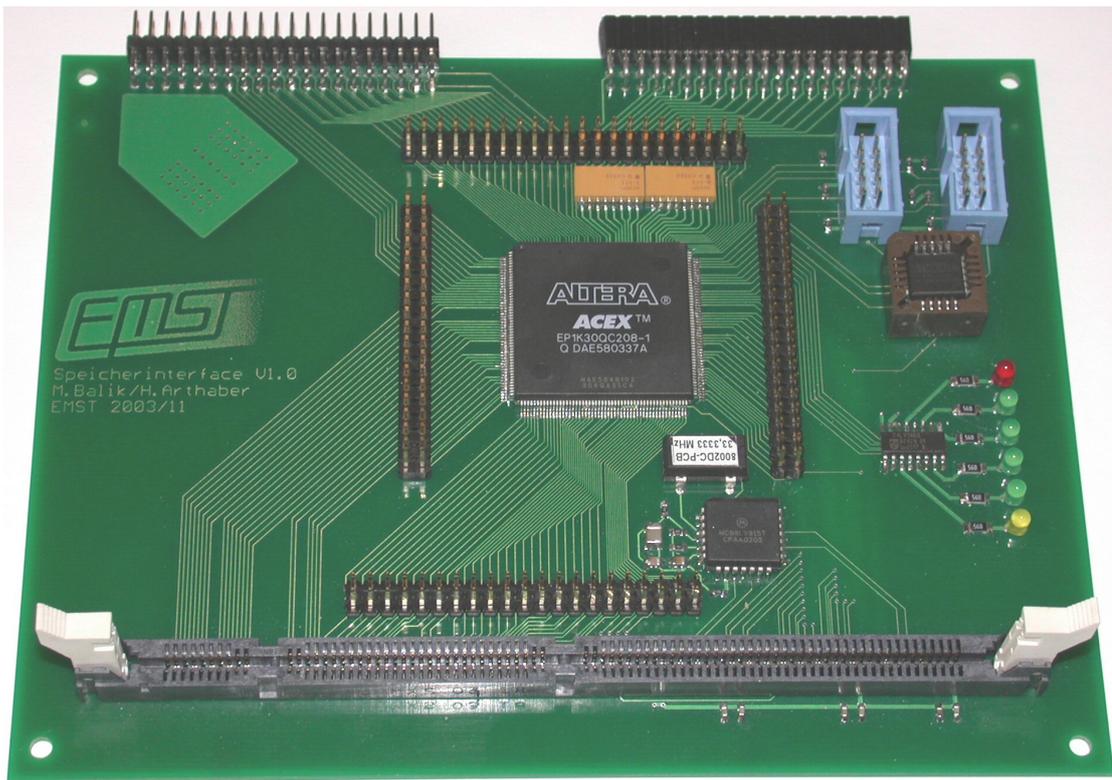


Abbildung 6.2: Speicherinterface-Platine Bauteilseite

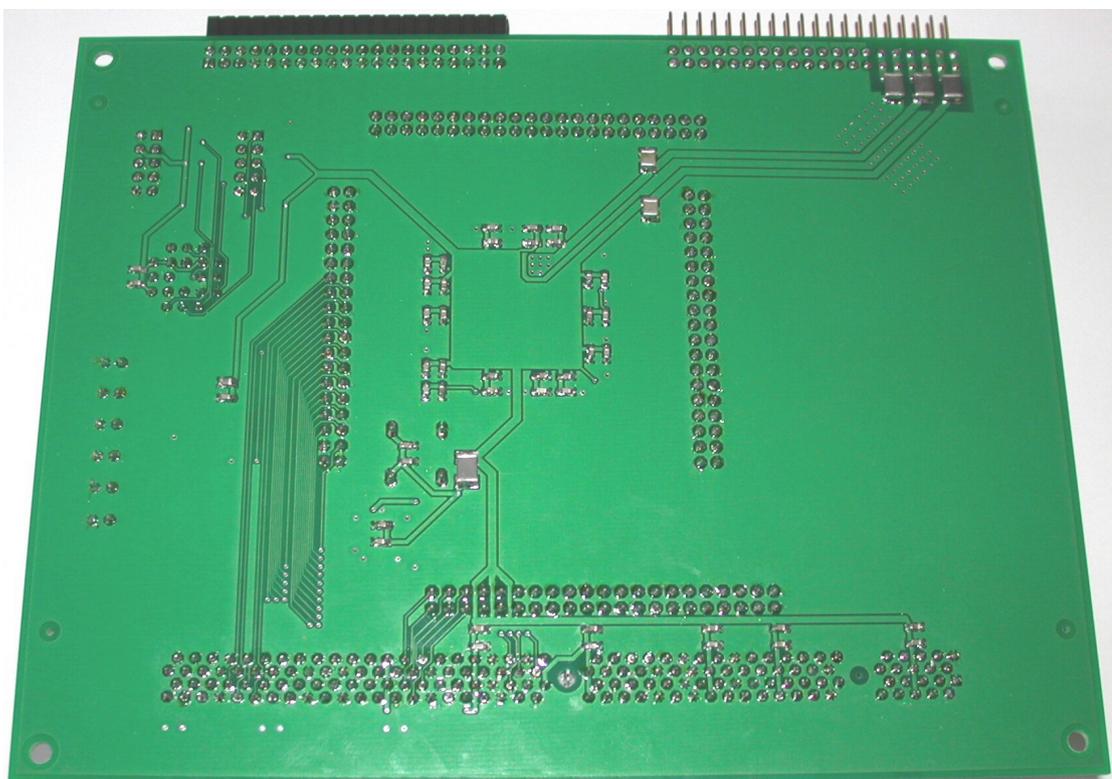


Abbildung 6.3: Speicherinterface-Platine Lötseite

## 6.2 Aufbau eines Messsystems

Mit der fertigen Leiterplatte des Speicherinterfaces wurde ein einfaches Messsystem zur Überprüfung der Funktion aufgebaut. Ein Blockschaltbild dieses Messsystem ist in Abbildung 6.4 zu sehen.

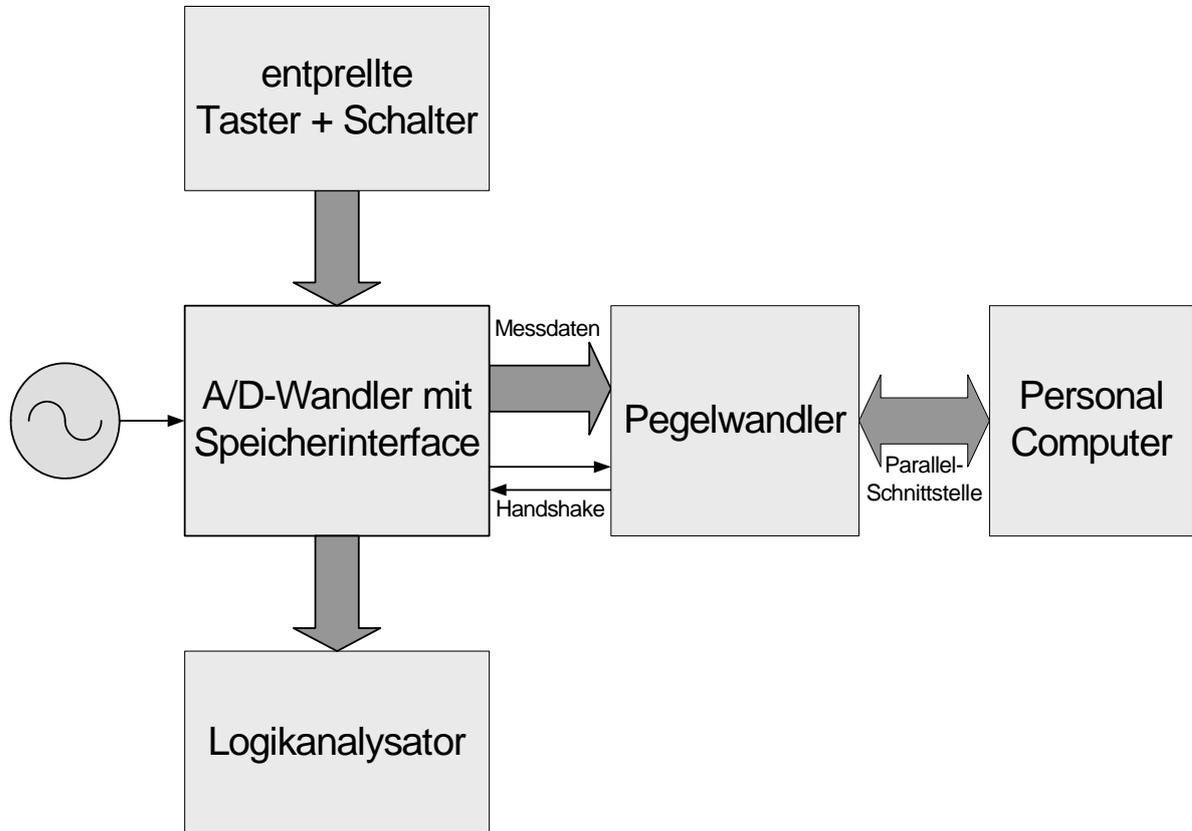


Abbildung 6.4: Blockschaltbild des Messsystems zur Überprüfung der Funktion

Ein Signalgenerator liefert das zu messende Eingangssignal für den “A/D-Wandler mit Speicherinterface”. Da die zum Auslesen des Speicherinterfaces notwendige I/O-Karte noch nicht vorhanden war, wurde das Auslesen der Messdaten aus dem Speicherinterface über die Parallel-Schnittstelle eines Personal Computers realisiert. Dazu war eine Pegelwandlung der 3,3 V-Pegel des Speicherinterfaces auf die 5 V-Pegel des Parallelports notwendig. Um Daten über die Parallel-Schnittstelle einlesen zu können, wurde der Parallelport im bidirektionalen EPP-Modus betrieben. Mit Hilfe zweier Steuerleitungen der Parallel-Schnittstelle wurde das Handshake zum Auslesen der Daten gesteuert, und über die Datenleitungen und vier Steuerleitungen konnten die höchstwertigen 12 Bits der Messwerte aus dem Speicherinterface ausgelesen werden. Die restlichen Steuerleitungen des Speicherinterinterfaces, wie zum Beispiel RESET oder SELECT, wurden mit Tastern und Schaltern realisiert. Es war wichtig diese Taster und Schalter zu entprellen, da sich im Betrieb des Speicherinterfaces zeigte, dass das Prellen die State-machine des Speicherinterfaces in undefinierte Zustände versetzte und so einen Fehlerfall hervorrief. An den Steckerleisten CON6 bis CON9 wurden Signale abgegriffen und mit einem Logikanalysator überprüft. In Abbildung 6.5 ist ein Foto des aufgebauten Messsystems zu sehen. Das Speicherinterface und der A/D-Wandler befinden sich im Bild links unten und sind in Abbildung 6.6 vergrößert dargestellt. Auf der linken Seite des Bildes ist der Signalgenerator zu sehen, in der Mitte befinden sich vier Netzgeräte zur Spannungsversorgung, und auf der rechten Seite erkennt man den Logikanalysator mit dem Ergebnis der Messung eines Schreibvorgangs ins SDRAM.

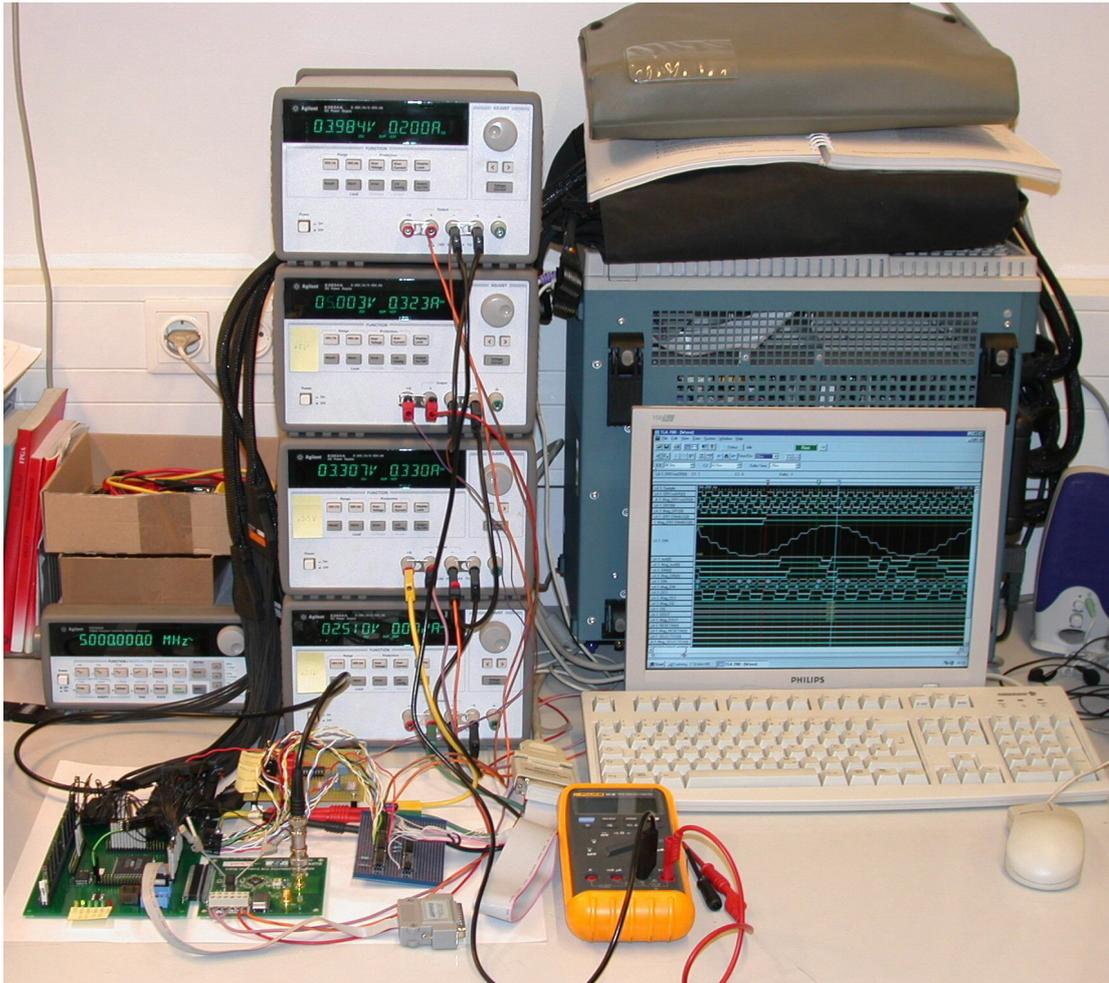


Abbildung 6.5: Messsystems zur Überprüfung der Funktion

Abbildung 6.6 zeigt ein Detail des Messsystems. Darauf ist das Speicherinterface mit montiertem SDRAM-Modul zu sehen, an das das A/D-Wandler Evaluation Board (rechts oben im Bild) angeschlossen ist. Bei den beiden Schaltungen links oben am Bildrand handelt es sich um die entprellten Taster und den Pegelwandler für die Parallel-Schnittstelle. Die schwarzen Kabel, die aus dem linken Bildrand führen, sind die Anschlüsse des Logikanalysators.

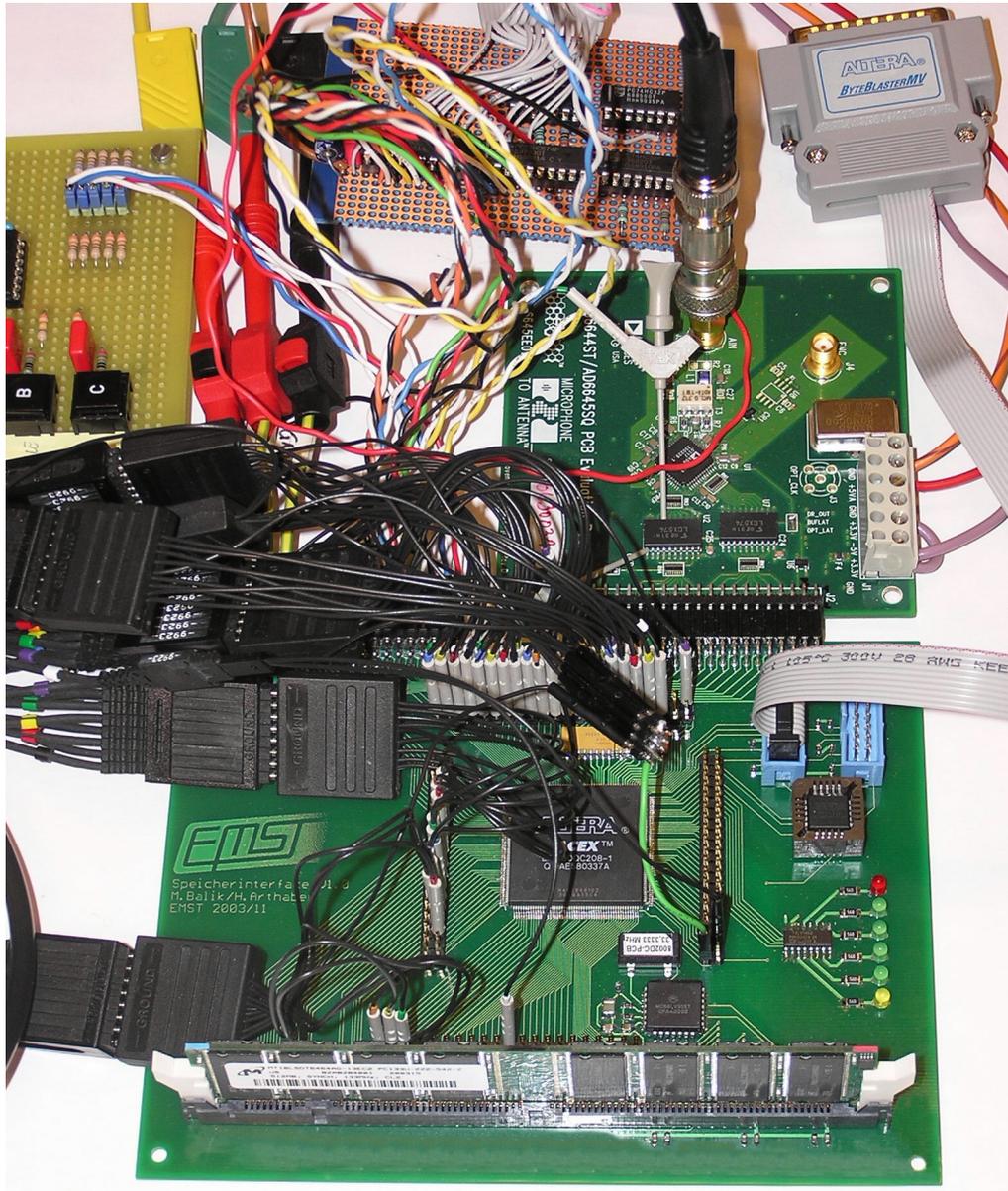


Abbildung 6.6: Detail des Messsystems zur Überprüfung der Funktion

Mithilfe dieses Messsystems wurde die korrekte Funktion des Speicherinterfaces überprüft. Bei den durchgeführten Tests konnte keine Fehlfunktion des Speicherinterfaces festgestellt werden.

# Kapitel 7

## Zusammenfassung

In dieser Diplomarbeit wurde die Entwicklung und der Aufbau eines Speicherinterfaces beschrieben, das die Messdaten eines schnellen A/D-Wandlers in einen Speicher schreibt. Das Auslesen der Daten aus diesem Speicher ist über eine einfache Schnittstelle im Handshake-Verfahren vorgesehen. Gefordert waren eine hohe Speichertiefe, d.h. eine lange Messzeit, und niedrige Kosten des Speichers. Die Messdaten des A/D-Wandlers sollten deshalb in ein handelsübliches 512 MByte SDRAM-Modul, wie es auch in Personal Computern eingesetzt wird, geschrieben werden. Weil die Daten vom ausgewählten 14 Bit-A/D-Wandler mit der hohen Datenrate von 80 MHz kontinuierlich ausgegeben werden, das Schreiben ins SDRAM aber wegen der Ansteuerung des SDRAMs und zur Erhaltung von Datenwerten im SDRAM notwendiger Refresh-Vorgänge, nicht kontinuierlich erfolgen kann, mussten die Messwerte vom A/D-Wandler in einem FIFO zwischengespeichert werden. Außerdem sollen mehrere Speicherinterfaces parallel geschaltet betrieben werden können, um eine Mehrkanalmessung zu ermöglichen.

Als *A/D-Wandler* wurde wegen seiner hohen Geschwindigkeit von 80 MSamples/s und der hohen Auflösung von 14 Bit der AD6645 der Firma Analog Devices ausgewählt. Um das bezüglich Schirmung kritische Layout des A/D-Wandlers auf der Leiterplatte des Speicherinterfaces zu vermeiden, wurde das von Analog Devices angebotene Evaluation Board für den AD6645 verwendet. Durch unterschiedliche Treiber am Evaluation Board, die verschiedene Verzögerungszeiten haben, ergibt sich ein gegenüber dem A/D-Wandler geändertes Timingverhalten am Ausgang. Dieses Timingverhalten wurde gemessen und die Eingangsstufe des Speicherinterfaces an diese Setup- und Hold-Zeiten angepasst. Es ist dazu zu bemerken, dass dieses Timingverhalten bei der maximalen Sample-Frequenz des A/D-Wandlers von 80 MHz gemessen wurde und für andere Sample-Frequenzen, wegen dem Versatz von Taktsignal und Datenwerten um einen Takt, nicht garantiert werden kann.

Es gibt eine Vielzahl von verschiedenen *SDRAM-Modulen* auf dem Markt, die sich bezüglich Speichergröße, Geschwindigkeit und Bauart unterscheiden. Daraus resultierend gibt es viele verschiedene Ansteuerungsarten für SDRAMs. Da der Aufwand der Ansteuerung sehr groß gewesen wäre, wenn alle möglichen Ansteuerungsvarianten im Speicherinterface ausprogrammiert worden wären, wurde ein bestimmter SDRAM-Typ ausgewählt und nur dessen Ansteuerung implementiert. Die Wahl des Speichers fiel auf die Type MT16LSDT6464AG-13E der Firma Micron, der folgende Eigenschaften hat:

- 512 MByte
- 64 Bit Datenbusbreite
- 3,3V Versorgung
- unbuffered
- 2 Bank-Aufbau
- Geschwindigkeit SDR, PC133, CL2
- 8k-Refresh
- Bauform 168-Pin DIMM

Zur Realisierung des Speicherinterfaces wurde wegen der hohen Flexibilität bezüglich der Programmierung bei geringen Kosten ein *FPGA-Baustein* verwendet. Als passender Typ wurde ein Baustein der

Altera ACEX 1K-Familie ausgewählt. Neben den Logik-Zellen besitzt dieser FPGA auch RAM-Zellen, die in diesem FPGA als Dual Port-RAM ausgeführt sind und somit auch als FIFO-Speicher verwendet werden können. Die Programmierung des FPGAs erfolgte in einer Kombination aus einer graphischen Eingabe von Design-Blöcken und der Hardwarebeschreibungssprache VHDL in der von Altera angebotenen Design-Software Quartus II. Quartus II bietet eine komplette Design-Umgebung von der Design-Eingabe bis hin zur Programmierung des FPGAs.

Den zentralen Teil der Funktion im FPGA bildet die *Steuerung*. Sie koordiniert das Zusammenspiel aller anderen Funktionseinheiten. Der A/D-Wandler liefert Messwerte mit 15 Bit (14 Datenbits und ein Overrange-Bit) sowie ein Taktsignal das anzeigt, wann die Daten übernommen werden können. Vier Messwerte werden in der *A/D-Wandler Schnittstelle* zu einem 60 Bit Datenwort zusammengefasst, dass dadurch nur mehr eine Datenrate von 20 MHz besitzt. Dieses Datenwort wird in einem *FIFO*-Speicher abgelegt, der als Zwischenspeicher notwendig ist. Die Daten werden vom A/D-Wandler nämlich kontinuierlich ausgegeben, aber das Schreiben in das SDRAM, den endgültigen Datenspeicher, kann aufgrund der Ansteuerung des Speichers und zwischendurch notwendiger Refresh-Zyklen nicht kontinuierlich erfolgen. Die Kommunikation des Speicherinterfaces mit dem SDRAM wird vom *SDRAM-Controller* durchgeführt. Der SDRAM-Controller übernimmt die Messdaten vom FIFO und speichert sie mit einem Burst 8-Write, bei dem acht Schreibzyklen hintereinander mit nur einer Adressierung erfolgen, im SDRAM. Er generiert selbständig alle  $7,8125 \mu\text{s}$  die notwendigen Refresh-Zyklen, damit die Daten nicht verloren gehen. Wenn die Messung beendet ist, bleiben die Daten im SDRAM gespeichert. Als Ende der Messung wurde das Ausbleiben von Messwerten für 100 Taktzyklen des  $33,3333 \text{ MHz}$ -Systemtaktes definiert. Wird das Speicherinterface mit dem *Selected*-Signal ausgewählt, liest der SDRAM-Controller die gespeicherten Datenworte mit einem Burst 1-Read aus dem SDRAM und liefert die Daten an die *Ausgangs-Schnittstelle*. In der Ausgangs-Schnittstelle wird das Datenwort wieder in die vier einzelnen Messwerte aufgeteilt, die im Handshake-Verfahren nacheinander am Ausgang ausgegeben werden. Die Ausgabe der Messergebnisse erfolgt in der Reihenfolge, in der sie ins SDRAM geschrieben wurde, d.h. es ist beim Auslesen keine explizite Adressierung eines Messwerts vorgesehen. Zum Auslesen der Daten aus dem Speicherinterface wird an die Ausgangs-Schnittstelle eine I/O-Karte der Firma ADLINK Technology Inc. angeschlossen. Die Daten können nur einmal aus dem Speicher ausgelesen werden, danach kann die ganze Schaltung mit dem *Reset*-Signal wieder in den Ausgangszustand gesetzt werden.

Damit der FIFO-Speicher nicht überläuft, muss schneller in das SDRAM geschrieben werden, als Daten vom A/D-Wandler geliefert werden. Der Systemtakt wurde deshalb mit  $33,3333 \text{ MHz}$  gewählt. Dieser Takt liegt an einem speziellen Pin des FPGAs, von dem die Laufzeit des Signals zur im FPGA enthaltenen PLL besonders kurz ist. Diese PLL für den Takt zu verwenden hat den Vorteil, dass die PLL das Taktsignal in ein im FPGA globales Clock-Signalnetz einspeist. Dadurch werden die Laufzeiten des Clock-Signals zu allen Gattern im FPGA genau gleich, d.h. alle Gatter schalten exakt synchron.

Mit Hilfe einer Serienschaltung jeweils zweier Flip Flops werden asynchrone Eingangssignale mit dem internen Takt synchronisiert. Dies ist notwendig, da Zustandswechsel von Eingangssignalen, die genau zum Zeitpunkt einer Taktflanke des internen Takts auftreten, unvorhersehbare Zustände von Gattern in FPGA hervorrufen können.

Um die Ergebnisse der Analyse, Synthese und vor allem des Fittings des Designs in Quartus II zu verbessern, wurden diverse Assignments festgelegt. Zum Beispiel wurden durch die exakte Festlegung der Platzierung von Logik-Gattern im FPGA geringe Laufzeiten zwischen den Gattern erreicht.

Als passender FPGA ergab sich am Ende des Designs ein Altera ACEX 1K mit 30.000 Logik-Gattern in der schnellsten Ausführung, d.h. mit dem Speed Grade 1.

Um die korrekte Funktion des FPGA-Designs zu testen, bevor das Speicherinterface in Hardware aufgebaut wurde, wurden verschiedene *Simulationen* in der FPGA-Design-Software Quartus II und mit Hilfe des reinen VHDL-Simulators ModelSim durchgeführt. Mit dem *Timing-Analyzer* von Quartus II wurde überprüft, dass für das Speicherinterface-Design im verwendeten FPGA die benötigten Timingverhalten bezüglich Setup- und Hold-Zeiten der Eingangssignale vom A/D-Wandler und maximaler Taktfrequenzen erreicht wurden. Mit dem Quartus II *Simulator* wurde das kompilierte FPGA-Design simuliert. Zur Überprüfung der Funktion des Speicherinterfaces wurde ein kurzer Messvorgang komplett simuliert. Zur Kontrolle des korrekten Timings zwischen dem SDRAM-Controller des Speicherinterfaces und einem SDRAM-Modul wurde der *VHDL-Simulator* ModelSim ALTERA 5.6a der Firma Model Technology Incorporated verwendet. Die Übergabe von Datenwerten zwischen Speicherinterface und SDRAM erfolgte in den Simulationen korrekt und es gab keine Fehlermeldungen wegen Verletzungen des Timing-Verhaltens

des SDRAMs.

Durch den Entwurf der Funktion des FPGAs und dessen anschliessende Simulation ergab sich der benötigte FPGA. Mit diesem FPGA musste eine *Schaltung* entworfen werden, welche die Programmierung des FPGAs ermöglicht, sowie eine Eingangs-Schnittstelle zum A/D-Wandler Evaluation Board und eine Ausgangs-Schnittstelle zum Auslesen der Daten besitzt. Für diese Schaltung musste zur Fertigung des Speicherinterfaces ein *Layout* entwickelt werden. Um in der ganzen Schaltung die vom FPGA und SDRAM benötigten LVTTTL-Pegel verwenden zu können, wurden bei den weiteren Schaltungskomponenten Low Voltage Bauteile mit einer Versorgungsspannung von 3,3 V ausgewählt. Zur Anzeige des Status des Speicherinterfaces sind *Leuchtdioden* vorgesehen. Bei dem verwendeten FPGA des Types ACEX 1K handelt es sich um einen auf CMOS SRAM-Elementen basierenden Typ, d.h. eine Programmierung des FPGAs geht mit dem Ausschalten der Versorgungsspannung verloren. Zur Programmierung des FPGAs sind die *JTAG*- und die *Byte Blaster-Schnittstelle* vorgesehen. Um das Speicherinterface nicht jedesmal vom Computer über eine der beiden Schnittstellen programmieren zu müssen, wurde ein *Configuration Device* des Typs EPC2 verwendet. Der 33,3333 MHz Takt des Systems wird von einem *Oszillator* geliefert und von einem *Takt-Generator* für den FPGA und das SDRAM aufbereitet. Vom Takt-Generator werden vier zum Signal des Oszillators synchrone Taktsignale mit einer sehr geringen Phasenverschiebung erzeugt, welche die vier Takt-Leitungen des SDRAMs ansteuern. Ausserdem ein zum Signal des Oszillators invertiertes Taktsignal, das den FPGA ansteuert. Das Layout konnte auf einer, wegen der niedrigen Kosten gewünschten, gewöhnlichen zweiseitigen Leiterplatte der Stärke 1,6 mm mit Kupferauflagen von 35  $\mu\text{m}$  realisiert werden, da ein Grossteil der Entflechtungsarbeit durch Quartus II durch die richtige Wahl der Pinvergabe im FPGA durchgeführt wurde.

Die Platine wurde wegen der feinen Leiterstruktur nicht selbst hergestellt, sondern von der Firma Q-print electronic GmbH gefertigt. Nach dem Bestücken der Leiterplatte wurde mit dem fertigen Speicherinterface ein einfaches Messsystem zur *Überprüfung* der Funktion aufgebaut. Bei den durchgeführten Tests konnte keine Fehlfunktion des Speicherinterfaces festgestellt werden.

# Anhang A

## Bedienung des Speicherinterface

### A.1 Programmierung des FPGAs über die ByteBlaster-Schnittstelle

Zur Programmierung des FPGAs über die ByteBlaster-Schnittstelle muss das Configuration Device IC5 aus dem Sockel entfernt werden!

1. Starten der Altera Design Software Quartus II.
2. Öffnen des Projekts Speicherinterface.
3. Den Programmer unter  $\rightarrow$  *Tools/Programmer* starten.
4. Kontrollieren ob als Hardware “ByteBlaster an LPT1” eingetragen ist und den Mode auf “Passive Serial” stellen.
5. Den ByteBlaster an die Parallele Schnittstelle und an den ByteBlaster-Stecker CON4 anstecken.
6. Die Spannungsversorgungen des Speicherinterfaces in der Reihenfolge 2,5V - 3,3V - 5V einschalten. (Die benötigten Leistungen eines Speicherinterfaces mit SDRAM und A/D-Wandler betragen: 2,5V ca. 80mA; 3,3V ca. 330mA; 5V ca. 330mA)
7. Das Programmieren des Files “..\Speicherinterface.sof” in den FPGA mit  $\rightarrow$  *Processing/Start Programming* starten.
8. Im Systemfenster von Quartus überprüfen ob Programmierung erfolgreich war.

### A.2 Programmierung des Configuration Devices und des FPGAs über die JTAG-Schnittstelle

Über die JTAG-Schnittstelle können der FPGA und das Configuration Device einzeln oder beide gemeinsam programmiert werden.

1. Starten der Altera Design Software Quartus II.
2. Öffnen des Projekts Speicherinterface.
3. Unter  $\rightarrow$  *Files/Open...* das Programming File “Speicherinterface\_für\_EPC2.cdf” öffnen.

4. Den ByteBlaster an die Parallele Schnittstelle und an den JTAG-Stecker CON3 anstecken.
5. Die Spannungsversorgungen des Speicherinterfaces in der Reihenfolge 2,5V - 3,3V - 5V einschalten. (Die benötigten Leistungen eines Speicherinterfaces mit SDRAM und A/D-Wandler betragen: 2,5V ca. 80mA; 3,3V ca. 330mA; 5V ca. 330mA)
6. Das Programmieren des Files "...\Speicherinterface.sof" in den FPGA bzw. "...\Speicherinterface.pof" in das Configuration Device mit  $\rightarrow$ Processing/Start Programming starten.
7. Zuerst wird das Configuration Device gelöscht und überprüft, ob es wirklich leer ist. Als nächstes wird das Configuration Device und der FPGA programmiert. Zuletzt wird noch die Programmierung des Configuration Devices überprüft.
8. Im Systemfenster von Quartus überprüfen, ob die Programmierung erfolgreich war.

### A.3 Messen von Daten

Anhand des Blockschaltbildes des Messsystems (Abbildung A.1) wird im folgenden erklärt, wie eine Messung mit zwei Speicherinterfaces ablaufen soll.

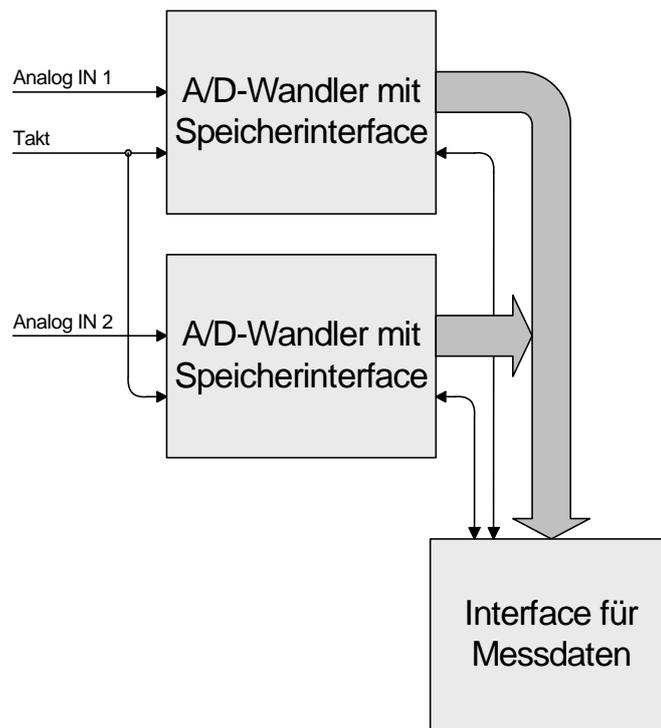


Abbildung A.1: Blockschaltbild des Messsystems

Alle Ausgänge eines Speicherinterfaces sind Tri State-Ausgänge. Mit  $SELECTED=1$  kann ein Speicherinterface jederzeit ausgewählt werden, man erkennt dann an den STATUS-Leitungen den momentanen Zustand. Während der Daten-Übernahme vom A/D-Wandler können mit  $SELECTED\_DOUT\_ADR=1$  die höchsten 15 Bits des 26 Bits langen Zählers der Messwerte an DOUT ausgegeben werden.

1. Die Spannungsversorgungen in der Reihenfolge 2,5V - 3,3V - 5V einschalten. (Die benötigten Leistungen eines Speicherinterfaces mit SDRAM und A/D-Wandler betragen: 2,5V ca. 80mA; 3,3V ca. 330mA; 5V ca. 330mA)
2. Die Programmierung der FPGA-Bausteine erfolgt selbständig durch die Configuration Devices (maximale Dauer 50 ms)
3. Die Speicherinterface mit einem mindestens 50 ns langen RESETIN=1 in einen definierten Anfangszustand bringen.
4. Überprüfen ob die Speicherinterfaces die Initialisierung (STATUS\_INIT\_TRI=1) abgeschlossen haben (maximale Dauer 220  $\mu$ s) und zur Aufnahme von Daten bereit sind (STATUS\_SDRAMEMPTY\_TRI=1).
5. Die A/D-Wandler in Betrieb nehmen (80 MHz-Taktsignal an ENC).
6. Dauer der Übernahme der Daten der A/D-Wandler von den Speicherinterfaces mit dem Signal DRY\_ENABLE=1 steuern. Die Messwerte werden nun in die SDRAMs geschrieben und der Status der Speicherinterfaces wechselt zu STATUS\_SDRAMWRITE\_TRI=1.
7. Nach dem Ende der Messung durch DRY\_ENABLE=0 oder wenn das SDRAM voll geschrieben ist, ist STATUS\_SDRAMFULL\_TRI=1 und alle anderen STATUS-Leitungen sind 0. Ist ein Fehler aufgetreten, so ist STATUS\_ERROR\_TRI=1 und die Messung muss wiederholt werden!
8. Zum Auslesen der Daten aus dem SDRAM muss das gewünschte Speicherinterface mit SELECTED=1 ausgewählt werden. Der Status wechselt zu STATUS\_SDRAMREAD\_TRI=1.
9. Das Speicherinterface zeigt mit DATA\_AVAILABLE=1, dass Daten zum Auslesen im SDRAM vorhanden sind, legt den ersten Datenwert an die DOUT-Leitungen und setzt DOUT\_RDREQ auf 1.
10. Die Ausleseschaltung muß die Datenübernahme mit DOUT\_RDACK=1 bestätigen.
11. Das Speicherinterface setzt daraufhin DOUT\_RDREQ auf 0 und bereitet den nächsten Datenwert vor.
12. Wenn die Ausleseschaltung DOUT\_RDACK=0 setzt, wird der nächste Datenwert an die DOUT-Leitungen angelegt und DOUT\_RDREQ wieder auf 1 gesetzt. (vgl. 9.)
13. Dieser Handshakevorgang (9. bis 12.) zur Datenausgabe kann jederzeit durch SELECTED=0 unterbrochen werden und mit SELECTED=1 an der selben Stelle fortgesetzt werden. Es können somit die beiden Speicherinterface entweder hintereinander komplett ausgelesen werden oder die Auslesevorgänge ineinander verschachtelt werden. Ist ein Fehler aufgetreten, so ist STATUS\_ERROR=1 und die Messung muss wiederholt werden!
14. Sind keine Messwerte mehr im SDRAM, so wird DATA\_AVAILABLE=0 und DOUT\_RDREQ bleibt 0. Außerdem sind alle STATUS-Leitungen=0.
15. Für eine erneute Messung müssen die Speicherinterface mit RESETIN=1 wieder in den Anfangszustand gesetzt werden.

# Anhang B

## Steckerbelegung

Anhand des B.1 ist die Lage der Stecker auf der Platine zu erkennen.

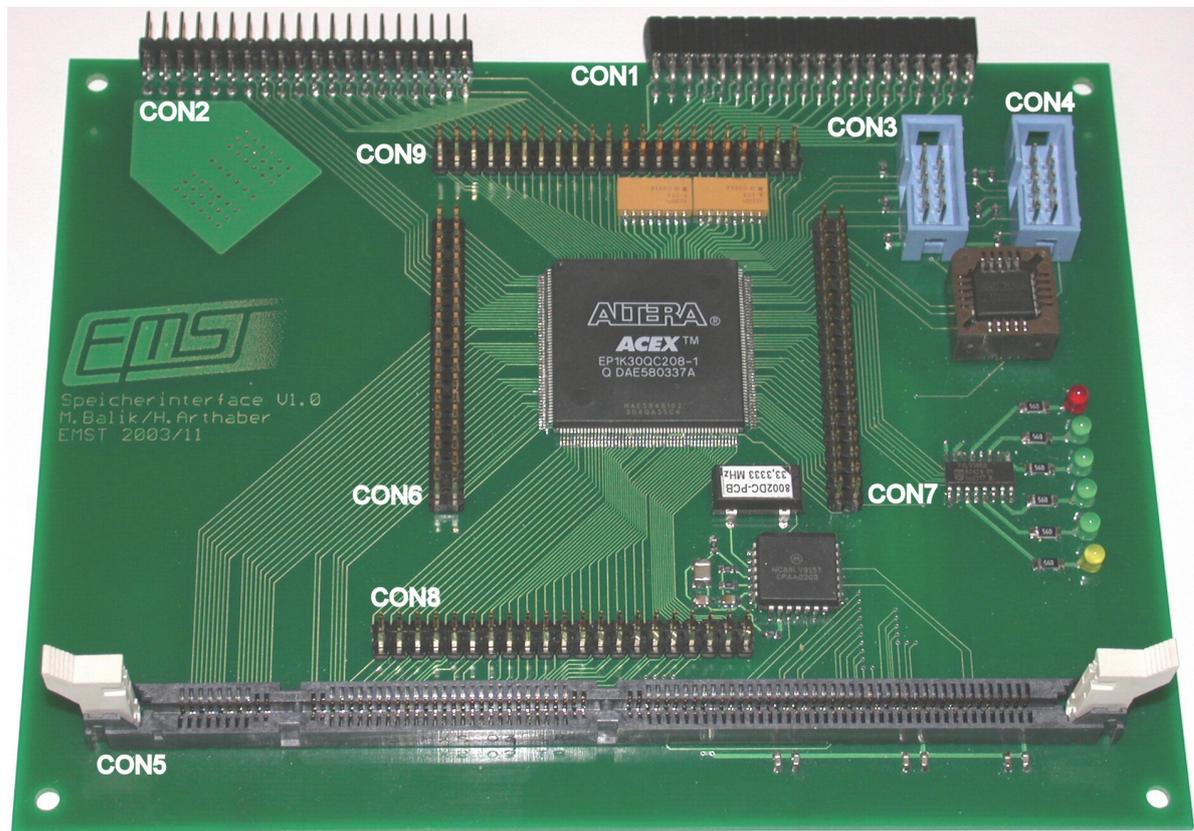


Abbildung B.1: Stecker des Speicherinterfaces

Anschließend folgen Tabellen mit der Belegung der einzelnen Stecker.

## B.1 Stecker CON1

Der Stecker CON1 ist der Eingangsstecker, an dem die vom A/D-Wandler gelieferten Daten eingelesen werden.

Pin	Richtung	Name	Funktion
1	-	-	
2	-	GND	
3	-	-	
4	IN	DRY	Takt der Daten vom A/D-Wandler (DataReady)
5	-	-	
6	-	GND	
7	-	GND	
8	IN	DIN13	Eingangs-Datenbit (MSB) vom A/D-Wandler
9	-	GND	
10	IN	DIN12	Eingangs-Datenbit vom A/D-Wandler
11	-	GND	
12	IN	DIN11	Eingangs-Datenbit vom A/D-Wandler
13	-	GND	
14	IN	DIN10	Eingangs-Datenbit vom A/D-Wandler
15	-	GND	
16	IN	DIN9	Eingangs-Datenbit vom A/D-Wandler
17	-	GND	
18	IN	DIN8	Eingangs-Datenbit vom A/D-Wandler
18	-	GND	
20	IN	DIN7	Eingangs-Datenbit vom A/D-Wandler
21	-	GND	
22	IN	DIN6	Eingangs-Datenbit vom A/D-Wandler
23	-	GND	
24	IN	DIN5	Eingangs-Datenbit vom A/D-Wandler
25	-	GND	
26	IN	DIN4	Eingangs-Datenbit vom A/D-Wandler
27	-	GND	
28	IN	DIN3	Eingangs-Datenbit vom A/D-Wandler
29	-	GND	
30	IN	DIN2	Eingangs-Datenbit vom A/D-Wandler
31	-	GND	
32	IN	DIN1	Eingangs-Datenbit vom A/D-Wandler
33	-	GND	
34	IN	DIN0	Eingangs-Datenbit (LSB) vom A/D-Wandler
35	-	GND	
36	-	GND	
37	-	GND	
38	-	GND	
39	-	-	
40	IN	OVR	Daten vom A/D-Wandler ungültig (OVerRange)

Tabelle B.1: Stecker CON1

## B.2 Stecker CON2

Der Stecker CON2 ist der Ausgangsstecker, an dem die Daten ausgegeben werden, die Kommunikation mit der Steuerung erfolgt und über den das Speicherinterface mit Strom versorgt wird.

Pin	Richtung	Name	Funktion
1	IN	+3V3_IN	
2	IN	+3V3_IN	
3	IN	+3V3_IN	
4	IN	+3V3_IN	
5	IN	+2V5_IN	
6	IN	+2V5_IN	
7	IN	+2V5_IN	
8	IN	+2V5_IN	
9	IN	GND_IN	
10	IN	GND_IN	
11	IN	GND_IN	
12	IN	GND_IN	
13	OUT	STATUS_ERROR_TRI	Ausgangs-Statusanzeige
14	OUT	STATUS_SDRAMREAD_TRI	Ausgangs-Statusanzeige
15	OUT	STATUS_SDRAMFULL_TRI	Ausgangs-Statusanzeige
16	OUT	STATUS_SDRAMWRITE_TRI	Ausgangs-Statusanzeige
17	OUT	STATUS_SDRAMEMPTY_TRI	Ausgangs-Statusanzeige
18	OUT	STATUS_INIT_TRI	Ausgangs-Statusanzeige
19	OUT	DOUT14	Ausgangs-Datenbit (Ovrrange)
20	OUT	DOUT13	Ausgangs-Datenbit (MSB)
21	OUT	DOUT12	Ausgangs-Datenbit
22	OUT	DOUT11	Ausgangs-Datenbit
23	OUT	DOUT10	Ausgangs-Datenbit
24	OUT	DOUT9	Ausgangs-Datenbit
25	OUT	DOUT8	Ausgangs-Datenbit
26	OUT	DOUT7	Ausgangs-Datenbit
27	OUT	DOUT6	Ausgangs-Datenbit
28	OUT	DOUT5	Ausgangs-Datenbit
29	OUT	DOUT4	Ausgangs-Datenbit
30	OUT	DOUT3	Ausgangs-Datenbit
31	OUT	DOUT2	Ausgangs-Datenbit
32	OUT	DOUT1	Ausgangs-Datenbit
33	OUT	DOUT0	Ausgangs-Datenbit (LSB)
34	IN	DOUT_RDACK	Acknowledge für Ausgangsdaten-Handshake
35	OUT	DOUT_RDREQ	Request für Ausgangsdaten-Handshake
36	OUT	DATA_AVAILABLE	Ausgangsdaten vorhanden
37	IN	SELECTED_DOUT_ADR	Speicherinterface ausgewählt (Anzeige der Anzahl der gemessenen Daten)
38	IN	SELECTED	Speicherinterface ausgewählt
39	IN	RESETIN	Reset
40	IN	DRY_ENABLE	Enable für Daten vom A/D-Wandler

Tabelle B.2: Stecker CON2

### B.3 Stecker CON3

Der Stecker CON3 ist der JTAG-Anschluss (von IEEE genormte Schnittstelle) zur Programmierung des FPGAs und des Configuration Devices und zum Auslesen der Daten beim Boundary-Scan Test.

Pin	Richtung	Name	Funktion
1	IN	TCK	Takt-Signal (Test Clock Input)
2	-	GND	
3	OUT	TDO	Daten vom FPGA und Configuration Device (Test Data Output)
4	-	+3V3	
5	IN	TMS	Steuerungs-Signal (Test Mode Select)
6	-	-	
7	-	-	
8	-	-	
9	IN	TDI	Daten zum FPGA und Configuration Device (Test Data Input)
10	-	GND	

Tabelle B.3: Stecker CON3

### B.4 Stecker CON4

Der Stecker CON4 ist der ByteBlaster-Anschluss (von Altera definierte Schnittstelle) zur Programmierung des FPGAs.

Pin	Richtung	Name	Funktion
1	IN	DCLK	Takt-Signal
2	-	GND	
3	INOUT	CONF_DONE	Steuerungs-Signal
4	-	+3V3	
5	IN	/CONFIG	Steuerungs-Signal
6		-	
7	INOUT	/STATUS	Steuerungs-Signal
8		-	
9	IN	DATA0	Daten zum FPGA
10	-	GND	

Tabelle B.4: Stecker CON4

## B.5 Stecker CON5

Der Stecker CON5 ist der SDRAM-Stecker, ein DIMM 168-Pin Sockel für ein ungebuffertes 3,3V SDRAM-Modul. In der nachfolgenden Tabelle sind nur die im Speicherinterface verwendeten Pins angeführt.

Pin	Name	Pin	Name	Pin	Name	Pin	Name
1	GND	43	GND	85	GND	127	GND
2	DQ0	44	-	86	DQ32	128	CKE
3	DQ1	45	/CS2	87	DQ33	129	/CS3
4	DQ2	46	GND	88	DQ34	130	GND
5	DQ3	47	GND	89	DQ35	131	GND
6	+3V3	48	-	90	+3V3	132	-
7	DQ4	49	+3V3	91	DQ36	133	+3V3
8	DQ5	50	-	92	DQ37	134	-
9	DQ6	51	-	93	DQ38	135	-
10	DQ7	52	-	94	DQ39	136	-
11	DQ8	53	-	95	DQ40	137	-
12	GND	54	GND	96	GND	138	GND
13	DQ9	55	DQ16	97	DQ41	139	DQ48
14	DQ10	56	DQ17	98	DQ42	140	DQ49
15	DQ11	57	DQ18	99	DQ43	141	DQ50
16	DQ12	58	DQ19	100	DQ44	142	DQ51
17	DQ13	59	+3V3	101	DQ45	143	+3V3
18	+3V3	60	DQ20	102	+3V3	144	DQ52
19	DQ14	61	-	103	DQ46	145	-
20	DQ15	62	-	104	DQ47	146	-
21	-	63	CKE	105	-	147	-
22	-	64	GND	106	-	148	GND
23	GND	65	DQ21	107	GND	149	DQ53
24	-	66	DQ22	108	-	150	DQ54
25	-	67	DQ23	109	-	151	DQ55
26	+3V3	68	GND	110	+3V3	152	GND
27	/WE	69	DQ24	111	/CAS	153	DQ56
28	GND	70	DQ25	112	GND	154	DQ57
29	GND	71	DQ26	113	GND	155	DQ58
30	/CS0	72	DQ27	114	/CS1	156	DQ59
31	-	73	+3V3	115	/RAS	157	+3V3
32	GND	74	DQ28	116	GND	158	GND
33	A0	75	DQ29	117	A1	159	GND
34	A2	76	DQ30	118	A3	160	GND
35	A4	77	DQ31	119	A5	161	GND
36	A6	78	GND	120	A7	162	GND
37	A8	79	CLK2	121	A9	163	CLK3
38	A10	80	-	122	BA0	164	-
39	BA1	81	-	123	A11	165	-
40	+3V3	82	-	124	+3V3	166	-
41	+3V3	83	-	125	CLK1	167	-
42	CLK0	84	+3V3	126	A12	168	+3V3

Tabelle B.5: Stecker CON5

## B.6 Stecker CON6

Der Stecker CON6 ist eine Stiftleiste zur Überprüfung der Pegel an den FPGA-Pins. Er befindet sich auf der Platine **links** vom FPGA.

Pin	Richtung	Name	Funktion
1	INOUT	/STATUS	ByteBlaster Steuerungs-Signal
2	IN	TRST	FPGA Steuerungs-Signal (liegt auf +3V3!)
3	IN	TMS	JTAG Steuerungs-Signal
4	INOUT	DQ42	SDRAM Datenbit
5	INOUT	DQ10	SDRAM Datenbit
6	INOUT	DQ41	SDRAM Datenbit
7	INOUT	DQ9	SDRAM Datenbit
8	INOUT	DQ40	SDRAM Datenbit
9	INOUT	DQ8	SDRAM Datenbit
10	INOUT	DQ39	SDRAM Datenbit
11	INOUT	DQ7	SDRAM Datenbit
12	INOUT	DQ38	SDRAM Datenbit
13	INOUT	DQ6	SDRAM Datenbit
14	INOUT	DQ37	SDRAM Datenbit
15	INOUT	DQ5	SDRAM Datenbit
16	INOUT	DQ36	SDRAM Datenbit
17	INOUT	DQ4	SDRAM Datenbit
18	INOUT	DQ35	SDRAM Datenbit
19	INOUT	DQ3	SDRAM Datenbit
20	INOUT	DQ34	SDRAM Datenbit
21	INOUT	DQ2	SDRAM Datenbit
22	INOUT	DQ33	SDRAM Datenbit
23	INOUT	DQ1	SDRAM Datenbit
24	INOUT	DQ32	SDRAM Datenbit
25	INOUT	DQ0	SDRAM Datenbit
26	-	PIN15	nicht verwendeter FPGA Pin
27	-	PIN14	nicht verwendeter FPGA Pin
28	-	PIN13	nicht verwendeter FPGA Pin
29	OUT	STATUS_ERROR_TRI	Ausgangs-Statusanzeige
30	OUT	STATUS_SDRAMREAD_TRI	Ausgangs-Statusanzeige
31	OUT	STATUS_SDRAMFULL_TRI	Ausgangs-Statusanzeige
32	OUT	STATUS_SDRAMWRITE_TRI	Ausgangs-Statusanzeige
33	OUT	STATUS_SDRAMEMPTY_TRI	Ausgangs-Statusanzeige
34	OUT	STATUS_INIT_TRI	Ausgangs-Statusanzeige
35	OUT	TDO	JTAG Daten vom FPGA
36	OUT	/CEO	nicht verwendeter FPGA Pin
37	INOUT	CONF_DONE	ByteBlaster Steuerungs-Signal
38	IN	TCK	JTAG Takt-Signal

Tabelle B.6: Stecker CON6

## B.7 Stecker CON7

Der Stecker CON7 ist eine Stiftleiste zur Überprüfung der Pegel an den FPGA-Pins. Er befindet sich auf der Platine **rechts** vom FPGA.

Pin	Richtung	Name	Funktion
1	IN	/CONFIG	ByteBlaster Steuerungs-Signal
2	IN	MSEL1	FPGA Steuerungs-Signal (liegt auf 0V!)
3	IN	MSEL0	FPGA Steuerungs-Signal (liegt auf 0V!)
4	INOUT	DQ19	SDRAM Datenbit
5	INOUT	DQ51	SDRAM Datenbit
6	INOUT	DQ20	SDRAM Datenbit
7	INOUT	DQ52	SDRAM Datenbit
8	INOUT	DQ21	SDRAM Datenbit
9	INOUT	DQ53	SDRAM Datenbit
10	INOUT	DQ22	SDRAM Datenbit
11	INOUT	DQ54	SDRAM Datenbit
12	INOUT	DQ23	SDRAM Datenbit
13	INOUT	DQ55	SDRAM Datenbit
14	INOUT	DQ24	SDRAM Datenbit
15	INOUT	DQ56	SDRAM Datenbit
16	INOUT	DQ25	SDRAM Datenbit
17	INOUT	DQ57	SDRAM Datenbit
18	INOUT	DQ26	SDRAM Datenbit
19	INOUT	DQ58	SDRAM Datenbit
20	INOUT	DQ27	SDRAM Datenbit
21	INOUT	DQ59	SDRAM Datenbit
22	INOUT	DQ28	SDRAM Datenbit
23	INOUT	DQ29	SDRAM Datenbit
24	INOUT	DQ30	SDRAM Datenbit
25	INOUT	DQ31	SDRAM Datenbit
26	-	PIN141	nicht verwendeter FPGA Pin
27	-	PIN142	nicht verwendeter FPGA Pin
28	OUT	STATUS_INIT	LED-Statusanzeige
29	OUT	STATUS_SDRAMEMPTY	LED-Statusanzeige
30	OUT	STATUS_SDRAMWRITE	LED-Statusanzeige
31	OUT	STATUS_SDRAMFULL	LED-Statusanzeige
32	OUT	STATUS_SDRAMREAD	LED-Statusanzeige
33	OUT	STATUS_ERROR	LED-Statusanzeige
34	IN	TDO_I	JTAG Daten zum FPGA
35	IN	/CE	FPGA Steuerungs-Signal (liegt auf 0V!)
36	IN	DCLK	ByteBlaster Takt-Signal
37	IN	DATA0	ByteBlaster Daten zum FPGA
38	-	-	-

Tabelle B.7: Stecker CON7

## B.8 Stecker CON8

Der Stecker CON8 ist eine Stiftleiste zur Überprüfung der Pegel an den FPGA-Pins. Er befindet sich auf der Platine **unterhalb** des FPGA.

Pin	Richtung	Name	Funktion
1	INOUT	DQ11	SDRAM Datenbit
2	INOUT	DQ43	SDRAM Datenbit
3	INOUT	DQ12	SDRAM Datenbit
4	INOUT	DQ44	SDRAM Datenbit
5	INOUT	DQ13	SDRAM Datenbit
6	INOUT	DQ45	SDRAM Datenbit
7	INOUT	DQ14	SDRAM Datenbit
8	INOUT	DQ46	SDRAM Datenbit
9	INOUT	DQ15	SDRAM Datenbit
10	INOUT	DQ47	SDRAM Datenbit
11	OUT	/WE	SDRAM Steuerungs-Signal
12	OUT	/CAS	SDRAM Steuerungs-Signal
13	OUT	/CS0	SDRAM Steuerungs-Signal
14	OUT	/CS1	SDRAM Steuerungs-Signal
15	OUT	/RAS	SDRAM Steuerungs-Signal
16	OUT	A0	SDRAM Adressbit
17	OUT	A1	SDRAM Adressbit
18	OUT	A2	SDRAM Adressbit
19	OUT	A3	SDRAM Adressbit
20	OUT	A4	SDRAM Adressbit
21	-	PIN78	nicht verwendeter FPGA Pin
22	IN	CLKIN	FPGA Takt-Signal
23	-	PIN80	nicht verwendeter FPGA Pin
24	OUT	A5	SDRAM Adressbit
25	OUT	A6	SDRAM Adressbit
26	OUT	A7	SDRAM Adressbit
27	OUT	A8	SDRAM Adressbit
28	OUT	A9	SDRAM Adressbit
29	OUT	A10	SDRAM Adressbit
30	OUT	BA0	SDRAM Adressbit
31	OUT	BA1	SDRAM Adressbit
32	OUT	A11	SDRAM Adressbit
33	OUT	A12	SDRAM Adressbit
34	OUT	CKE	SDRAM Steuerungs-Signal
35	OUT	/CS2	SDRAM Steuerungs-Signal
36	OUT	/CS3	SDRAM Steuerungs-Signal
37	INOUT	DQ16	SDRAM Datenbit
38	INOUT	DQ48	SDRAM Datenbit
39	INOUT	DQ17	SDRAM Datenbit
40	INOUT	DQ49	SDRAM Datenbit
41	INOUT	DQ18	SDRAM Datenbit
42	INOUT	DQ50	SDRAM Datenbit

Tabelle B.8: Stecker CON8

## B.9 Stecker CON9

Der Stecker CON9 ist eine Stiftleiste zur Überprüfung der Pegel an den FPGA-Pins. Er befindet sich auf der Platine **oberhalb** des FPGA.

Pin	Richtung	Name	Funktion
1	OUT	DOUT14	Ausgangs-Datenbit (Overrange)
2	OUT	DOUT13	Ausgangs-Datenbit (MSB)
3	OUT	DOUT12	Ausgangs-Datenbit
4	OUT	DOUT11	Ausgangs-Datenbit
5	OUT	DOUT10	Ausgangs-Datenbit
6	OUT	DOUT9	Ausgangs-Datenbit
7	OUT	DOUT8	Ausgangs-Datenbit
8	OUT	DOUT7	Ausgangs-Datenbit
9	OUT	DOUT6	Ausgangs-Datenbit
10	OUT	DOUT5	Ausgangs-Datenbit
11	OUT	DOUT4	Ausgangs-Datenbit
12	OUT	DOUT3	Ausgangs-Datenbit
13	OUT	DOUT2	Ausgangs-Datenbit
14	OUT	DOUT1	Ausgangs-Datenbit
15	OUT	DOUT0	Ausgangs-Datenbit (LSB)
16	IN	DOUT_RDACK	Acknowledge für Ausgangsdaten-Handshake
17	OUT	DOUT_RDREQ	Request für Ausgangsdaten-Handshake
18	OUT	DATA_AVAILABLE	Ausgangsdaten vorhanden
19	IN	SELECTED_DOUT_ADR	Speicherinterface zur Ausgabe der Anzahl der gemessenen Daten ausgewählt
20	IN	SELECTED	Speicherinterface ausgewählt
21	IN	RESETIN	Reset
22	-	PIN183	nicht verwendeter FPGA Pin
23	IN	DRY_ENABLE	Enable für Daten vom A/D-Wandler
24	IN	OVR	Daten vom A/D-Wandler ungültig (OverRange)
25	IN	DIN0	Eingangs-Datenbit (LSB) vom A/D-Wandler
26	IN	DIN1	Eingangs-Datenbit vom A/D-Wandler
27	IN	DIN2	Eingangs-Datenbit vom A/D-Wandler
28	IN	DIN3	Eingangs-Datenbit vom A/D-Wandler
29	IN	DIN4	Eingangs-Datenbit vom A/D-Wandler
30	IN	DIN5	Eingangs-Datenbit vom A/D-Wandler
31	IN	DIN6	Eingangs-Datenbit vom A/D-Wandler
32	IN	DIN7	Eingangs-Datenbit vom A/D-Wandler
33	IN	DIN8	Eingangs-Datenbit vom A/D-Wandler
34	IN	DIN9	Eingangs-Datenbit vom A/D-Wandler
35	IN	DIN10	Eingangs-Datenbit vom A/D-Wandler
36	IN	DIN11	Eingangs-Datenbit vom A/D-Wandler
37	IN	DIN12	Eingangs-Datenbit vom A/D-Wandler
38	IN	DIN13	Eingangs-Datenbit (MSB) vom A/D-Wandler
39	IN	DRY	Takt der Daten vom A/D-Wandler (DataReady)
40	-	PIN161	nicht verwendeter FPGA Pin
41	-	PIN160	nicht verwendeter FPGA Pin
42	-	PIN159	nicht verwendeter FPGA Pin
43	-	PIN158	nicht verwendeter FPGA Pin
44	-	PIN157	nicht verwendeter FPGA Pin

Tabelle B.9: Stecker CON9

# Anhang C

## Schaltplan

Anschliessend folgen die kompletten Schaltpläne des Speicherinterfaces.

Teil 1 ist in eine linke (Abbildung C.1) und rechte (Abbildung C.2) Hälfte geteilt und zeigt die zur Funktion notwendige Schaltung.

Teil 2 (Abbildung C.3) zeigt die Stiftleisten CON6 - CON9, die nur zur Überprüfung der Pegel an den FPGA-Pins dienen und zur Funktion der Schaltung nicht eingelötet werden müssen.

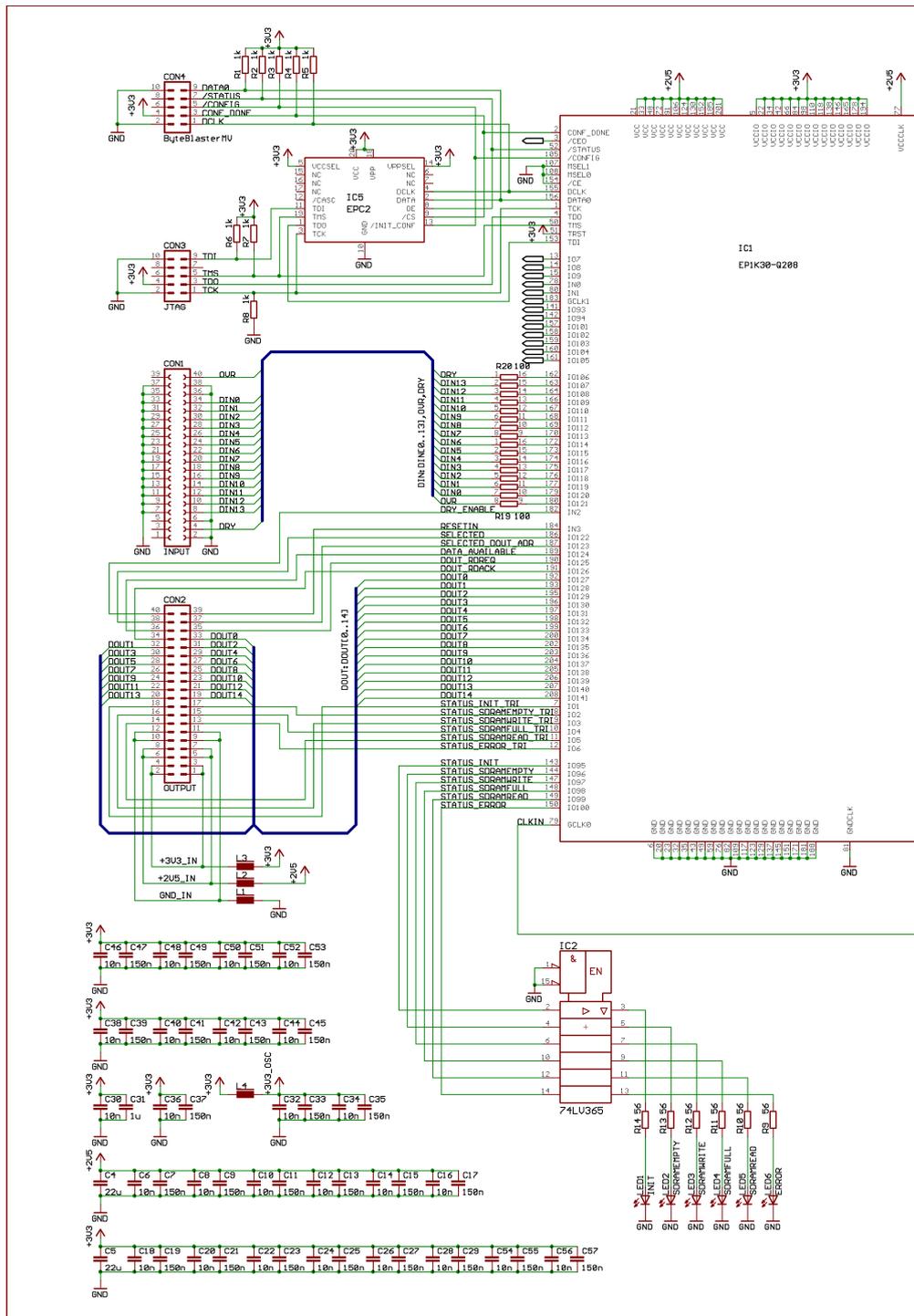
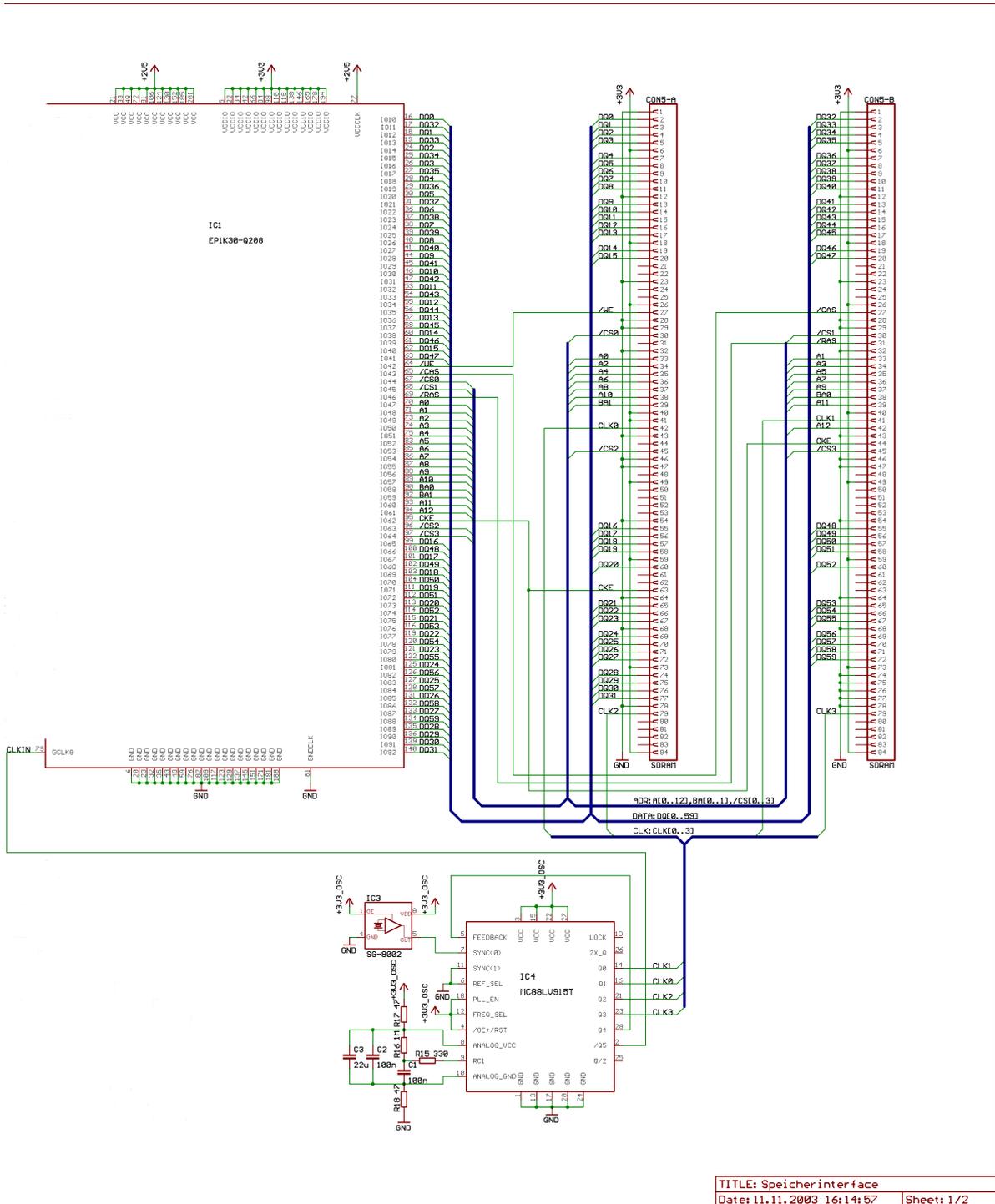
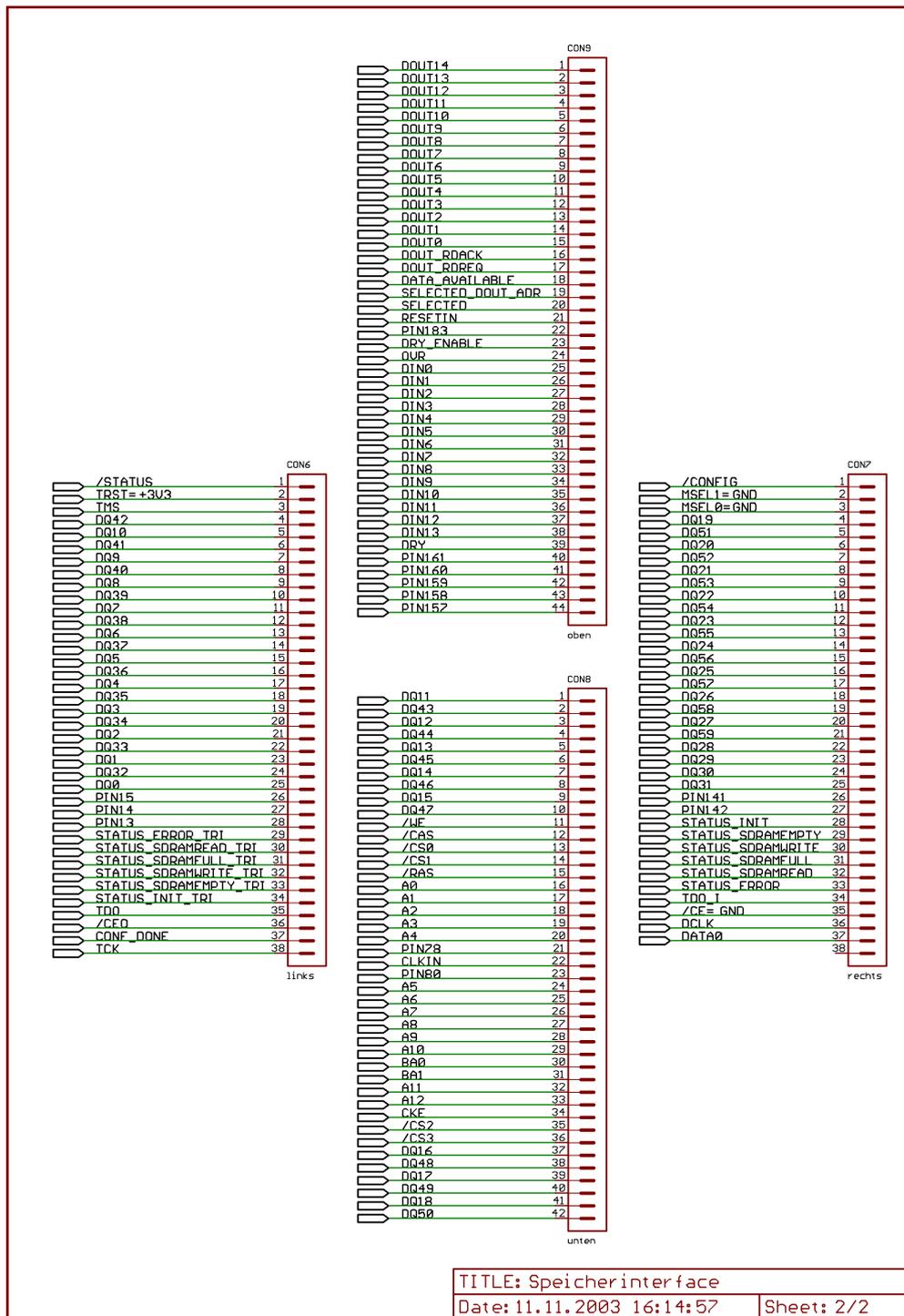


Abbildung C.1: Schaltplan Teil1 links





TITLE: Speicherinterface  
 Date: 11.11.2003 16:14:57 Sheet: 2/2

Abbildung C.3: Schaltplan Teil 2

# Anhang D

## Bauteilliste

Zum Bau des Speicherinterfaces wurden folgende Bauteile verwendet:

Nummer	Bezeichnung	Hersteller Typen Nummer
C1, C2	Keramik-Vielschicht-Chip-Kondensator Typ805, 100nF, 0805	Kemet 0805/X7R
C3, C4, C5	Keramik-Vielschicht-Chip-Kondensator GRM200, 22 $\mu$ F, 1210	Murata GJ232NF51A226ZD01L
C6, C8, C10, C12, C14, C16, C18, C20, C22, C24, C26, C28, C30, C32, C34, C36, C38, C40, C42, C44, C46, C48, C50, C52, C54, C56	Keramik-Vielschicht-Chip-Kondensator GRM18, 10nF, 0603	Murata GRM188R71H103KA01D
C7, C9, C11, C13, C15, C17, C19, C21, C23, C25, C27, C29, C33, C35, C37, C39, C41, C43, C45, C47, C49, C51, C53, C55, C57	Keramik-Vielschicht-Chip-Kondensator GRM18, 150nF, 0603	Murata GRM188R71A154KA01D
C31	Keramik-Vielschicht-Chip-Kondensator GRM, 1 $\mu$ F, 0805	Murata GRM21BF51E105ZA12L
CON1	Buchsenleiste 0,1", 2-reihig, 90°, 100 Pole	Preci-Dip 803-91-100-20-001001
CON2	Stiftleiste 0,1", 2-reihig, 90°, 2x20 Pole	Preci-Dip 330.40.2 L 02 902
CON3, CON4	IDC-Stiftleiste "Low Profile"	tyco Electronics 609-1027
CON5	DIMM-Sockel 168 Pins ungepuffert 3,3V	Molex 717360011
CON6, CON7	Stiftleiste 0,1", 2-reihig, gerade, 2x20 Pole	Preci-Dip 330.40.2 Y 02 803
CON8, CON9	Stiftleiste 0,1", 2-reihig, gerade, 2x25 Pole	Preci-Dip 330.50.2 Y 02 803

Tabelle D.1: Bauteilliste Teil 1

Nummer	Bezeichnung	Hersteller Typen Nummer
IC1	Altera ACEK1K, 30.000 Gates, Speed Grade 1, 208 Pin PQFP	Altera EP1K30QC208-1
IC2	Hex buffer/line driver, 16 Pin Plastic Small Outline	Philips 74LV365D
IC3	Crystal Oszillator, 33.3333MHz, 8 Pin DIP	Epson SG8002DC
IC4	Low Voltage Low Skew CMOS PLL Clock Driver, 28 Pin PLCC	Motorola MC88LV915T
IC5	Altera EPC2, 20 Pin PLCC	Altera EPC2LC20
L1, L2, L3, L4	SMD-EMV-Ferritfilter für Hochstromanwendung	Würth Elektronik 742792515
LED1	LED Ø3mm gelb	Kingbright L-934YT
LED2, LED3, LED4, LED5	LED Ø3mm grün	Kingbright L-934GD
LED6	LED Ø3mm rot	Kingbright L-934HD
R1, R2, R3, R4, R5, R6, R7, R8	SMD Widerstand 1%, D11-CR0603, 1k $\Omega$	Draloric D11-CR0603
R9, R10, R11, R12, R13, R14	SMD Widerstand 5%, CR-1206, 56 $\Omega$	Draloric CR-1206
R15	SMD Widerstand 1%, D11-CR0603, 330 $\Omega$	Draloric D11-CR0603
R16	SMD Widerstand 1%, D11-CR0603, 1M $\Omega$	Draloric D11-CR0603
R17, R18	SMD Widerstand 1%, D11-CR0603, 47 $\Omega$	Draloric D11-CR0603
R19, R20	SOP-Widerstands-Netzwerk T01, 100 $\Omega$	Bourns 4816P-T01-101

Tabelle D.2: Bauteilliste Teil 2

Als A/D-Wandler wurde ein "Analog Devices AD6645 Evaluation Board Rev D" verwendet. Das verwendete SDRAM-Modul war ein "Infineon 512MB-SDRAM-Modul HYS64V64220GU-7-D".

# Anhang E

## Layout

### E.1 Bauteilseite

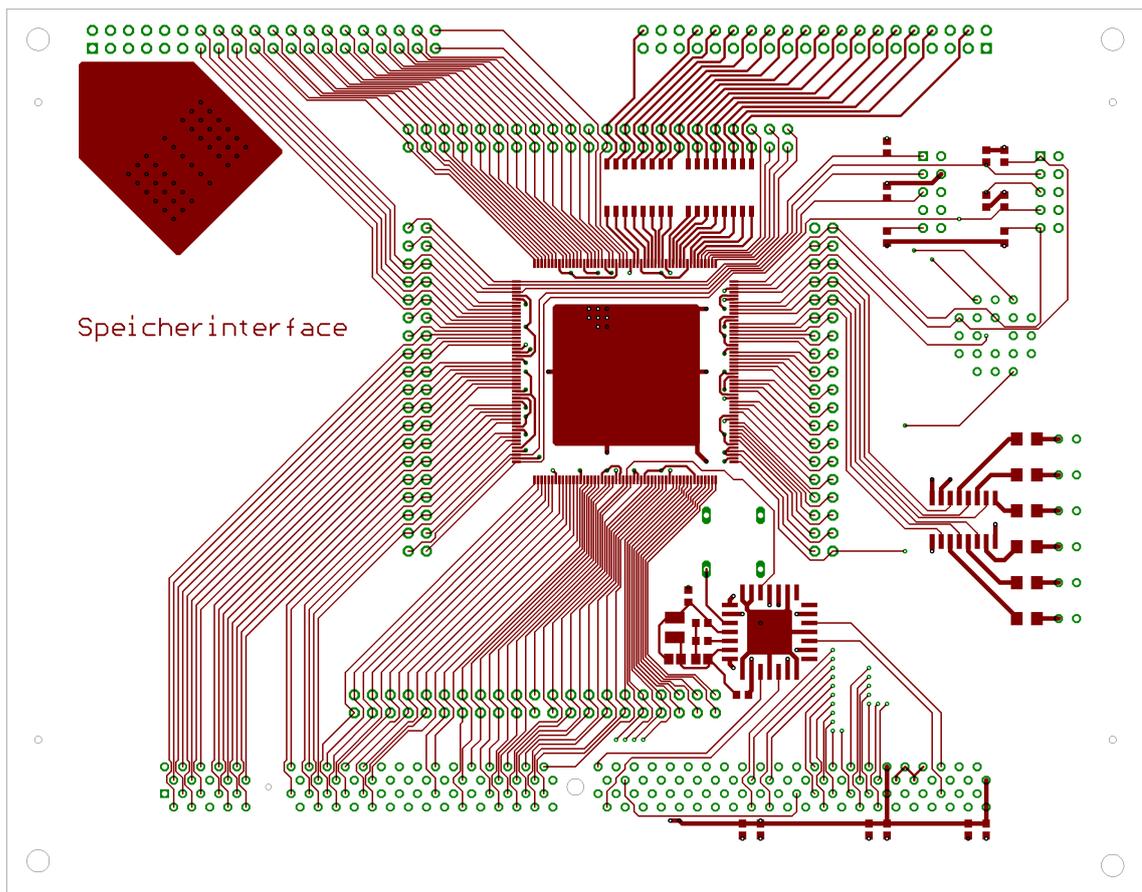


Abbildung E.1: Layout der Bauteilseite

## E.2 Lötseite

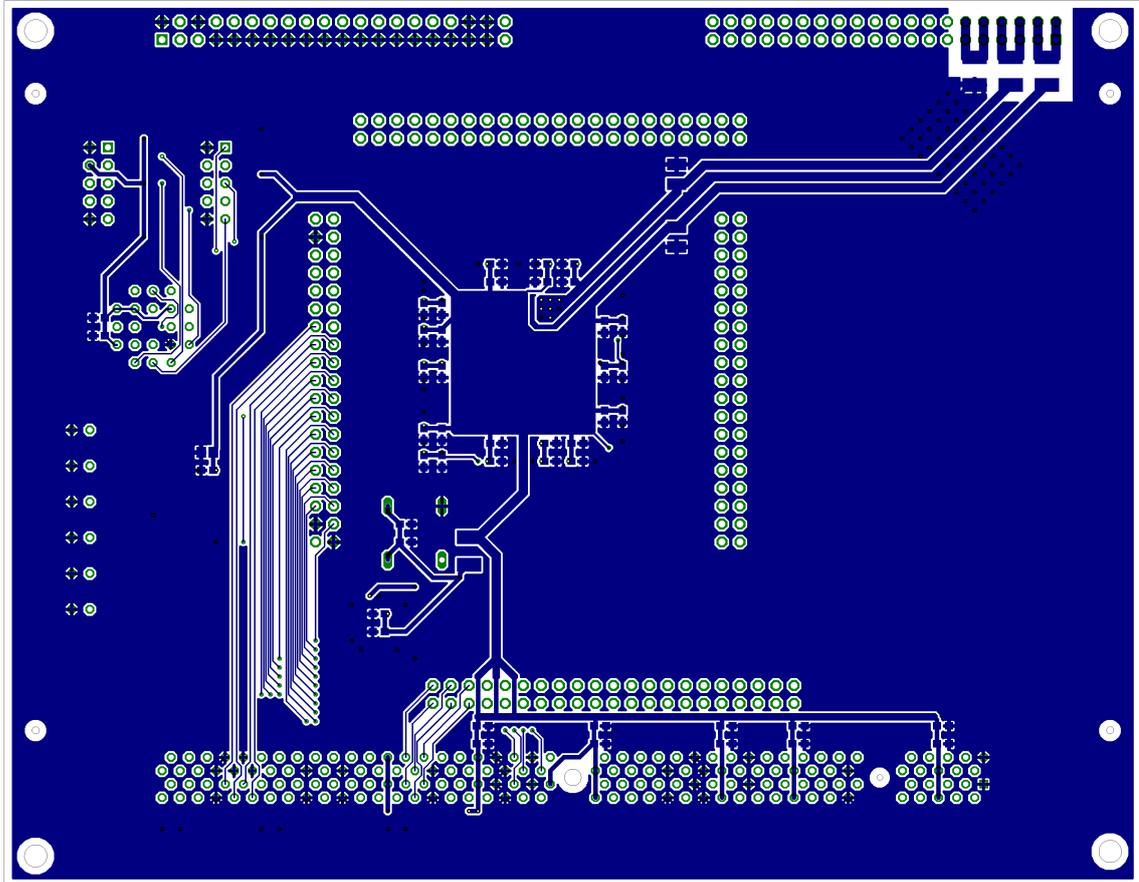


Abbildung E.2: Layout der Lötseite

# Anhang F

## Bestückungsplan

### F.1 Bauteilseite

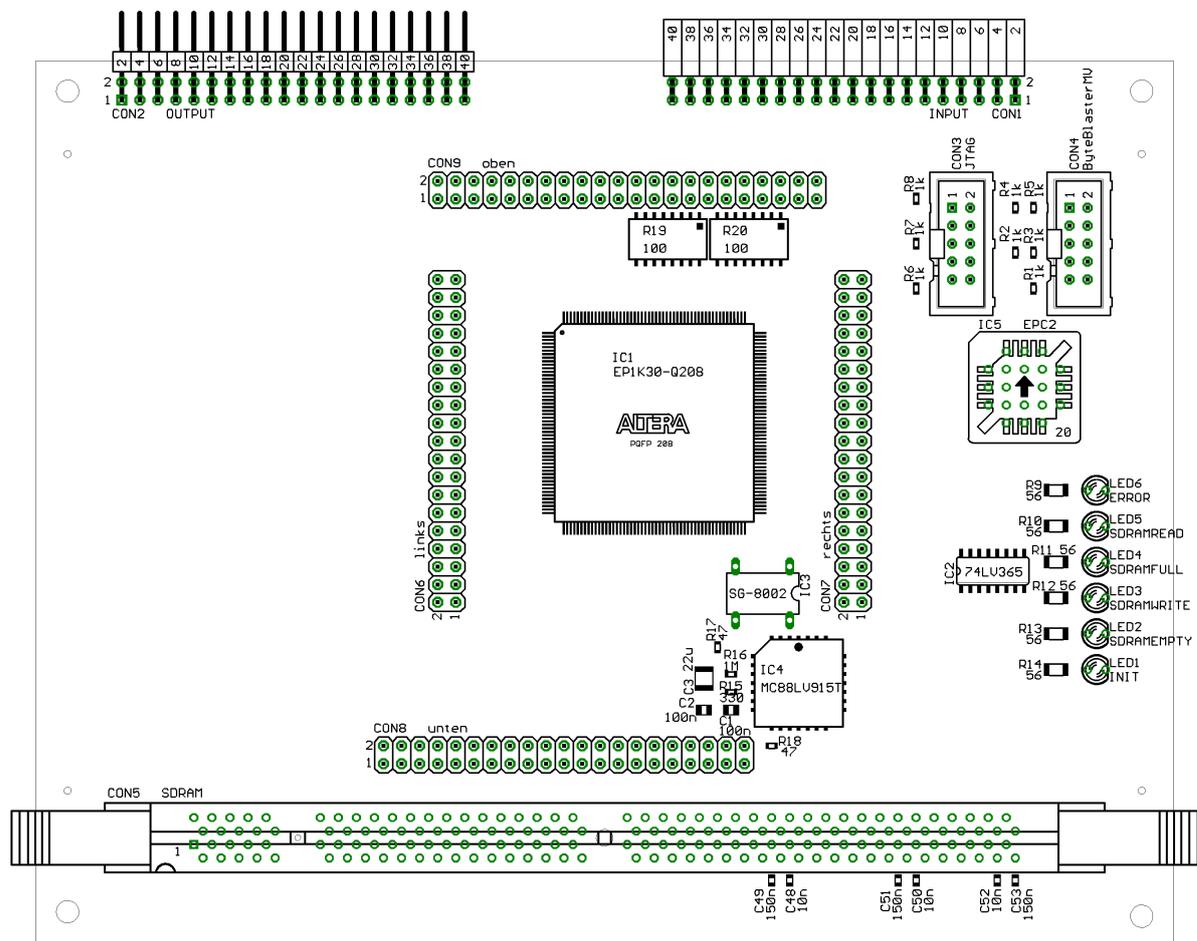


Abbildung F.1: Bestückungsplan der Bauteilseite

## F.2 Lötseite

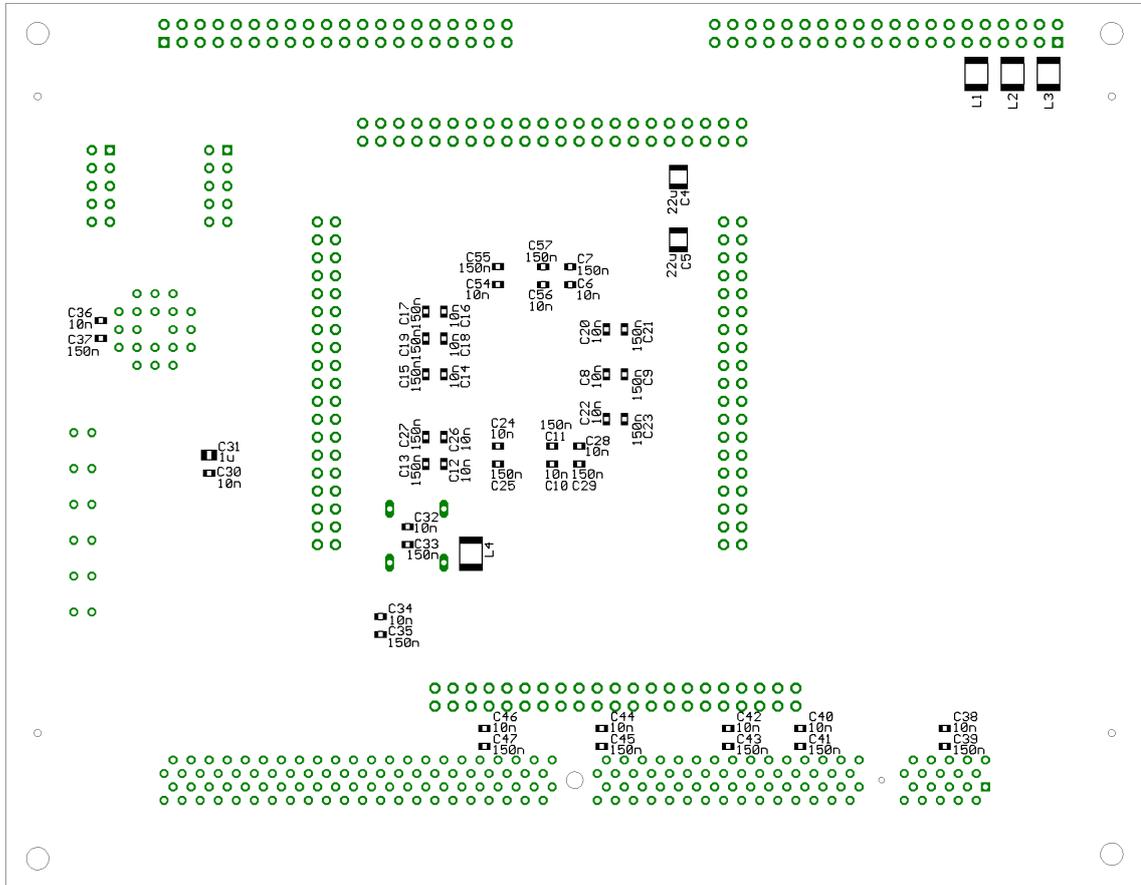


Abbildung F.2: Bestückungsplan der Lötseite

# Literaturverzeichnis

- [1] Analog Devices Datasheet: 14-Bit 80 MSPS A/D Converter AD6645  
Analog Devices 2002
- [2] Analog Devices Preliminary Technical Data: High-Speed ADC FIFO Evaluation Kit HSC-ADC-EVAL-SC/DC,  
Analog Devices 2002
- [3] Betty Prince: High Performance Memories: New architecture DRAMs and SRAMs – evolution and function Revised Edition  
Chichester/England: John Wiley & Sons Ltd. 1999  
ISBN 0-471-986100
- [4] Christian Ellwein: Programmierbare Logik mit GAL und CPLD: Einführung in die Schaltungsentwicklung mit Logikbausteinen in ISP-Technologie  
München,Wien:Oldenburg 1999  
ISBN 3-486-24610-0
- [5] Adolf Auer/Dieter J. Rudolf: FPGA Feldprogrammierbare Gate Arrays  
Heidelberg:Hüthig 1995  
ISBN 3-7785-2359-7
- [6] Altera Component Selector Guide  
Altera Corporation March 2002
- [7] Altera ACEX 1K Data Sheet, ver 3.3  
Altera Corporation September 2001
- [8] Altera ACEX 1K Data Sheet, ver 3.4  
Altera Corporation May 2003
- [9] Altera Design Software Selector Guide  
Altera Corporation July 2002
- [10] PCI-7200/cPCI-7200 - 12MB/S High Speed Digital Input/Output Card - User's Guide, Manual Rev. 2.20: October 14, 2000  
ADLINK Technology Inc. 1999-2000
- [11] cPCI-7300A & PCI-7300A - 80MB Ultra-High Speed 32-CH Digital I/O Boards - User's Guide, Manual Rev. 2.21: May 9, 2001  
ADLINK Technology Inc. 1999-2000
- [12] Peter Heusinger/Karlheinz Ronge/Gerhard Stock: PLDs und FPGAs in der Praxis: komplexe Logikbausteine erfolgreich programmieren  
Poing:Franzis 1994  
ISBN 3-7723-6075-0
- [13] Andreas Bleck/Michael Goedecke/Sorin A. Huss/Klaus Waldschmidt: Praktikum des modernen VLSI-Entwurfs: eine Einführung in die Entwurfsprinzipien und –beschreibungen, unter besonderer Berücksichtigung von VHDL; mit einer umfangreichen Anleitung zum Praktikum

- Stuttgart:Teubner 1996  
ISBN 3-519-02296-6
- [14] Gunther Lehmann/Bernhard Wunder/Manfred Selz: Schaltungsdesign mit VHDL: Synthese, Simulation und Dokumentation digitaler Schaltungen  
Poing:Franzis 1994  
ISBN 3-7723-6163-3
- [15] David R. Coelho: The VHDL handbook  
Norwell/Massachusetts 02061:Kluwer Academic Publishers 1989  
ISBN 0-7923-9031-8
- [16] Joel M. Schoen: Performance and fault modeling with VHDL  
Englewood Cliffs/New Jersey 07632: Prentice-Hall 1992  
ISBN 0-13-658816-6
- [17] Peter Rössler: Komplexe Schaltwerke Rechenübung – Beispielsammlung mit Lösungen  
Wien: TU Wien-E384 Institut für Computertechnik 2001
- [18] Peter Rössler: Komplexe Schaltwerke Rechenübung – Skriptum für den Laborteil  
Wien: TU Wien-E384 Institut für Computertechnik 2001
- [19] Dietmar Loy: VHDL Introduction  
Wien:TU Wien-E384 Institut für Computertechnik 1995
- [20] Andreas Mäder: VHDL Kompakt  
Hamburg: Universität Hamburg-Fachbereich Informatik  
<http://tech-www.informatik.uni-hamburg.de/vhdl/doc/kurzanleitung/vhdl.pdf>
- [21] Peter J. Ashenden: The VHDL Cookbook First Edition  
Adelaide:University of Adelaide-Dept. Computer Science 1990  
<ftp://ftp.cs.adelaide.edu.au/pub/VHDL-Cookbook/VHDL-Cookbook.pdf>
- [22] Altera Single & Dual-Clock FIFO Megafunctions User Guide, ver 1.0  
Altera Corporation June 2003
- [23] Samsung CMOS SDRAM Device Operations & Timing Diagram  
Samsung Corporation 2002
- [24] hynix SDRAM Device Operation  
Hynix Semiconductor 2002
- [25] Infineon 3.3V 64Mx64/72-Bit, 512MByte SDRAM Modules, 168-pin Unbuffered DIMM Modules HYS64/72V64220GU, Data Sheet ver 9.01  
Infineon Technologies 2001
- [26] Altera Application Note 204: Using ModelSim-Altera in a Quartus II Design Flow, ver 1.2  
Altera Corporation December 2002
- [27] Eagle 4.0 Trainings-Handbuch 2.Auflage  
CadSoft Computer GmbH 2001
- [28] Eagle 4.0 Referenz-Handbuch 3.Auflage  
CadSoft Computer GmbH 2001
- [29] Altera ByteBlasterMV Parallel Port Download Cable, ver 3.3  
Altera Corporation July 2002
- [30] Altera Configuration Handbook  
Altera Corporation 2003
- [31] Altera Application Note 116: Configuring SRAM-Based LUT Devices, ver 3.0  
Altera Corporation February 2002

- [32] Altera Data Sheet: Configuration Devices for SRAM-Based LUT Devices, ver 12.2  
Altera Corporation December 2002
- [33] Altera Data Sheet: Enhanced Configuration Devices, ver 2.0  
Altera Corporation April 2002
- [34] Motorola Technical Data: Low Voltage Low Skew CMOS PLL Clock Driver, 3-State, Rev 3  
Motorola Inc. 2001
- [35] Altera Application Note 315: Altera Guidelines for Designing High-Speed FPGA PCBs, ver 1.0  
Altera Corporation July 2003
- [36] Altera Application Note 75: Altera High-Speed Board Designs, ver 4.0  
Altera Corporation November 2001
- [37] Altera Application Note 224: Altera High-Speed Board Layout Guidelines , ver 1.1  
Altera Corporation September 2003
- [38] Newsgroup: comp.arch.fpga
- [39] Newsgroup: comp.lang.vhdl