RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

Joachim Philipp Becker

Ein FPGA-basiertes Testsystem für gemischt analog/digitale ASICs

Diplomarbeit

HD-KIP-01-11

KIRCHHOFF-INSTITUT FÜR PHYSIK

Fakultät für Physik und Astronomie

Ruprecht-Karls-Universität Heidelberg

Diplomarbeit

im Studiengang Physik

vorgelegt von

Joachim Philipp Becker

aus Hanau/Main

2001

Ein FPGA-basiertes Testsystem

für gemischt analog/digitale ASICs.

Die Diplomarbeit wurde von Joachim Philipp Becker ausgeführt

am Kirchhoff Institut für Physik

unter der Betreuung von

Herrn Prof. Dr. Karlheinz Meier

Ein FPGA-basiertes Testsystem für gemischt analog/digitale ASICs:

Der Test rekonfigurierbarer gemischt analog/digitaler ASICs erfordert neben der digitalen Konfiguration auch eine Evaluierung der analogen Fähigkeiten des Chips. Aufgabe eines Testsystems ist die Aufnahme des ASICs und Bereitstellung von Versorgungsspannungen, digitaler Kommunikation und analogen Testmustern. Außerdem müssen die analogen Ausgaben des ASICs gemessen und verarbeitet werden. Zur Ausnutzung der Geschwindigkeit des ASICs ist es notwendig, auch komplexe Testabfolgen wie genetische Algorithmen vollständig in Hardware zu implementieren, wozu eine ausreichend große programmierbare Logik bereitgestellt werden muss.

Die vorliegende Arbeit zeigt eine Spezifizierung der Anforderungen, die Entwicklung einer entsprechenden Schaltung und die Implementierung als Leiterplatte. Zur schnellen Rechneranbindung ist die Platine als PCI-Karte ausgeführt und beinhaltet als Kernstück einen FPGA. Im Rahmen dieser Arbeit wurde das Testsystem in Betrieb genommen und Teile der Programmierung erstellt.

An FPGA-based testbench for mixed-signal ASICs:

The test of reconfigurable mixed-signal ASICs demands digital configuration in addition to the evaluation of the analogue abilities of a chip. The task of the testsystem is to affiliate the ASIC and provide power, digital communication and analogue test patterns. Besides that, analogue outputs of the ASIC have to be measured and processed. In order to exploit the speed of the ASIC, it is necessary to completely implement complex testsequences like evolutionary algorithms in the hardware, for which a sufficiently large programmable logic has to be provided.

The present thesis describes the specifications demanded, the development of an appropriate circuit and implementation as printed circuit board. For fast communication with a computer, the hardware is designed as an PCI-board and contains an FPGA as its central unit. As a result of this work the board was commissioned and parts of the software were implemented.

Inhaltsverzeichnis

Kapitel 1

Motivation

Zur Bewältigung von Aufgaben der digitalen Signalverarbeitung geht die Lösungsstrategie in den meisten Fällen von einer möglichst allgemeinen, programmierbaren Hardware aus und reduziert das Problem auf die Definition und Optimierung eines geeigneten Lösungsalgorithmus in Software. Durch die immer weiter steigende Leistungsfähigkeit moderner Computer und die gute Beherrschbarkeit der digitalen Methoden ist auch die analoge Signalverarbeitung oft darauf reduziert, das analoge Signal möglichst bald zu digitalisieren, um es dann mit den Mitteln der digitalen Signalverarbeitung zu verarbeiten und gegebenenfalls wieder in ein analoges Ausgangssignal umzuwandeln. Dies gilt auch für die Simulation analoger Schaltkreise, wo auf die allgemeinen numerischen Methoden zurück gegriffen wird, um das Verhalten einer Schaltung zu berechnen.

Bei manchen Problemen verspricht man sich jedoch Geschwindigkeitsgewinne, falls es gelingt, möglichst viele Bestandteile des Lösungsansatzes direkt in Hardware zu implementieren. Dabei soll ein Lösungsalgorithmus nicht in Software geschrieben und dann auf einer allgemeinen Recheneinheit ausgeführt werden, sondern eine auf das Problem spezialisierte Hardware zum Einsatz kommen. Im digitalen Bereich gibt es einige Beispiele für dieses Vorgehen, wie mathematische Co-Prozessoren, Vektor-Recheneinheiten moderner Prozessoren oder spezialisierte Grafik-Chips für 3D-Effekte. Weil analoge Signale oft der digitalen Weiterverarbeitung zugeführt werden, gibt es aber kaum spezialisierte Hardware, die komplizierte analoge Signalverarbeitung leistet. Die Entwicklung analoger Bausteine für spezielle Anwendungen ist Objekt aktueller Forschung. Dabei erfolgt die Verwirklichung der Schaltung als Mikrochip, indem ASIC[1]-Prototypen hergestellt werden.

Die Arbeitsgruppe *electronic vision(s)* am Kirchhoff Institut für Physik entwickelt konfigurierbare gemischt analog/digitale ASICs für Anwendungen in der Bildverarbeitung. Es sind dies beispielsweise Kamerachips [DIVICHI], Chips zur Kantendetektion auf Bildern [EDDA], analoge neuronale Netzwerke [EVOOPT], ANNA oder programmierbare Transistor-Felder [FPTA].

[1] application specific integrated circuit

Der FPTA[2]-Chip soll zur künstlichen Evolution analoger Schaltungen benutzt werden. Kernstück ist ein Feld aus 256 Transistoren, die zu einer quadratischen Matrix zusammengefügt sind. Transistoren bilden die Grundeinheit analoger Schaltungen, da aus ihnen alle anderen Bauteile wie Kondensatoren, Widerstände, Operationsverstärker, etc. aufgebaut werden können. Die Kennlinie jeder Transistorzelle im FPTA ist konfigurierbar, und durch programmierbare Vernetzung der Zellen untereinander können Schaltkreise implementiert werden.

Die Suche nach neuen Schaltungen oder das Trainieren eines neuronalen Netzwerkes durch künstliche Evolution beruhen auf dem Prinzip von Selektion und Mutation. Dabei wird ein Problem definiert, das eine Schaltung bewältigen soll, indem zu einer Anzahl von Eingabevektoren die gewünschten Ausgaben des ASICs angegeben werden. Das Testsystem kann eine Konfiguration in den ASIC laden, alle Eingabevektoren anlegen und die Reaktion des ASICs mit dem vorgegebenen Ausgabewert vergleichen. Durch ein Bewertungsschema erhält man ein Maß für die Funktionsfähigkeit einer Konfiguration (Fitness) und kann beliebige Konfigurationen testen, bis ein befriedigendes Ergebnis erzielt wird. Da die Menge der sinnvollen Konfigurationen gegenüber dem gesamten Konfigurationsraum sehr klein ist, scheint eine zufällige Suche aussichtslos. Daher geht ein genetischer Algorithmus von einer Population von Individuen aus, also einer Menge von zufällig gewählten Konfigurationen, bewertet deren Funktionalität und nimmt dann gezielte Veränderungen vor. Individuen, die eine schlechte Fitness aufweisen, werden verworfen (Selektion), während die Population aufgefüllt wird mit Konfigurationen, die von den besser bewerteten Individuen abgeleitet sind. Mechanismen dafür sind die zufällige Veränderung (Mutation) und die Vermischung (Crossover) von Konfigurationen. So wird eine neue Generation gewonnen, die wiederum getestet wird, und auf diese Weise wird der Algorithmus über viele Generationen hinweg ausgeführt, um eine Verbesserung der Funktionalität bis hin zu einer Konfiguration zu erreichen, die das Problem löst.

Um die erste Inbetriebnahme eines konfigurierbaren gemischt analog/digitalen ASICs und später den Ablauf eines genetischen Algorithmus zu ermöglichen, muss ein Testsystem die folgenden Aufgaben bewältigen:

Spannungsversorgung des Chips: Die Bereitstellung der Versorgungsspannung des zu testenden ASICs ist grundlegend. Diese sollte gegen Überlast gesichert sein und einer Zerstörung des ASICs vorbeugen. Es sollte aber auch die Möglichkeit bestehen, mehrere regelbare Gleichspannungen bereitzustellen, die für getrennte Versorgung von IO[3]-Treibern, Spannungsreferenzen, o.ä. benutzt werden können.

Digitale Kommunikation: Ein ASIC wird über digitale Leitungen angesprochen, wobei ein digitaler Bus mit einer Breite von mehreren Byte ohne Probleme realisierbar sein sollte. Da es verschiedene Standards von Signallogiken gibt, soll es möglich sein, die Kommunikation an die unterschiedlichen Anforderungen anzupassen. Die hohen Geschwindigkeiten von bis zu 400 MHz implizieren die Vorab-Programmierung der digitalen Kommunikation und das Ablaufen dieser Programme auf einer schnellen Mikroelektronik.

[2] field programmable transistor array
[3] input/output

Analoge Stimuli: Um den Analogteil eines ASICs zu testen, müssen schnelle analoge Signale erzeugt werden, die jede beliebige Amplitude im zulässigen Wertebereich annehmen können. Eine freie Programmierung verschiedener Signalformen und Testmuster sollte leicht umzusetzen sein.

Analoge Messwerte aufnehmen: Zur Überprüfung des Analogverhaltens zählt auch die Aufnahme von analogen Spannungsmesswerten, die der ASIC unter Umständen in Abhängigkeit von analogen oder digitalen Eingaben ausgibt. Diese Messwerte müssen parallel zur Abarbeitung der digitalen Kommunikation und Erzeugung der analogen Stimuli aufgezeichnet werden und nach dem programmierten Test für eine Auswertung zur Verfügung stehen.

Rückkopplung: Für den Betrieb konfigurierbarer Hardware wie neuronalen Netzwerken oder programmierbaren Transistorfeldern ist es wesentlich, dass aktuelle Messungen Einfluss auf die Konfiguration bzw. Stimuli haben können und somit eine Rückkopplung implementiert werden kann. Dies kann im einfachsten Falle bedeuten, dass Eingaben für den Chip sich ändern, wenn ein bestimmtes analoges Verhalten festgestellt wurde bis hin zum komplizierten Fall, dass nach Auswertung bestimmter Messwerte eine komplette Neukonfiguration des Chips durchgeführt werden soll.

In der vorliegenden Arbeit werden die Entwicklung und Implementierung eines solchen Testsystems beschrieben, das es ermöglicht, die genannten Aufgaben zu erfüllen und komplexe digitale und analoge Kommunikation auf Chip-Ebene mit einer flexiblen Programmierung und komfortabler Benutzerschnittstelle zu vereinigen. Die grundlegenden Strukturen des Testaufbaus wurden bereits in [Bli00] für den Test des EDDA-Chips benutzt und haben bewiesen, dass sie den Anforderungen gerecht werden. Es gilt nun, das Konzept zu verallgemeinern und so eine Plattform für unterschiedlichste Testaufbauten bereit zu stellen. Außerdem wird den Neuentwicklungen auf dem Gebiet der elektronische Bauteile Rechnung getragen und modernste Technologien verwendet, um das bestehende Konzept umzusetzen.

Kapitel 2

Übersicht über das Gesamtsystem

Kernstück des Testsystems ist eine PCI[1]-Karte, die die Schnittstelle zwischen einer PC-Software und dem zu testenden ASIC darstellt. Die Ansteuerung des Chips wird beschrieben durch Timing-Diagramme, die meist auf einer synchronen digitalen Kommunikation aufbauen. Das heißt, ein Systemtakt stellt die Zeitbasis dar, wobei jeweils zu einer steigenden Flanke des Signals ein neuer Zustand auf der Datenleitung Gültigkeit erlangt. Da diese Kommunikation mit Taktfrequenzen im Bereich von 20 bis 100 MHz (LVTTL[2]) oder sogar bis zu 400 MHz (LVDS[3]) betrieben wird, wird ein Experiment nicht mehr voll durch direkte Interaktion eines Menschen, sondern durch eine programmierbare Hardware gesteuert. Ein wesentliches Merkmal dieses Verfahrens ist, dass die Programmabläufe mit der Geschwindigkeit der Hardware ausgeführt und Reaktionen in Echtzeit ($10^{-8}\,s$) ausgewertet werden, während die Programmierung des Verhaltens und Eingriffe in das Experiment auf einer wesentlich größeren Zeitskala erfolgen (Tage).

Dies wird verwirklicht durch die Verwendung eines FPGAs[4], der auf der einen Seite als Hardwareimplementation einer Gatterlogik die Kommunikation auf Chip-Ebene bewerkstelligen kann, auf der anderen Seite aber seine Konfiguration aus einer anspruchsvollen Entwicklungsumgebung erhält. Der Programmablauf wird in einer Hochsprache[5] programmiert, compiliert und auf die Hardware des FPGAs abgebildet. Im Entwicklungsstadium ist es außerdem möglich, Simulationsmodelle der zu testenden Chips zu erstellen und die Kommunikation zwischen FPGA und ASIC zu simulieren. Mit den heute verfügbaren Mitteln können die Anbindung an den PCI-Bus, Ansteuerung von Analog/Digitalwandlern, Verwaltung von Speicherbausteinen und komplizierte Testabläufe wie z.B. genetische Algorithmen programmiert, simuliert und auf einem FPGA dargestellt werden. Wegen der Rekonfigurierbarkeit des FPGAs können mit ein und derselben Hardware durch reine Programmentwicklung die unterschiedlichsten Experimente durchgeführt werden. Mit der Einbindung der Hardware in einen PC können die diversen Testaufbauten auch auf die Ressourcen

[1] peripheral component interconnect
[2] low voltage transistor-transistor logic
[3] low voltage differential signaling
[4] field programmable gate array
[5] Hardwarebeschreibungssprache

des Rechners zugreifen. Dies ermöglicht die Beeinflussung des laufenden Experimentes über Eingabemasken auf dem Bildschirm, direkte Visualisierung von Messungen und Statusanzeigen, Speicherung von Experimenten auf Festplatte, Vereinigung von mehreren Testsystemen in einem Computer oder sogar Zugriff auf die Hardware über ein Netzwerk.

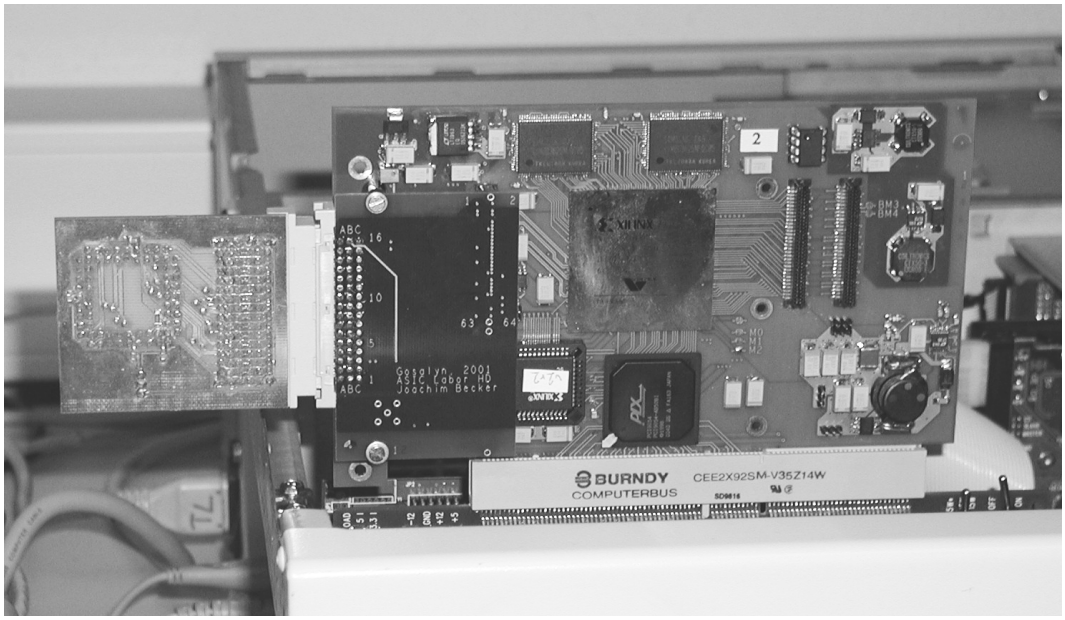


Abbildung 2.1: Aufbau eines Experiments

Abbildung 2.1 zeigt ein Foto eines aufgebauten Experiments. Die PCI-Karte mit dem Projektnamen *Darkwing* ist mit dem Computer (PC) verbunden, wobei sie hier auf einem PCI-Bus Extender aufgesteckt ist, um Messungen vorzunehmen und normalerweise in den geschlossenen Computer eingebaut ist. Auf der linken Seite der Karte befindet sich eine Adapterplatine (hier *Gosalyn*) die mit der Gehäuserückwand bündig abschließt und Anschlüsse für den externen Träger jedes individuellen ASICs bereitstellt. Hier zu sehen ist die Unterseite der Trägerplatine für einen EVOOPT-Chip. Im folgenden sollen ein tieferer Einblick in die Funktionsweise des Gesamtsystems sowie eine vollständige Erläuterung aller notwendigen Bestandteile gegeben werden.

2.1 Hardware

Eine schematische Sicht der Hauptkomponenten und Datenpfade zeigt Abbildung 2.2. Die Kommunikation zum PC findet auf dem PCI-Bus statt, an den viele verschiedene Karten in einem PC angeschlossen sind. Ein PCI-Masterchip vom Typ PCI9054 der Firma PLX übernimmt diese Kommunikation und stellt dann ausschließlich die Daten auf einem lokalen Bus auf Darkwing zur Verfügung, die an diese spezielle

Karte adressiert wurden [PLX9054]. Ein serielles EEPROM[6] dient zur dauerhaften Speicherung der Konfigurationsdaten, die bei jedem Systemstart ausgelesen werden.

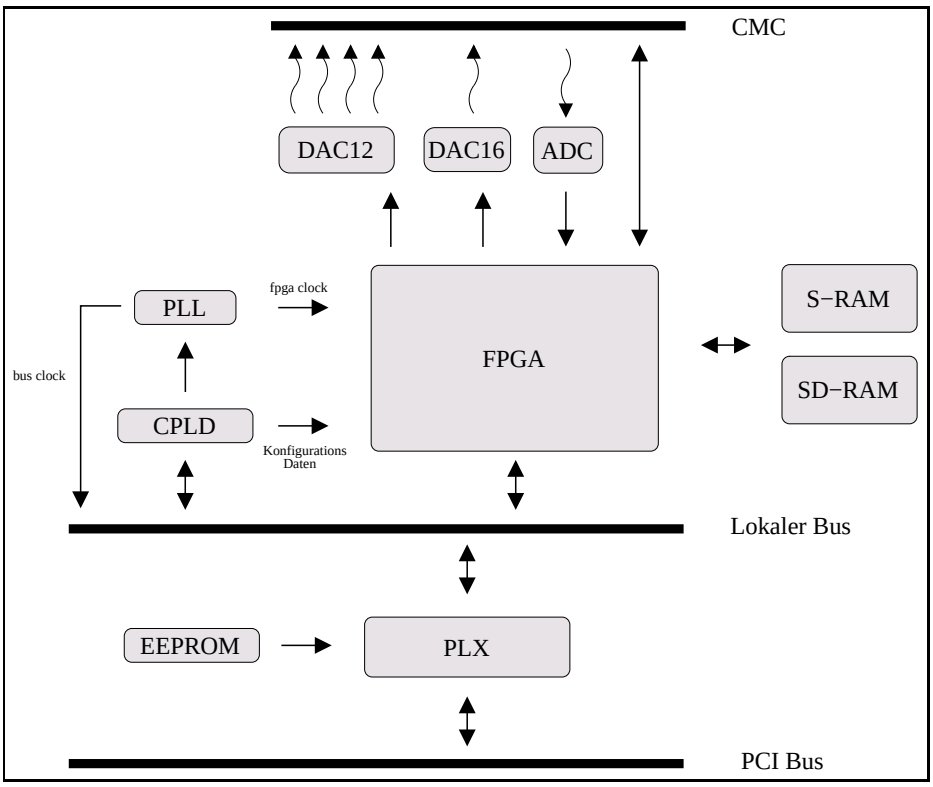


Abbildung 2.2: Blockdiagramm der PCI-Karte

Auch der FPGA ist bei einem Systemstart nicht konfiguriert, und könnte eine Programmierung aus einem ROM lesen. Da es aber möglich sein soll, je nach Anwendung unterschiedliche Konfigurationen zu laden und den FPGA auch während des Betriebs zu rekonfigurieren, wird eine Programmierung über den PCI-Bus ermöglicht. Dabei empfängt der FPGA die Daten nicht selber, sondern ein fest programmierter Logikbaustein (CPLD[7]) setzt die Daten vom lokalen Bus auf das Initialisierungsprotokoll des FPGAs um. Eine weitere Aufgabe des CPLDs besteht in der Programmierung der PLL[8], die zwei Taktsignale erzeugt. Dies sind die Zeitbasis für den lokalen Bus und eine zusätzliche Taktung für den FPGA.

Ist der FPGA initialisiert, so übernimmt er die zentrale Rolle auf der Karte. Er kommuniziert über den lokalen Bus und den PLX-Chip mit dem PC und kann alternativ zwei fest eingebaute S-RAM[9] Bausteine oder ein SD-RAM[10] Modul als lokalen Speicher ansteuern. Außerdem bedient er die analogen Bauteile. Dies sind

[6] electrically erasable programmable read only memory
[7] complex programmable logic device
[8] phase locked loop
[9] static random access memory
[10] synchronous dynamic random access memory

im Einzelnen vier DACs[11] mit einer Auflösung von 12 Bit zur Bereitstellung von Gleichspannungen und je ein DAC mit 16 Bit und ein ADC[12] mit 12 Bit Auflösung. Diese Bausteine werden benutzt, um das Analogverhalten des angeschlossenen ASICs (auch DUT[13]) zu testen. Weiterhin stellt der FPGA digitale Leitungen zum ASIC bereit. Alle Verbindungen nach außen liegen auf Steckern, damit jeder ASIC mit einer individuellen Tochterplatine auf die PCI-Karte aufgesteckt werden kann. Diese Tochterplatine kann kompatibel zum CMC[14]-Standard sein, und daher sind die Kontakte als CMC-Stecker ausgeführt.

2.2 Software

Der Begriff *Software* wird in diesem Zusammenhang unterteilt in Hardware beschreibende Programme, die die Konfiguration des FPGAs darstellen und auf der Karte ablaufen, und PC-Software im eigentlichen Sinne, die als Benutzeroberfläche des Experimentes erscheint und auf einem PC-Betriebssystem aufsetzt, das die benutzte Hardware ansteuert.

2.2.1 Hardwarebeschreibung

Um eine FPGA-Konfiguration zu erstellen, beginnt man mit der Entwicklung eines Quellcodes in einer Hardware beschreibenden Sprache (HDL[15]), wie zum Beispiel VHDL[16]. Diese Sprachen unterscheiden sich im Aufbau wesentlich von anderen Hochsprachen durch hierarchische Strukturen und Variablentypen, die Logik auf Bauteilebene repräsentieren können, womit der späteren Instantiierung des Codes in Hardware Rechnung getragen wird. Es kann die Verhaltensbeschreibung eines Bausteins programmiert werden, die durch Synthese auf Operatoren boolscher Algebra abgebildet wird [VHDL]. Die Hochsprache wird also in Gatterlogik übersetzt und an die Technologie des jeweiligen FPGAs angepasst. Daraus wird ein Bitstrom erstellt, der eine Konfiguration des FPGAs ermöglicht.

Ein weiteres Merkmal von HDLs ist die Parallelität, das heißt, dass alle Anweisungen in einem Block gleichzeitig ausgeführt werden. Eine Serialisierung muss manuell durch endliche Automaten, so genannte *state machines*, erzwungen werden, die synchron zum Taktsignal ihre Zustände ändern. Durch Verzweigungen in den entsprechenden Flussdiagrammen können die Timing-Diagramme, die zur Kommunikation mit anderen Bauteilen eingehalten werden müssen, abgebildet werden.

Mehrere solche Automaten laufen unter Umständen unabhängig voneinander gleichzeitig auf einem FPGA ab und übernehmen Teilaufgaben, die zusammengesetzt die gesamte Funktionalität des Chips ergeben. Für das vorliegende Testsystem gibt es einen umfangreichen Quellcode, der in die folgenden Bereiche unterteilt ist:

[11] digital analog converter
[12] analog digital converter
[13] device under test
[14] common mezzanine card
[15] hardware description language
[16] very high speed integrated circuit - hardware description language

- Interne Register im FPGA
- Kommunikation mit dem lokalen Bus
- Ansteuerung des RAMs
- Ansteuerung der analogen Komponenten
- Kommunikation mit dem zu testenden Chip

Dementsprechend ist auch die feste Programmierung des CPLDs in einer HDL verfasst und wird als Bitkonfiguration in den nicht flüchtigen Speicherzellen des Bausteins hinterlegt. Der CPLD beinhaltet eine *state machine*, die die Konfiguration des FPGAs und der PLL leistet.

2.2.2 PC-Software

Der Datenaustausch zwischen PC und FPGA findet in Form von Registerzugriffen statt. Im HDL-Code sind Register definiert, die Daten halten und durch eine entsprechende *state machine* über den lokalen Bus adressierbar sind, also beschrieben und ausgelesen werden können. Darin werden zum Beispiel Werte für die DACs und den ADC zwischengespeichert, und es gibt die Möglichkeit, über Registerzugriffe die *state machines* im FPGA zu beeinflussen und sie gezielte Kommandos ausführen zu lassen. Einer PCI-Karte ist ein gewisser Adressraum auf dem PCI-Bus zugeordnet, auf den durch einfache Lese- und Schreiboperationen zugegriffen werden kann, die durch einen Betriebssystem-Treiber [Bau01, Jungo] zur Verfügung gestellt werden. Diese Zugriffe werden über den PLX-Chip auf den lokalen Bus abgebildet, so dass eine PC-Software durch das Schreiben von Werten und Adressen auf dem PCI-Bus Zugriff auf die Register des FPGA hat.

Die PC-Software ist in einer beliebigen Hochsprache verfasst, und im vorliegenden Fall wird C++ benutzt, um Datenaustausch mit der Hardware und eine Benutzeroberfläche auf Windows oder Linux bereitzustellen. Eine Testsoftware besteht aus Eingabemasken für die spezifische Chip-Konfiguration, Bearbeitung von analogen Stimuli und Anzeige und Auswertung der analogen Messwerte. Experimente können auf Festplatte gespeichert werden und jederzeit wieder reproduziert werden. Weiterhin besteht die Möglichkeit, anspruchsvolle Rückkopplungen von Messungen auf Konfigurationen in Software zu schreiben, viele Konfigurationen im Speicher vorzuhalten und miteinander zu vergleichen. Somit ist dieses System die ideale Plattform für die Implementierung eines genetischen Algorithmus und die Anwendung künstlicher Evolution zur Gewinnung interessanter Konfigurationen.

Kapitel 3

Hardware

In der vorliegenden Arbeit liegt der Schwerpunkt auf der Implementierung der Hardware. Es ist eine Auswahl der Komponenten zu treffen und eine vollständige, funktionierende Schaltung zu entwerfen. Die funktionalen Elemente, die in der Übersicht Abbildung 2.2 genannt sind, bedingen durch ihre elektrischen Eigenschaften die Verwendung von Pegelkonvertern, Operationsverstärkern und passiven Bauelementen, die basierend auf den Datenblättern der entsprechenden Bauteile richtig dimensioniert werden müssen. Das folgende Kapitel wird auf diese Details der Hardware näher eingehen.

3.1 Mechanische Spezifikationen der Karte

Die Realisierung des Testsystems auf einer PCI-Karte bringt nicht nur die Vorteile der schnellen Rechneranbindung, sondern auch die der Bereitstellung von Versorgungsspannungen, Einbettung in ein Gehäuse und Belüftung. Allerdings erfordert sie auch die Einhaltung der Normen, wie sie in [PCI] definiert sind.

3.1.1 Format der Platine

Abbildung 3.1 zeigt die mechanischen Abmessungen der Platine, die in einen Steckplatz auf der PC-Hauptplatine eingesteckt und mittels eines Kartenhalters am Gehäuse des PCs festgeschraubt wird. Die Dicke der Platine beträgt $1.77 \pm 0.1\,\mathrm{mm}$, und die erlaubten Bauhöhen sind $14.48\,\mathrm{mm}$ auf der gezeigten Bestückungsseite und $2.67\,\mathrm{mm}$ auf der Rückseite. Dies ermöglicht das Aufbringen einer Tochterplatine nach dem CMC-Standard auf der Oberseite, wie in 3.1.2 beschrieben.

3.1.2 Anordnung der Bauteile

Die Platzierung der Bauteile auf der Karte ist von großer Bedeutung, da durch eine geschickte Anordnung Leiterbahnlängen minimiert und die Signalqualitäten maxi-

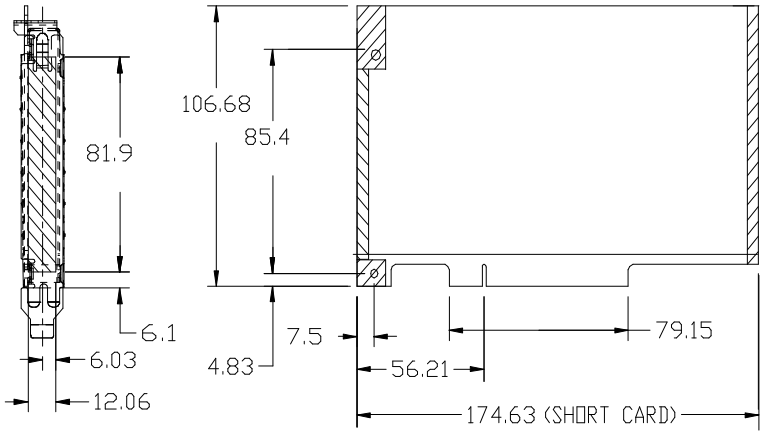


Abbildung 3.1: Abmessungen nach PCI-Norm [PCI]

miert werden können. Weiterhin ist es für die Qualität der Signale von Vorteil, wenn die analogen Baugruppen räumlich getrennt von potentiellen Störungsquellen sind.

Fest definiert sind die Platzierung und Belegung der Kontaktleiste an der Unterkante der Karte. Die Anschlussbelegung des PLX-Chips sieht eine ausgezeichnete Richtung vor, die es erlaubt, die PCI-Signale kreuzungsfrei an den Stecker zu führen, und durch eine Zentrierung des Chips über der Kontaktleiste ist die erlaubte Leitungslänge der Bussignale von 37 mm[1] gut einzuhalten. Der lokale Bus führt von der Oberkante des PLX-Chips direkt zum FPGA, der im Zentrum der Karte liegt. Dies illustriert Abbildung 3.2

Entsprechend des Blockdiagramms (Abbildung 2.2) muss der CPLD ebenfalls Zugriff auf einen Teil der lokalen Busleitungen haben und ist deshalb links in der Nähe des Busses angesiedelt. Zusammen mit der Orientierung des FPGAs wird erreicht, dass der FPGA den lokalen Bustakt auf einem globalen Takteingang[2] empfangen kann und die vordefinierten Anschlüsse für die Konfigurationsleitungen[3] an der linken Seite des FPGAs in Nähe zum CPLD liegen. Die PLL befindet sich auf der Rückseite der Karte hinter dem CPLD, damit die Programmiereingänge und Taktausgänge zum CPLD und lokalen Bus kurz gehalten werden.

Die Spezifikation für Tochterplatinen nach dem CMC-Standard [IEEE:CMC] sieht eine rechteckige Platine[4] vor, die mindestens zwei CMC-Buchsen an definierten Stellen und einen sogenannten *IO-Space* an einer Kurzseite hat. Dieser *IO-Space* bedeutet, dass die CMC-Platine die volle Bauhöhe von 10 mm auf einer Länge von 40 mm benutzen darf, um Stecker für externe Anschlüsse bereitzustellen. Daher sind die CMC-Stecker und Haltelöcher so angebracht, dass die CMC-Karte mit dem *IO-Space* zur Rückwand des PCs zeigt und die definierte Frontblende in eine Ausfräsung im PCI-Kartenhalter passt. Dies schliesst nicht nur eine Bestückung der PCI-Karte im Bereich des *IO-Space* aus und legt die Anordnung der zwei standardmässigen CMC-

[1] Steuerleitungen und Taktsignal dürfen bis zu 61 mm lang sein.
[2] siehe 3.3.2
[3] verteilt auf FPGA-Bänke 2 und 3
[4] 80 * 140 mm

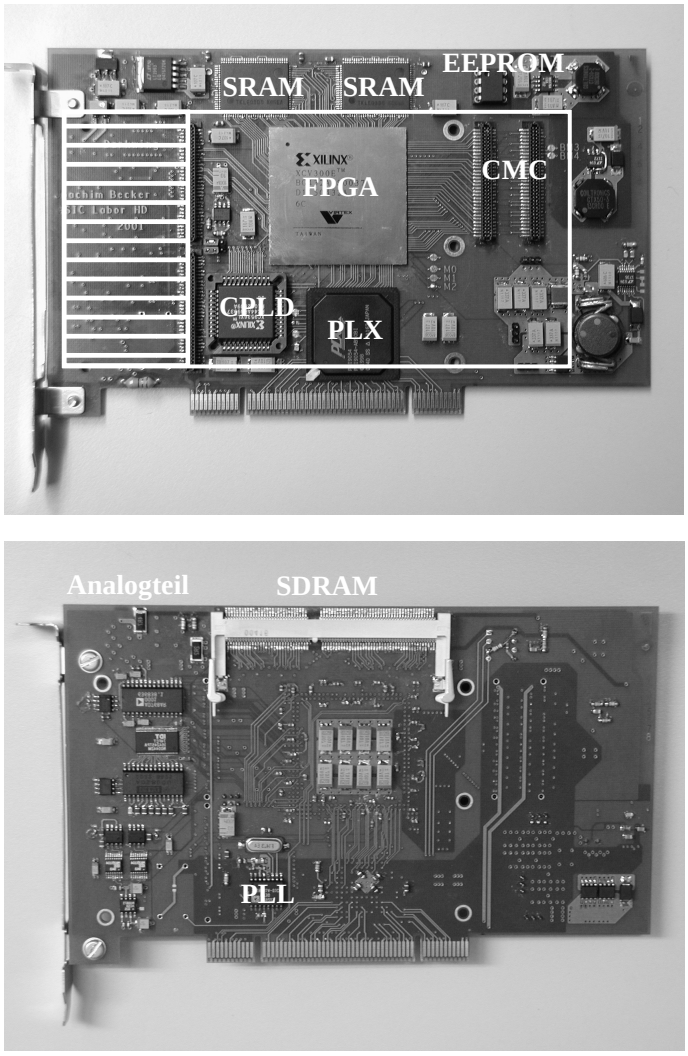


Abbildung 3.2: Anordnung der Bauteile

Stecker fest, sondern definiert zusätzliche Randbedingungen für die verbleibenden Bauteile.

Die für den *IO-Space* freigehaltene Fläche auf der Vorderseite der PCI-Karte ist ideal für die Platzierung der analogen Bauteile auf der Rückseite. In diesem Bereich befinden sich keine digitalen Bausteine, die Rauschen auf der Versorgungsspannung verursachen oder mit digitalen Bussen aus vielen Leitungen Störungen auf analogen Signalen durch Übersprechen hervorrufen können. Die Gruppierung der analogen Bauteile separat von digitalen ermöglicht die Trennung von Spannungsversorgungen, und die Anbringung auf der Rückseite sorgt zusätzlich für eine Abschirmung gegen elektromagnetische Felder, die auf der Oberseite erzeugt werden.

Der CMC-Standard definiert für die zwei obligatorischen Stecker rein digitale Signale (TTL) und eine Anordnung rechts vom FPGA. Darüber hinaus muss es eine Möglichkeit geben, die analogen Signale und digitalen LVDS-Signale mit der Toch-

terplatine auzutauschen. Aus Konsistenzgründen werden dazu auch CMC-Steckverbinder benutzt, jedoch befinden diese sich nicht an einer durch Normen festgelegten Stelle, sondern für die spezielle Funktion dieser Karte sinnvoll direkt neben dem *IO-Space* auf der Trennungslinie zwischen Analog- und Digitalteil. Dadurch können sowohl die digitalen Leitungen vom FPGA angeschlossen werden, als auch die analogen Signale von der Rückseite des *IO-Spaces*.

An der Oberkante des FPGAs liegt das RAM, das in direkter Nähe zum FPGA platziert ist, um die Leitungslänge dieses schnellen Busses zu minimieren. Dort befindet sich auch ein globaler Takteingang des FPGAs, der für die Ansteuerung des SD-RAMs benutzt wird. Die beiden S-RAM Bausteine sind auf der Oberseite zu sehen, währen sich der Sockel für das SD-RAM Modul auf der Unterseite befindet.

Die Anordnung der Netzteile berücksichtigt, dass Schaltregler, die wegen hochfrequenter Schaltvorgänge an Induktivitäten elektromagnetische Störungen verursachen, möglichst weit entfernt von den empfindlichen Analogbauteilen sind und die unkritischen Linearregler für die analoge Spannungsversorgung direkt neben dem Analogteil liegen.

3.1.3 Lagenaufbau

Der Lagenaufbau bezeichnet die interne Struktur der Platine, die aus abwechselnden Lagen von Leitern und Isolatoren besteht. Bedingt durch den Fertigungsprozess besteht eine Platine aus einem Kern aus Isolationsmaterial (glasfaserverstärktes Epoxydharz „FR4“), das auf der Ober- und Unterseite leitend beschichtet ist (Kupferfolie). Um eine mehrlagige Platine zu erhalten, werden mehrere solcher zweilagigen Teile verklebt, so dass immer eine geradzahlige Anzahl von leitenden Lagen entsteht. Die Materialdicke des Klebers (Prepreg) ist genau so gross wie die der Kerne und so wird ein symmetrischer Lagenaufbau erreicht. Einzelne Leiterbahnen entstehen durch Ätzung entsprechender Strukturen vor dem Verpressen, und Verbindungen zwischen Bauteilen können entweder auf einer Lage entlang führen oder durch sogenannte Vias die Lage wechseln.

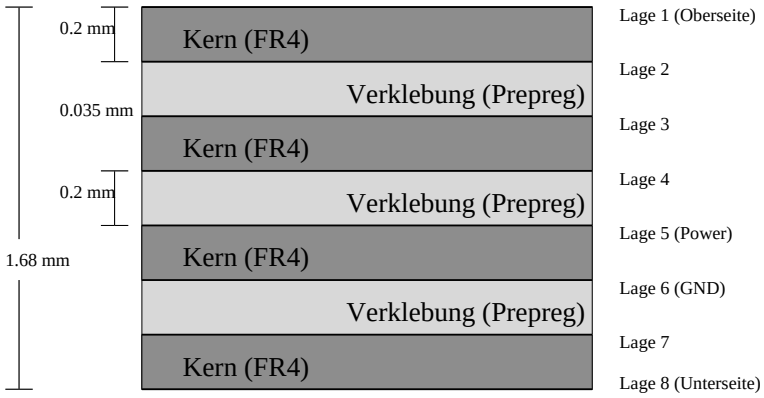


Abbildung 3.3: Lagenaufbau der Platine

Um eine ausreichende Anzahl von Leiterbahnen zu ermöglichen, ist ein achtlagiger Aufbau gewählt, der als Querschnitt in Abbildung 3.3 zu sehen ist. Die inneren Lagen 5 und 6 werden als Flächen belassen und dienen der Spannungsversorgung, sowie als Abschirmung der Analogseite (Unterseite mit einer Ausweichlage) von der Digitalseite (Oberseite und 3 innere Lagen).

3.2 Bauteil-Gruppen zur Initialisierung der Karte

Sobald nach dem Einschalten des Rechners das Signal $\overline{RESET}$ deaktiviert wird, beginnt die Initialisierung der Karte. Dabei sind die Inhalte aller flüchtigen Speicher ungültig und die Bauteile arbeiten mit Standardeinstellungen oder beziehen Konfigurationsdaten aus nicht flüchtigem Speicher. Der PLX-Chip initialisiert den lokalen Bus und meldet sich am PCI-Bus an, und der CPLD arbeitet gemäß seiner statischen Programmierung.

3.2.1 Lokale Taktgenerierung (PLL)

Zunächst wird für die gesamte synchrone Logik ein Taktsignal benötigt. Der lokale Bus kann mit bis zu 50 MHz betrieben werden, und der PLX, FPGA und CPLD benötigen eine entsprechende Taktung, um zu kommunizieren. Der FPGA kann intern diese Zeitbasis teilen oder vervielfachen, um Teile der Logik schneller oder langsamer zu takten. Darüber hinaus kann der FPGA eine zweite globale Taktung erhalten, die asynchron zur ersten ist.

Ein geeigneter Takt wäre der PCI-Takt, der eine feste Frequenz von 33 MHz besitzt. Um eigene Frequenzen zu generieren, mit denen man die Geschwindigkeit des Systems in kleinen Stufen bis zu einem Maximalwert regeln kann, werden jedoch eigene PLLs auf dem Board implementiert. Der [ICS9161A] ist ein Baustein, der zwei unabhängige Taktsignale von 0.39 bis 120 MHz erzeugen kann. Als Eingangsfrequenzen dienen wahlweise ein Quarz oder ein Oszillator, der entweder auf dem Board eingelötet oder über einen CMC-Stecker auch von einer Tochterplatine bereit gestellt werden kann.

Durch ein serielles Programmierinterface, das der CPLD bedient, können die gewünschten Frequenzen eingestellt und zwischen Quarz oder Oszillatoreingang umgeschaltet werden. Dies gibt die Möglichkeit, während des laufenden Betriebes die Eingangsfrequenzen zu wechseln. So können Frequenzen, bei denen das Teilerverhältnis der PLL eine zu schlechte Langzeit-Phasentreue bedingen würde, am externen Eingang eingespeist und mit besseren Teilerverhältnissen getroffen werden oder Synchronisationen zu externen Takten erfolgen.

3.2.2 CPLD

Da der FPGA von der PC-Software aus flexibel über den PCI-Bus programmiert werden soll, ist es nicht sinnvoll, eine feste Programmierung in nicht flüchtigem Speicher

auf der Karte zu hinterlegen. Allerdings kann der FPGA während der Neuprogrammierung nicht eigenständig die Ansteuerung des PCI-Busses übernehmen, sondern die Konfiguration muss von einem anderen Baustein gesteuert werden. Dies passiert durch den fest programmierten CPLD, dem ein Adressbereich auf dem lokalen Bus zugeordnet ist, über den er Konfigurationsdaten für den FPGA vom PC empfängt und im richtigen Protokoll an den FPGA liefert. Auf dieselbe Weise können Konfigurationsdaten für die PLL geschrieben werden, die vom CPLD auf die Programmierleitungen der PLL gegeben werden. Zur Kommunikation auf dem lokalen Bus besitzt der CPLD eine minimale Schnittstelle, bestehend aus acht Datenleitungen und den vier notwendigen Steuersignalen.

PLL-Kontrolle und Taktsignalkonvertierung

Der XC9536XL ist ein CPLD, der mit 3.3 V betrieben wird, aber 5 V-tolerante Eingänge hat. Dies wird ausgenutzt, um die beiden Taktsignale, die von der PLL mit 5 V getrieben werden, in den CPLD einzuspeisen und dann mit 3.3 V zum lokalen Bus und FPGA weiterzuleiten, denn die Eingänge des FPGAs sind nur mit maximal 3.3 V zu betreiben. Dies gibt neben dem notwendigen Pegelwandel die Möglichkeit, die Takte im CPLD ein- und auszuschalten oder zu halbieren/verdoppeln.

Die Weiterleitung des Taktes geschieht asynchron, während die PLL-Programmierung Teil einer synchronen state machine ist, die die Daten vom lokalen Bus entweder in das Programmier-timing der PLL umwandelt oder die Programmierung des FPGAs vornimmt.

FPGA Initialisierung

Die FPGAs der Virtex-E Reihe (siehe Abschnitt 3.3) unterstützen vier verschiedene Methoden zur Programmierung, die über Lötbrücken *M0, M1, M2* auf der Platine gewählt werden können. Neben seriellen Protokollen wird auch der *SelectMAP*-Modus bereitgestellt, der eine 8 Bit parallele Datenübertragung erlaubt. Dieser Modus wird vom CPLD unterstützt, um schnelle Rekonfiguration während des Betriebs zu erlauben. Dazu werden acht Datenleitungen sowie FPGA_CCLK, FPGA_DONE, FPGA_CS, FPGA_WRITE, FPGA_PROG und FPGA_INIT benutzt.

Für die einsetzbaren FPGA-Typen XCV300E, XCV400E und XCV600E ist die Größe des Konfigurationsbitstromes 1.8 bis 3.9 MBit. Bei Ausnutzung der maximalen Geschwindigkeit des lokalen Busses von 50 MHz kann eine Initialisierung in 9 bis 20 ms erfolgen, während die serielle Übertragung entsprechend acht mal langsamer wäre.

Die Initialisierung des FPGAs [Xil:Data] beginnt automatisch beim Einschalten oder durch Betätigen der FPGA_PROG Leitung. Daraufhin löscht der FPGA die internen Speicherzellen und signalisiert Bereitschaft mit FPGA_INIT. Nun muss der Konfigurationstakt FPGA_CCLK bereitgestellt werden, und sobald FPGA_CS und FPGA_WRITE vom CPLD aktiviert werden, empfängt der FPGA die Initialisierungsdaten. Falls die Taktfrequenz unter 50 MHz liegt, ist sichergestellt, dass der FPGA die Daten schnell genug verarbeiten kann. Höhere Frequenzen bis zu 66 MHz

sind nur mit Handshake möglich, damit der FPGA das Schicken weiterer Pakete unterbrechen kann. In der vorliegenden Implementierung wurde darauf aber verzichtet.

Sobald der FPGA fehlerfrei Daten in der Länge des Bitstroms empfangen hat, wird FPGA_DONE aktiv und der CPLD erkennt, dass die Konfiguration abgeschlossen ist. CCLK bleibt bestehen, denn nach dem Empfangen der Daten muss der FPGA noch weitere Initialisierungen vornehmen. Der genaue Ablauf dieser Interna kann im Konfigurationsbitstrom definiert werden.

Abschaltung des ADC-Buffers Zum Abschluss dieses Abschnitts ist zu bemerken, dass die FPGA_D Programmierdatenleitungen nicht nur vom CPLD zum FPGA gehen, sondern nach Abschluss der Programmierung im FPGA auch als Eingänge für die Daten des ADCs benutzt werden. Um die Konfiguration nicht zu stören, muss der Ausgangsbuffer des ADCs [FCT164245] deaktiviert werden, damit er nicht gleichzeitig zum CPLD versucht, die Leitungen zu treiben. Deshalb wird der Konfigurationsmodus des FPGAs gewählt[5], der vor der Programmierung alle Ausgänge auf *high* setzt, damit auch der *Output Enable* $\overline{OE}$ des Buffers *high* und damit ausgeschaltet ist. Ist der FPGA initialisiert, schaltet der CPLD seine Treiber inaktiv, und die Leitungen sind als Eingänge für den ADC im FPGA verfügbar.

3.2.3 PLX

Das standardisierte PCI-Bus Protokoll sieht vor, dass es einen Bus-*Master* und einen *Slave* gibt, wobei der *Master* immer der Initiator eines Datentransfers auf ein Ziel (Target) ist. Obwohl es Bausteine mit integrierter Logik für PCI-Kommunikation gibt[6], erfolgt die Anbindung der Prozessoren gewöhnlich über Wandler-Bausteine, die das komplizierte Timing auf dem Bus[7] in ein einfacheres, zeitlich nicht mit dem PCI-Bus korreliertes Protokoll auf einem lokalen Bus abbilden. Der [PLX9054] ist ein PCI-Bus *Master*-Chip, er kann also nicht nur Target eines Zugriffs sein, sondern auch selber Zugriffe initiieren. Die speziellen Fähigkeiten des *Master*-Chips werden benötigt, um DMA-Zugriffe benutzen zu können.

Ein normaler Datentransfer wird vom *Master* initiiert, und der *Slave* liest oder schreibt in einem festgelegten Protokoll einzelne Datenworte. Einzelne Wortübertragungen sind ineffizient, weil auf dem PCI-Bus Daten und Adressen auf den 32 Datenleitungen gemultiplext werden. Ausgehend davon, dass eine Lokalität der Daten besteht, ist es sehr wahrscheinlich, dass oft längere zusammenhängende Bereiche transferiert werden. Daher ist der PCI-Bus *burst*-orientiert. *Burst*-Zugriff bedeutet die Übertragung von mehreren aufeinander folgenden Datenworten, wobei zu Beginn nur einmal die Startadresse übergeben wird und das Target die Adressen selber entsprechend inkrementiert. Je länger ein *Burst* andauert, desto effizienter ist die Benutzung des Busses, da die Übertragung von Adressen entfällt.

[5] Lötbrücke *M2* geschlossen
[6] sowohl ASICs als auch FPGAs
[7] u.U. mit langen Wartezeiten auf die Antwort eines *Slaves*

Es gibt einen entscheidenden Unterschied zwischen Lesen und Schreiben auf dem PCI-Bus. Beim Schreiben liegen die Daten dem Initiator vor und das Target kann die Übertragung so lange abblocken, bis es bereit zum Empfang ist. Dies geschieht durch das Signal RETRY#, wobei der Bus zwischen den Versuchen anderen Transaktionen zur Verfügung steht. Ist das Target bereit, wird nur die Zeit auf dem Bus verbraucht, die wirklich zur Übertragung notwendig ist.

Beim Lesen gibt der Initiator die Adresse des gewünschten Datums und wartet auf die Bereitstellung der Daten. Dabei hält er den gesamten Bus blockiert, während das Target die Daten (u.U. auf einem eigenen lokalen Bus) aquiriert und sendet. Es ist also wesentlich effizienter, statt eines Lesezyklus dem gewünschten Target nur die Adresse zu senden und diesen Schreibvorgang dann abzuschließen. Dann ist der PCI-Bus wieder frei, bis der angesprochene *Slave* die Daten bereit hat, den PCI-Bus für sich reserviert und als Initiator die gewünschten Daten im *Burst* zurück *schreibt*. Damit hat man einen langen, unteilbaren Lesevorgang in zwei effiziente Schreibvorgänge aufgeteilt. Dies ist der Grund, warum ein *Master*-Chip verwendet wird, der als Initiator arbeiten kann.

Beim ersten Einschalten des PCs fragt das BIOS[8] den PCI-Bus ab und prüft in der so genannten *Enumeration*, welche Steckplätze mit welchen Karten belegt sind. Außerdem geben die Karten an, wieviel Platz des adressierbaren 32 Bit (= 4 GByte großen) Adressraumes sie reservieren möchten. Das Betriebssystem verwaltet dann eine Tabelle der zugeordneten Adressierungsbereiche und kann so auf jede Karte individuell zugreifen. Auf der anderen Seite hat der lokale Bus des PLX-Chips auch 32 Bit, und da die Aufteilung der Adressräume der beiden Busse im Allgemeinen nicht übereinstimmt, muss der PLX-Chip die Abbildung zwischen den Adressbereichen der verschiedenen Busse vornehmen. In welcher Weise dieses *Mapping* funktioniert, kann beeinflusst werden[9].

Der PLX unterstützt drei Betriebsmodi des lokalen Busses. Neben einer speziellen Anpassung an das Protokoll eines Motorola-Prozessors unterscheidet man zwischen je einem Modus mit und ohne Daten/Adress-Bündelung. Wie bereits erwähnt, ist der PCI-Bus ein gebündelter Bus, da sowohl Daten als auch Adressen über dieselben Busleitungen nacheinander übertragen werden. Ein nicht gemultiplexter Bus hat zusätzliche Leitungen für die Adressen und kann deshalb gleichzeitig Adressen und Daten liefern. Diesem potentiellen Geschwindigkeitsgewinn steht die verdoppelte Anzahl von Busleitungen entgegen. Da aber der PCI-Bus selbst mit 33 MHz und gemultiplext läuft und der lokale Bus mit bis zu 50 MHz, wird selbst ein gemultiplexter lokaler Bus schnell genug sein, um die Leistungsfähigkeit des PCI-Busses auszunutzen.

3.2.4 EEPROM

Die zuvor erläuterten Einstellungen müssen vor der *Enumeration* durch das BIOS im PLX feststehen, sollen aber dennoch mit mäßigem Aufwand rekonfigurierbar sein.

[8] basic input output system
[9] siehe dazu auch [PLX9054] und [Bau01]

Deshalb wird die Konfiguration nicht im PLX-Chip hinterlegt, sondern auf einem externen seriellen EEPROM. Dieses EEPROM wird in einem DIL-8 Gehäuse auf einen Sockel auf der Karte gesteckt und kann so in einem separaten Schreibgerät programmiert, aber auch auf der Karte benutzt werden. Die Standardkonfiguration findet sich in Anhang A.

Der Zugriff auf das EEPROM erfolgt mit einem einfachen seriellen Protokoll auf den drei Signalen *Chip-Select*, *Clock* und *Data* wie in [EEPROM] beschrieben. Der verwendete Typ 93LC56 unterstützt *sequential read*, das einem *Burst*-Zugriff mit kompletter Auslese des gesamten Speichers entspricht. Beim Einschalten des PCs liest der PLX-Chip die Konfigurationsdaten aus dem EEPROM und übernimmt sie in interne Register. So kann sich die Karte an der *Enumeration* beteiligen und am Betriebsystem angemeldet werden.

Ist der Bootvorgang durchgeführt, kann auf den PLX und den lokalen Bus zugegriffen werden. Außerdem ist es möglich, über die serielle Verbindung Daten auf das EEPROM zu schreiben. Da der PLX aber nur einen *open-drain* Treiber für diese Leitung besitzt, muss ein externer *Pullup*-Widerstand am EEPROM die Treiberleistung bringen. Dieser Widerstand ist im Layout nicht vorgesehen, und somit wird ein versehentliches Überschreiben des EEPROMs ausgeschlossen.

3.3 FPGA

Der FPGA ist das Herzstück des Testsystems, weil er die „lokale Intelligenz“ jeder einzelnen Karte darstellt. Er definiert durch seine Konfigurationsdaten die Anpassung der allgemeinen Hardware an den jeweils zu testenden ASIC und ist der Schlüssel zur Benutzung der lokalen Hardware.

Ein FPGA ist ein Mikrochip, der aus einer Matrix von einigen hundert bis zu zehn tausend CLBs[10] besteht. Jeder dieser CLBs ist eine programmierbare digitale Zelle, die binäre Funktionen mit mehreren Ein- und Ausgängen darstellen kann. Ebenfalls konfigurierbar ist die Verschaltung dieser Zellen untereinander, so dass komplexe binäre Funktionen in Hardware implementiert werden können. Moderne FPGAs beinhalten außerdem Speicherzellen, konfigurierbare IO-Zellen, die vielfältige Standards zur Kommunikation mit anderen Bausteinen bereitstellen und herstellerspezifische Erweiterungen.

Entsprechend der Anforderungen an den FPGA, wie Matrixgröße, Geschwindigkeit, IO-Fähigkeiten und der technischen Möglichkeiten der heute erhältlichen FPGA-Typen, fiel die Wahl auf die Virtex-E Reihe der Firma Xilinx. Die Wahl des Gehäuses legt fest, welche Typen der Reihe verfügbar sind, und in diesem Projekt wird das BG432-Gehäuse benutzt, welches durch ein *ball grid array* mit 1.27 mm Rasterabstand und 432 Kontakten auf der Unterseite auf die Platine aufgebracht wird. In diesem Gehäuse verfügbar sind die Typen XCV300-E, XCV400-E und XCV600-E jeweils in den Geschwindigkeiten 6, 7 und 8. Es besteht also die Möglichkeit, den FPGA je nach Komplexität des Testsystems auszuwählen.

3.3.1 IO-Aufteilung

Das BGA432 hat neben Stromzuführungen und fest zugeordneten Anschlüssen 316 frei belegbare digitale Ein- und Ausgänge[11]. Diese IOs werden im FPGA eingeteilt in acht Bänke, und die Aufteilung der wichtigsten externen IO-Blöcke ergibt sich aus der Orientierung des FPGAs auf der Platine wie in Abbildung 3.4 angedeutet. Diese Orientierung stimmt mit der Sicht auf die Oberseite (Abbildung 3.2) überein.

Den größten Teil nimmt das RAM ein, denn es benötigt 18 Adress-, 64 Daten- und 17 Steuerleitungen. Die standardisierten CMC-Stecker 1 und 2 befinden sich mit jeweils ca. 40 digitalen IOs rechts, und der lokale Bus mit 32 Daten- und 6 Steuerleitungen ist auf der rechten Unterseite angesiedelt, wo in direkter Nachbarschaft der PLX-Chip anschließt. Die selbst definierten CMC-Stecker 3 und 4 liegen auf *Darkwing* links vom FPGA und haben jeweils ca. 30 digitale IOs. Die übrigen Leitungen werden auf die Analogbauteile verteilt.

Die Aufteilung der Gruppen möglichst auf abgeschlossene Bänke ist aus zwei Gründen sinnvoll. Zum einen ist eine effiziente Platzierung der Logik im FPGA möglich, wenn Teile, die sich auf die Kommunikation mit einem bestimmten externen Chip beziehen, innerhalb einer Bank bzw. in benachbarten Bänken Platz finden und in

[10] configurable logic block
[11] im folgenden kurz IOs genannt

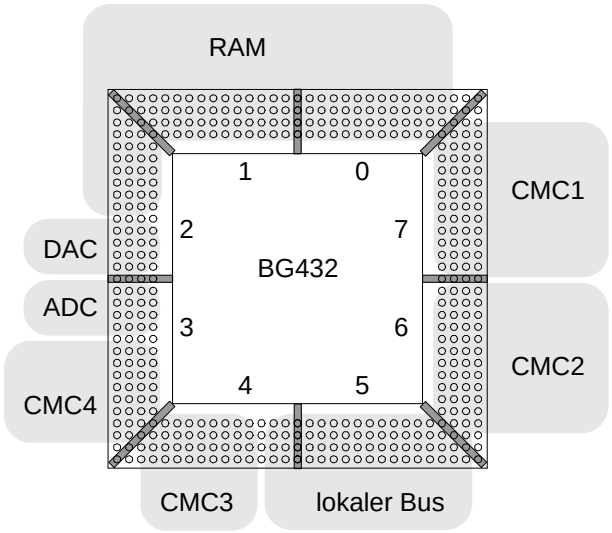


Abbildung 3.4: Aufteilung der IO-Bänke des FPGAs

räumlicher Nähe in der Matrix angesiedelt werden. Auch die Leiterbahnen auf der Platine können kreuzungsfrei und platzsparender verlegt werden, wenn Busse dicht nebeneinander liegen und vom FPGA direkt in Richtung des betreffenden Bauteils führen. Zum anderen kann die Wahl des in den IO-Treibern benutzten Logik-Standards (LVDS / LVTTL / etc.) nur bankweise zugeordnet werden und erfordert unter Umständen eine alternative Spannungsversorgung der gesamten Bank. Deshalb sollten sich die gleichartigen IOs in einer Bank befinden, um von anderen Logik-Standards separiert zu sein.

Die Bestrebungen, diese Richtlinien einzuhalten, haben Grenzen. So bleiben nach der Zuordnung der breiten Busse einzelne Leitungen übrig, die auf die vereinzelten, noch nicht zugeordneten Anschlüsse in den Bänken verteilt werden müssen. Dies betrifft die langsamen seriellen Leitungen zu den 12 Bit-DACs, einzelne Taktsignale oder andere Signale, die zeitlich unkritisch sind.

3.3.2 FPGA-Taktung

Der FPGA besitzt intern vier globale Netze für die Verteilung von Taktsignalen, die entweder von externen oder internen Signalen gespeist werden können. Wie bereits in 3.2.1 bzw. 3.2.2 beschrieben, erzeugt die PLL zwei Taktsignale auf TTL-Basis. Da die Eingangsbuffer des FPGAs aber maximal mit LVTTL mit 3.3 V betrieben werden dürfen, ist der CPLD mit 5 V-toleranten Eingängen und 3.3 V-Ausgängen dazwischen geschaltet, um die Signale an zwei globale Takteingänge des FPGAs zu bringen. Eine Taktfrequenz ist identisch mit der des lokalen Busses, die andere ist frei programmierbar.

Ein dritter Eingang wird für die SD-RAM Ansteuerung benötigt. Die Virtex-E FPGAs besitzen acht DLLs[12], die entweder zur Verdopplung oder Phasenangleichung

[12] delay locked loop

von Taktfrequenzen benutzt werden können [Xil:Data]. Bei der Ansteuerung des SD-RAMs benutzt man zwei gekoppelte DLLs, die diese Fähigkeiten ausnutzen. Die interne Logik des FPGAs kann mit einer niedrigen Frequenz von 50 MHz betrieben werden, während eine DLL diese verdoppelt und das RAM extern mit 100 MHz taktet. Bei derart hohen Frequenzen spielt die Laufzeit der Signale eine Rolle, denn das RAM erhält seine Taktung vom FPGA über einen Ausgangstreiber und eine Leitung mit gewisser Laufzeit. Werden Daten zum RAM geschrieben, erfahren Daten und Taktsignal dieselbe Verzögerung, und die Flanken von Daten und Takt sind synchron. Beim Lesen werden die Daten vom RAM zu einer steigenden Taktflanke auf den Bus gegeben. Da sich aber die Verzögerung des Taktsignals vom FPGA zum RAM und die Verzögerung der Daten vom RAM zum FPGA addieren, sind die Daten erst einige Zeit nach der steigenden Flanke am FPGA gültig. Bei einer Frequenz von 100 MHz dauert ein Zyklus 10 ns, und die Summe der Verzögerungen kann diese Zeitspanne überschreiten. Daher kann es passieren, dass ein verzögertes Signal nicht mehr zur steigenden Flanke des unverzögerten Taktsignals am FPGA empfangen werden kann und das Lesen scheitert.

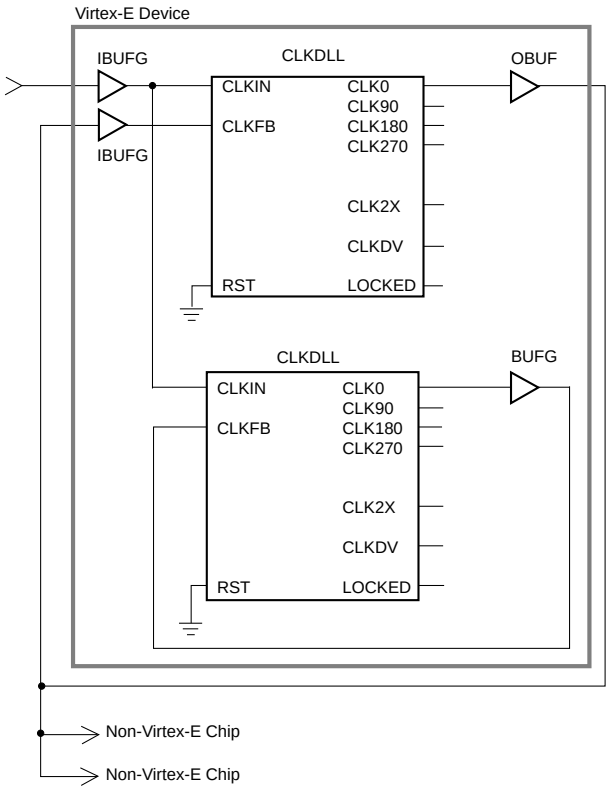


Abbildung 3.5: Taktangleichung (*clock mirror*) mit DLLs [Xil:Data]

In [Xil:DLL] und [Xil:Data] ist ausgeführt, wie zwei DLLs benutzt werden, um die Verzögerung der Leitungen zu kompensieren. Über einen Abgriff des Taktsignals auf halber Länge zu den Empfängerbausteinen (hier RAM) wird eine Rückkopplung

zum FPGA hergestellt, und die Taktflanke kommt gleichzeitig am Empfänger und am Rückkopplungseingang des FPGAs an.

Die Kombination der beiden DLLs wird verdeutlicht durch das Blockdiagramm in Abbildung 3.5. Die dargestellten Komponenten sind schematisierte Bibliotheksfunktionen, die als vordefinierte Blöcke verfügbar sind und im VHDL-Code benutzt werden können. Beide DLLs erhalten die interne Taktung an den Anschlüssen $CLKIN$. Die obere DLL kann über die Rückkopplung externe Verzögerungen kompensieren. Das Ausgangssignal an $CLK0$ wird dabei so eingestellt, dass die Phase des rückgekoppelten Signals $CLKFB$ exakt mit der des internen $CLKIN$ übereinstimmt. Durch die gleiche Weglänge der Rückkopplung und der Leitung zum Empfänger ist damit auch eine Taktflanke am Empfänger exakt gleichzeitig zum internen Signal.

Die untere DLL erzeugt die Taktung für die Empfängerlogik im FPGA. Dadurch, dass die Leitungen durch interne Buffer $BUFG$ gehen, die in der Verzögerung dem Ausgangsbuffer $OBUF$ entsprechen, kann durch zusätzliche interne Rückkopplung eine vollkommene Synchronität der IO-Buffer des FPGAs und des externen Empfängers mit dem internen Eingangstakt erreicht werden.

Der vierte der globalen Takteingänge ist auf einen CMC-Stecker gelegt, um eine Taktung des FPGAs von einer Tochterplatine aus zuzulassen. Dieser Eingang ist auch im LVDS-Modus zu benutzen, um die Register des FPGAs mit den Datensignalen des LVDS-Busses synchronisieren zu können. Dies ermöglicht quellensynchrone Übertragung, bei der das Taktsignal jeweils von dem Baustein erzeugt wird, der Daten sendet.

3.3.3 LVDS/LVTTL-Bänke

Die Bänke 3 und 4 des FPGAs sind an die CMC-Stecker 4 und 3[13] verbunden, die keiner Norm unterliegen und deshalb frei belegt werden können. Dies wird ausgenutzt, um die Verteilung der Signale auf den Stecker so zu wählen, dass LVDS-Signale als differentielle Leitungspaare verwirklicht werden, was im folgenden erläutert wird.

Ein typischer LVDS-Kanal ist in Abbildung 3.6 dargestellt. Eine Stromquelle von 3.5 mA im Sender treibt einen Stromfluss über die Leiterschleife. Direkt vor dem Empfänger ist ein Widerstand, über dem der Spannungsabfall gemessen wird. Typischerweise wird mit einer Spannungsdifferenz von 350 mV über einem 100 Ω Abschlusswiderstand gearbeitet. Die logischen Zustände *high* und *low* werden durch die Stromrichtung repräsentiert.

Diese Schleife ist weniger anfällig gegen Störungen als herkömmliche Verbindungen, die eine Spannungsdifferenz zwischen einer Leitung und einem gemeinsamen Massepotential benutzen. Offensichtlich sind getrennte Stromrückflüsse die beste Möglichkeit, um die Signale zu separieren und ein Übersprechen, das von der gemeinsamen Masse herrührt, zu vermeiden. Dies verdoppelt zwar die Anzahl der benötigten Leitungen, was für einen Bus unter Umständen viel Platz verbraucht, aber es reduziert auf der anderen Seite die Anzahl der Masseverbindungen. Darüber hinaus schafft es die Möglichkeit, die elektrischen Eigenschaften der Übertragungsleitungen

[13] siehe Abbildung 3.4

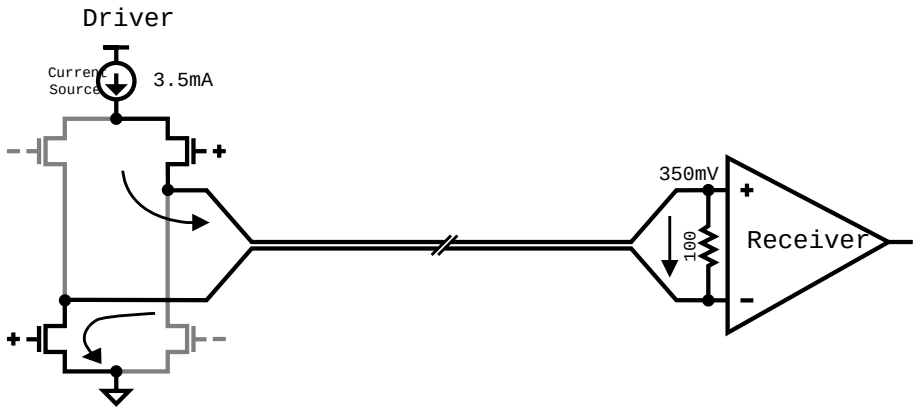


Abbildung 3.6: Ein LVDS Übertragungskanal [LVDS]

zu verbessern, um höhere Übertragungsraten zu erreichen. Die Leitungen werden in minimalem Abstand nebeneinander geführt. Dabei wird ausgenutzt, dass die Leitungen ähnlich auf elektromagnetische Störfelder reagieren, da sie nahezu am selben Ort liegen und die selbe Ausrichtung zum Feld haben. Weit auseinanderliegende Leitungen oder harte Ecken in den Leitungen führen dazu, dass Felder in unterschiedlicher Weise auf die Leitungen wirken können. Den Extremfall stellt der bei TTL übliche Stromrückfluss über eine Massefläche dar, wo ein Störfeld auf die frei laufende einzelne Leitung vollkommen anders wirkt als auf den zugehörigen Rückfluss. Da das Signal am Empfänger definiert ist als die Spannungsdifferenz an der Terminierung, wird eine so genannte Gleichtaktstörung dadurch unterdrückt, dass sie auf beide Leitungen gleichermaßen wirkt und der Hin- und Rückfluss nicht verschiedenen Einflüssen ausgesetzt sind.

Diese Verbesserung der Signalqualität ermöglicht erst den kleinen Spannungshub von einigen hundert mV und somit auch eine schnellere Umkehr der Stromrichtung. Somit können die Taktzeiten verkürzt und Taktfrequenzen von bis zu zwei GHz erreicht werden. Diese Techniken stellen aber auch höhere Ansprüche als bisher an das Leiterplattenlayout, denn bei hohen Geschwindigkeiten spielen Reflexion und Dämpfung von Signalen eine immer größere Rolle.

Aus der Telegraphengleichung[14] folgt, dass ein Signal auf einer realen Leitung mit Kapazität C pro Länge und Induktivität L pro Länge als Welle aufgefasst werden kann. Bei endlichen Leitungen tritt daher eine Reflexion des Signales am Ende auf, die in entgegengesetzter Richtung auf der Leitung weiter läuft. Bei einer getakteten Signalübertragung kann es passieren, dass die reflektierte Welle sich mit der einlaufenden Welle oder der des darauffolgenden Taktes überlagert und so das Signal verfälscht. Die Spannungsamplitude U am Ort x ist

$$U(x) = U_{rück}(x) + U_{hin}(x) = Ae^{-\gamma x} + Be^{\gamma x} \quad ; \quad \gamma = i\omega\sqrt{LC} \tag{3.1}$$

[14] siehe jedes Elektronik-Lehrbuch; beispielsweise [Hin96, Formel (2.15)]

Der Reflexionsfaktor r ist gegeben durch das Verhältnis der Amplituden der ein- und auslaufenden Welle und schreibt sich mit der Impedanz Z der Leitung und dem Abschlusswiderstand R_a als

$$r = \frac{U_{rück}}{U_{hin}} = \frac{R_a - Z}{R_a + Z} \qquad ; \qquad Z = \sqrt{L/C} \tag{3.2}$$

Bei hohen Übertragungsgeschwindigkeiten besteht keine Zeit, auf ein Abklingen der Reflexionen durch parasitäre Dämpfung der Leitung zu warten, sondern die Impedanz der Leitungen muss auf den Abschlusswiderstand angepasst sein, um die gesamte Energie der einlaufenden Welle aufzunehmen und Reflexion zu verhindern. Es ist sofort ersichtlich, dass bei gegebenem Abschlusswiderstand die Impedanz der Leitung gleich dem Abschlusswiderstand sein muss, um eine Reflexion zu vermeiden, denn der Einfluss der reflektierten Welle wird mit der Differenz der Widerstände größer. Für die Fälle $R_a < Z$ und $R_a > Z$ erfolgt Reflexion mit Phasenverschiebung kleiner bzw. größer 90°, wie aus der Wellenlehre bekannt.

Dies ist der Grund, warum LVDS nicht als das Treiben eines Signals auf einen Spannungspegel, sondern eines Stromes bei gegebenem Abschlusswiderstand definiert ist. Bei Standard TTL[15] oder LVTTL ist weder der Widerstand des Empfängers noch die Impedanz der Leitung festgelegt, was bei einer beliebigen Treiberleistung zu unkontrollierbaren Reflexionen führt. Abhilfe schafft hier nur eine einstellbare Treiberleistung, wie sie die IO-Zellen des FPGAs bieten, oder die Terminierung der Leitung durch einen Widerstand.

Um die Impedanz einer differentiellen Leitung zu kontrollieren, genügt es nicht, die Kapazität und Induktivität wie im Falle einer einzelnen Leitung auszurechnen, sondern es muss neben der kapazitiven Kopplung an eine Potentialfläche auch die Kopplung an die eng anliegende Nachbarleitung einbezogen werden. Abbildung 3.7 verdeutlicht den Verlauf der Feldlinien der zwei Standardkonfigurationen *Microstrip* und *Stripline*.

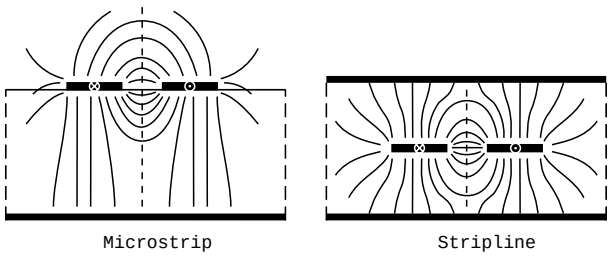


Abbildung 3.7: Feldlinien verschiedener LVDS Konfigurationen [LVDS]

Bei der *Microstrip* Leitung liegen die Leiterbahnen auf der Oberseite der Platine im Feld einer Potentialfläche (Masse oder Versorgungsspannung). Im Falle der *Stripline* ist das Leitungspaar eingeschlossen zwischen zwei Potentialflächen. Wichtig ist immer die Symmetrie sowohl des Leitungspaares untereinander, als auch der Lage des Paares in einem äußeren Feld, damit Felder auf die Leitungen gleich wirken.

[15] transistor-transistor logic

Zur Berechnung der Impedanz unter Berücksichtigung der kapazitiven Kopplung gibt es heuristische Formeln [LVDS], die einige Standardfälle abdecken. Für eine genaue Berechnung auch besonderer Anordnungen existieren Computersimulationen, die durch lineare Approximation finiter Elemente den Feldverlauf annähern und aus der Geometrie der Leiter die Impedanz berechnen.

Zur Spezifikation der Anordnung benötigt das Simulationsprogramm [Polar] die Abmessungen und den Abstand der differentiellen Leitungen, den Abstand zu festgelegten Potentialflächen und die Dielektrizitätskonstanten der Isolatoren (Luft, Kernmaterial oder Verklebung). Durch den Lagenaufbau[16] sind die Dicke der Kerne und der Verklebung mit 0.2 mm festgelegt sowie die Dicke der Leiterbahnen mit 0.035 mm. Die Dielektrizitätskonstanten von FR4 und Prepreg sind ungefähr gleich und näherungsweise mit $\epsilon_r = 4.5$ anzunehmen.

Auf der PCI-Karte sollen LVDS-Busse mit 15 Bit zu CMC3 und 13 Bit zu CMC4 auf engstem Raum untergebracht werden. Daher kann nicht auf eine einfache Konfiguration zurückgegriffen werden, wo zum Beispiel alle Leitungen nebeneinander als Microstrip verlegt werden. Der Lagenaufbau sieht über der flächigen Lage 5 vier Lagen für digitale Leitungen vor. Um genügend großen Abstand zwischen den Paaren zu ermöglichen, wird im Bereich der LVDS-Leitungen eine Abschirmungsfläche auf Lage 3 eingezogen, damit es zwei unabhängige Lagen für Striplines (Lage 4) und Microstrip (Oberseite bzw. Lage 2) gibt. Leitungen auf der Oberseite haben den Vorteil, dass sie keine Vias benötigen, die Reflexionen verursachen können. Falls aber auf der Oberseite Bauteile sind, die keinen Platz für Leiterbahnen erlauben, muss auf Lage 2 ausgewichen werden.

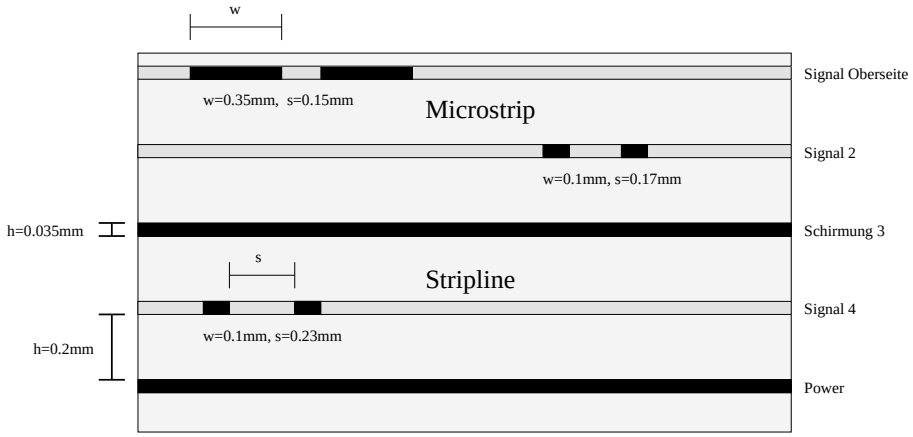


Abbildung 3.8: Querschnitt-Sicht auf LVDS Leitungen

Die Prozessparameter der Herstellung erlauben eine minimale Leiterbahnbreite und einen minimalen Isolationsabstand von 0.1 mm. Die kleinste Realisierung von LVDS-Leitungen mit kontrollierter Impedanz von 100 Ω ist in Abbildung 3.8 dargestellt, wobei über der obersten Lage noch Lötstopp-Lack angenommen wird. Die

[16] siehe Absatz 3.1.3

korrekten Bezeichnungen der Übertragungswege sind: coated microstrip (Lage 1), embedded microstrip (Lage 2) und symmetrical stripline (Lage 4).

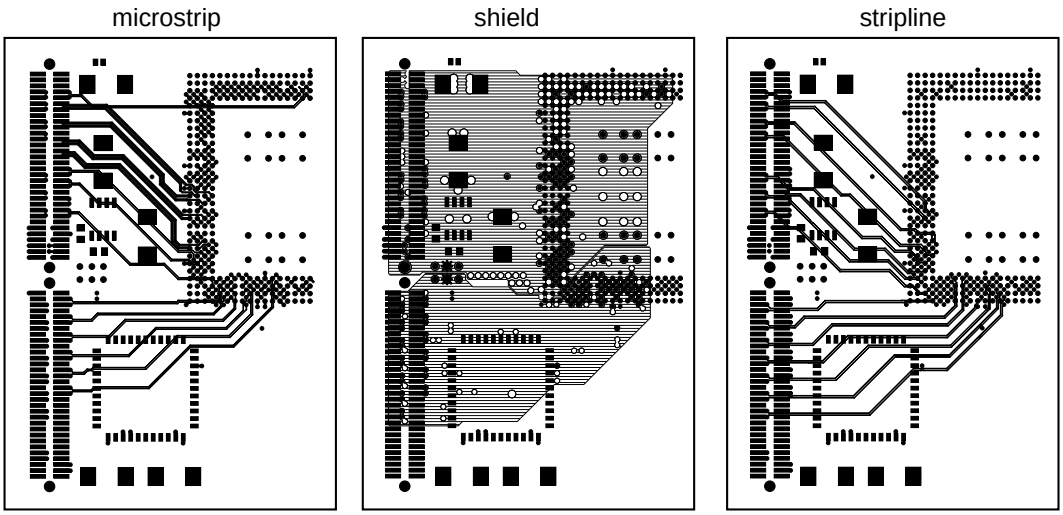


Abbildung 3.9: Aufsicht auf LVDS Leitungen

Die Aufsicht auf die Lagen zeigt Abbildung 3.9, wobei zur Orientierung immer auch die Pads der Oberseite eingezeichnet sind. Dies verdeutlicht, warum die Microstrip Leitungen im Bereich des CPLD (unten) und des 2.5 V Spannungsreglers (oben) nicht auf der Oberseite sondern auf Lage 2 laufen müssen. Da LVDS-Bänke mit 2.5 V IO-Spannung betrieben werden, ist der Regler in der Nähe der Bänke angesiedelt. Auf Lage 3 ist die flächige Abschirmung zu erkennen, die ausserdem die Versorgungsspannung vom Regler zu den LVDS-Bänken leitet. Lage 4 wird hier ausschließlich für das Verlegen der Striplines benutzt.

3.4 Lokaler Speicher

Die PCI-Karte verfügt über 2 S-RAM Bausteine zu je 1 MByte und einen Sockel für bis zu 128 MByte SD-RAM. Obwohl die beiden RAM-Typen grundlegend verschieden sind, können die Daten- und Adressleitungen gemeinsam benutzt werden. Dies verhindert zwar eine gleichzeitige Benutzung der beiden RAMs, jedoch bleibt es möglich, in kurzer Zeit zwischen den RAMs umzuschalten und sie uneingeschränkt nacheinander zu benutzen. Dies wird durch eine separate Beschaltung der *Chip-Select* Steuersignale erreicht.

Im Allgemeinen ist die Ansteuerung des S-RAMs einfacher zu realisieren und 2 MByte RAM sind standardmäßig auf jeder Karte vorhanden und reichen für viele Anwendungen aus. Soll ein speicherintensiveres System aufgebaut werden, das viele Daten lokal lagert, so kann optional bis zu 128 MByte SD-RAM aufgesteckt werden. Aus Platzgründen befindet sich ein SO-DIMM-Sockel mit 144 Anschlüssen [SDModul] auf der Rückseite der Platine. In [Xil:DLL] ist eine mögliche Ansteuerung des SD-RAMs beschrieben sowie als VHDL-Code verfügbar und kann mit Anpassungen an die spezifische Anschlussbelegung des FPGAs verwendet werden.

3.4.1 Statisches (S-) RAM

Die Bausteine der Reihe K7M-803625 [SRAM] sind S-RAMs mit 1 MByte Speicherkapazität in einer 36 Bit breiten, 256 K tiefen Architektur, die in Abbildung 3.10 vereinfacht dargestellt ist. Die Breite ist aufgeteilt in vier Worte zu 9 Bit, damit ein Byte Nutzdaten und ein Paritätsbit gespeichert werden können. Da das Paritätsbit aber weitere vier Signale benötigt und von dem SD-RAM Modul nicht unterstützt wird, wird in diesem Fall auf eine Benutzung verzichtet. Die Bausteine sind so genannte *no turnaroud*-SRAMs, das bedeutet, dass Lese- und Schreibzyklen ohne Wartezyklen wechseln dürfen, was die Ansteuerung vereinfacht.

Um eine hohe Bandbreite und doppelte Speicherkapazität zu erreichen, werden zwei S-RAM Bausteine nebeneinander an einen 64 Bit breiten Datenbus angeschlossen, so dass sie dieselben Steuersignale bekommen und sich wie ein einziger Speicher verhalten. Dabei wird die byteweise Aufteilung der Bus-Datenleitungen auf die RAM-Anschlüsse durch eine günstige Anordnung im Layout bestimmt.

Neben den Adress- und Datenleitungen besitzt das RAM die folgenden Steuerleitungen:

Takt: Der Anschluss CLK ist der Eingang für die Zeitbasis, zu deren steigenden Flanken Zugriffe auf den Speicher erfolgen können. Durch das *low* aktive $\overline{CKE}$-Signal kann diese Taktung gehemmt werden, so dass keine Kommunikation mit der Logik des Bausteins möglich ist, weil keine Befehle decodiert werden können. Mit den benutzten Bausteinen kann eine maximale Taktfrequenz von 100 MHz erreicht werden.

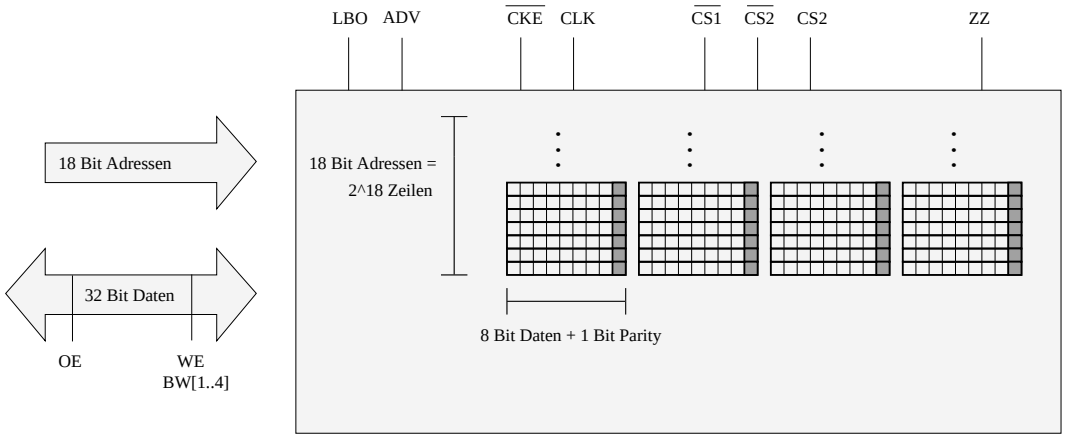


Abbildung 3.10: Speicheraufbau des S-RAMs

Chip Select: Der Baustein hat drei *Chip Select* Leitungen, die zusammen die Befehlsdecodierung aktivieren. Damit der Baustein selektiert ist und Befehle akzeptiert, müssen $\overline{CS_1}$ und $\overline{CS_2}$ *low* sein und CS_2 als Negation von $\overline{CS_2}$ *high*. In jedem anderen Fall ist der Baustein nicht selektiert und empfängt zwar Befehle, führt sie aber nicht aus. Die drei Eingänge sind vollkommen gleichberechtigt und redundant, darum genügt es, $\overline{CS_2}$ fest auf *low* und CS_2 fest auf *high* zu legen und die Selektierung ausschließlich mit $\overline{CS_1}$ zu steuern.

Output Enable: Das $\overline{OE}$-Signal erlaubt es zu jeder Zeit, die Ausgangstreiber des RAMs auszuschalten und so die Busleitungen hochohmig zu setzen. Dies bedeutet, dass die Leitungen nicht getrieben werden, damit andere Bauteile die Leitungen mit ihren Ausgangstreibern ansprechen und benutzen können.

Write Enable, Byte Write: Ein Arbeitszyklus beginnt, wenn der Baustein selektiert ist, zu jeder steigenden Taktflanke. Ist dabei $\overline{WE}$ *high* gesetzt, beginnt ein Lesezyklus. Adressdaten werden eingelesen, und am Ende des Taktes liegen die Daten an den Ausgangstreibern an. Für den Beginn eines Schreibzyklus muss $\overline{WE}$ *low* gesetzt sein und die anliegenden Daten werden im darauf folgenden Zyklus in den Speicher übernommen. Mittels der $\overline{BW_{1,2,3,4}}$ Leitungen kann eine Maskierung angegeben werden, die bestimmt, welche der vier Bytes in einer Reihe mit den Daten des Busses überschrieben werden sollen.

Linear Burst Order, Adress Advance/Load: Über die Signale $\overline{LBO}$ und ADV kann das RAM im *Burst*-Modus benutzt werden und intern automatisch Adressen erzeugen. Dies ist für die vorliegende Anwendung nicht von Interesse, da die Adressleitungen nicht mit den Datenleitungen gemultiplext sind und es keine Einschränkung bedeutet, die Adressen zu jedem Takt von außen einzulesen.

Power Sleep: Das asynchrone Signal $\overline{ZZ}$ versetzt den Baustein zu jeder Zeit in einen energiesparenden Modus, der aber die Daten des Speichers erhält. Dieser Modus bleibt unbenutzt, da die Funktionalität des Abschaltens durch das *Chip Select* abgedeckt ist und auf eine Reduzierung der Stromaufnahme in diesem Fall verzichtet werden kann.

3.4.2 Synchron dynamisches (SD-) RAM

SD-RAM unterscheidet sich in der internen Funktionsweise grundsätzlich von S-RAM. Während in einer statischen Speicherzelle mehrere Transistoren so verschaltet sind, dass sie einen Zustand behalten, bis durch einen Treiber ein anderer Zustand erzwungen wird, unterliegt die Speicherung der Information in der Kapazität einer dynamischen RAM-Zelle einer ständigen Fluktuation. Da durch parasitäre Effekte immer Ladungen abfließen, muss in regelmäßigen Abständen eine Auffrischung der Information erfolgen. Der Vorteil von dynamischen Speicherzellen besteht in der geringeren Größe, daher kann im Allgemeinen mehr Speicherkapazität in einem Baustein untergebracht werden.

Ein Modul mit 64 Bit breitem Datenbus besteht aus 4 gleichartigen parallel geschalteten Bausteinen, denen je 16 Bit des Busses zugeordnet sind. Die Ansteuerung der gemeinsamen Adress- und Steuerleitungen ergibt sich aus den Spezifikationen der einzelnen Bausteine wie in [SDRAM] beschrieben. Zunächst ist der interne Speicher jedes Bauteils in zwei oder vier *Bänke* aufgeteilt, die getrennt voneinander angesprochen werden. Innerhalb einer Bank gibt es 2^{14} Reihen und 2^8 Spalten von 16 Bit breiten Speicherzellen. Die Adressierung eines Datums erfolgt in zwei Takten, wobei im ersten Takt die Reihe selektiert wird und im zweiten Takt die Spalte.

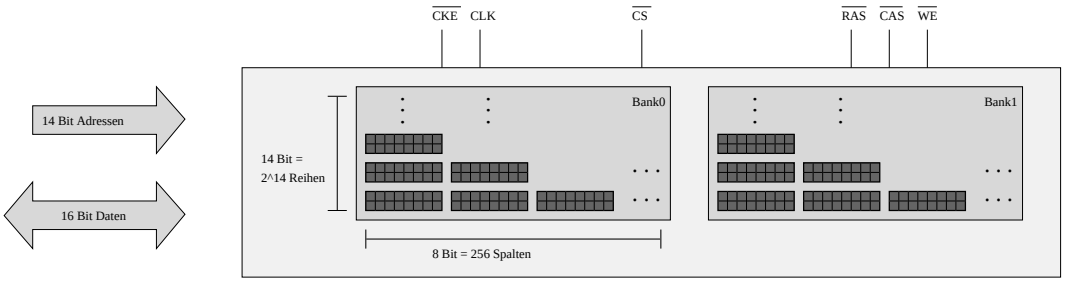


Abbildung 3.11: Speicheraufbau eines SD-RAM Chips

Nach dem Einschalten muss es eine Initialisierungszeit von 100 μs geben, in der keine Befehle angenommen werden. Danach muss die Logik als erstes im Leerlauf[17] betrieben werden, und es müssen einige *Refresh*-Zyklen erfolgen. Danach ist das SD-RAM bereit für den normalen Betrieb.

Taktung und Chip Select: Die CLK und $\overline{CKE}$-Signale sowie die einzelne $\overline{CS}$-Leitung haben beim SD-RAM dieselbe Funktion wie beim S-RAM. Bei einem RAM-

[17] Befehl NOP - no operation

Modul, bei dem 4 Bausteine die Datenbusbreite abdecken, kann bei einer Bestückung mit 8 Bauteilen über doppelte *Clock Enable* und *Chip Select* Leitungen gewählt werden, welcher der beiden Chipsätze aktiv ist. So kann die Speicherkapazität verdoppelt werden.

Row Address Select, Column Address Select, Write Enable: Die Leitungen $\overline{RAS}$, $\overline{CAS}$ und $\overline{WE}$ definieren den Befehlssatz des SD-RAMs. Sind alle drei Leitungen *high*, so bedeutet dies NOP, da die Signale *low* aktiv sind. Das SD-RAM ist nicht in jedem Takt identisch anzusprechen wie das S-RAM, sondern es gibt Befehle, die sich über mehrere Zyklen erstrecken und in der korrekten zeitlichen Abfolge angesteuert werden müssen. Durch $\overline{RAS}$ wird eine Reihe aktiviert, was notwendige Voraussetzung für jeden Speicherzugriff ist. Im nächsten Zyklus kann durch $\overline{CAS}$ eine Spalte selektiert werden und abhängig von $\overline{WE}$ ein Schreib- oder Lesezyklus erfolgen.

Neben den Speicherzugriffen gibt es beim SD-RAM auch administrative Befehle wie das Schreiben von Werten in interne Konfigurationsregister, *Precharge* oder *Refresh*. Ein *Refresh*-Zyklus bedeutet, dass eine Reihe aufgefrischt wird, also intern gelesen und neu geschrieben. Eine Auffrischung des gesamten Speichers erfordert 4096 *Refresh*-Zyklen alle 64 ms, entweder verteilt über die Zeit oder gebündelt als *Burst Refresh*, wobei das Adressschema der Auffrischung von der internen Logik erzeugt wird. Bei einer Taktung von 100 MHz dauert ein *Burst Refresh* circa 0.4 ms, und somit ist der Anteil der Zeit, der für Auffrischung nötig wird, vernachlässigbar.

3.4.3 Gemeinsame Benutzung der Busleitungen

Wie zu Beginn dieses Abschnittes erwähnt, werden möglichst viele Leitungen zum FPGA von beiden RAM-Sorten gemeinsam benutzt. Für die 64 Datenleitungen ist es selbstverständlich, dass beide RAMs byteweise den gesamten Bus abdecken müssen. Die Anzahl der Adress- und Steuerleitungen ist aber verschieden und muss sinnvoll aufgeteilt werden. Die *Chip Select* und *Clock Enable* Signale der Bausteine müssen einzeln ansprechbar sein, damit sie wechselseitig ausgeschaltet werden können und sichergestellt ist, dass die ausgeschalteten RAMs nicht die Kommunikation mit den eingeschalteten stören. Beim S-RAM muss außerdem durch eine individuelle Ansteuerung des *Output Enable* sichergestellt werden, dass die Ausgangstreiber deaktiviert werden können. Die Adressleitungen 1 bis 13 können direkt von beiden RAMs benutzt werden. Da das S-RAM aber mehr Adressleitungen benötigt als das SD-RAM, können die Adressleitungen 17 und 18 für Steuersignale des SD-RAMs benutzt werden. Tabelle 3.1 verdeutlicht diese Zuordnung.

SD-RAM	**FPGA**	**S-RAM**
$\overline{S_0}$	$SDRAM\_S0$	--
$\overline{S_1}$	$SDRAM\_S1$	--
CKE_0	$SDRAM\_CKE0$	--
CKE_1	$SDRAM\_CKE1$	--
CK_0	RAM_CLK	CLK
CK_1		

$\overline{WE}$	RAM_WE	$\overline{WE}$
$A_{0..13}$	$RAM\_A0..A13$	$A_{0..13}$
BA_0	RAM_A14	A_{14}
BA_1	RAM_A15	A_{15}
--	RAM_A16	A_{16}
$\overline{RAS}$	RAM_A17	A_{17}
$\overline{CAS}$	RAM_A18	A_{18}
--	$SRAM\_ADV$	ADV
--	$SRAM\_OE$	$\overline{OE}$
--	*low*	$\overline{CKE}$
--	$SRAM\_CS$	$\overline{CS_1}$
--	*low*	$\overline{CS_2}$
--	*high*	CS_2

Tabelle 3.1: RAM Ansteuerung

Die Verlegung dieser Verbindungen auf der Platine ist in Abbildung 3.12 gezeigt, wobei das S-RAM mit zwei Bausteinen auf der Oberseite der Platine liegt und der Sockel für das SD-RAM Modul auf der Rückseite der Platine.

In den oberen vier Zeichnungen sieht man immer die Pads der Oberseite, auf den beiden unteren die Pads der Unterseite, um die Bauteile identifizieren zu können. Die Leiterbahnen der Lage 1 werden soweit möglich direkt vom FPGA zum S-RAM gelegt, um Vias zu vermeiden. Die Leitungen an der Unterkante der RAMs sind je zwei Byte Daten. Im Zentrum gehen die Adressleitungen zu den diagonal angebrachten Vias, und laufen auf den Lagen 2 und 3 horizontal zu allen drei RAM-Bausteinen. Auf Lage 4 erkennt man, wie die Datenleitungen der anderen vier Byte an die oberen Seiten der RAMs angeschlossen sind. Charakteristisch ist, dass sowohl die obere als auch untere Reihe horizontaler Vias für die Verteilung der Daten sorgen. Die obere Reihe erhält Datenleitungen auf Lage 1 vom FPGA, die untere Reihe auf Lage 4. Nach dem Via spalten sich die Bahnen auf zum S-RAM auf der Oberseite und SD-Ram auf der Unterseite.

Da die Laufzeit des Taktsignals an die beiden S-RAM Bausteine ungefähr gleich sein sollte, wurde auf der Oberseite eine vorsätzliche Verlängerung der Leiterbahn zum rechten Baustein eingebaut. Das Via, an dem das Signal an die Bausteine verteilt wird, ist auf halber Länge zum SD-RAM angebracht, damit an dieser Stelle auch die Rückführung zur DLL[18] abzweigen kann.

3.5 CMC-Stecker

Auf der PCI-Karte sind wie bei der Platzierung in Absatz 3.1.2 erläutert insgesamt vier CMC-Stecker vorhanden. Zwei sind CMC-Standard konform, so dass kommerzielle Karten benutzt werden können, und zwei sind selbst definiert und liegen nahe am Analogteil und *IO-Space*, um direkt an den Ausgang des PCs zu kontaktieren.

[18] siehe auch Absatz 3.3.2

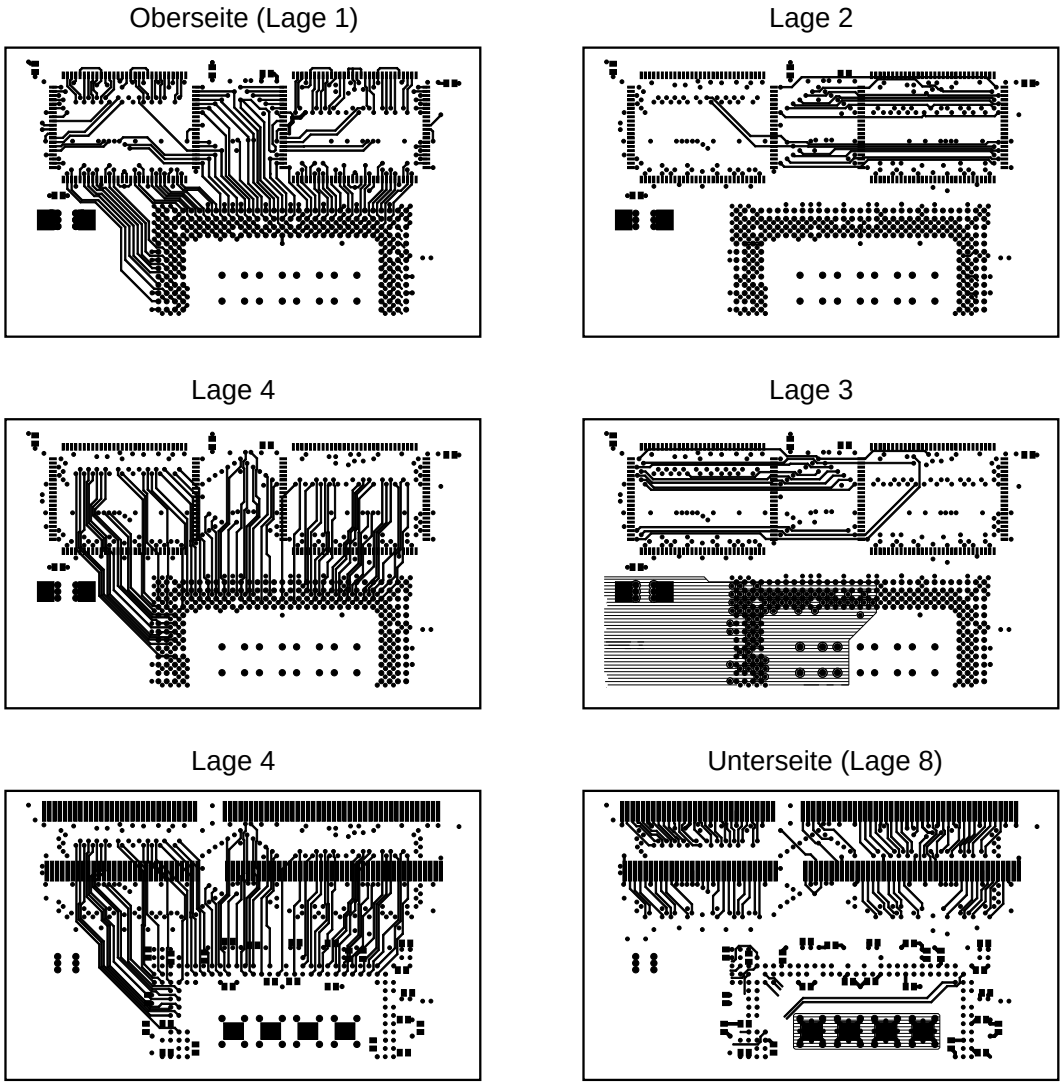


Abbildung 3.12: Routing des RAMs

3.5.1 CMC-Standard konforme Stecker

Die CMC-Stecker 1 und 2 entsprechen in der Beschaltung der Norm [IEEE:CMC]. Es gibt zwei digitale Busse mit 39 und 40 Bit und Stromversorgungen mit 3.3, 5 und 12 V. Neben den Busleitungen gibt es noch vier *Busmode* Leitungen, die der Identifizierung von Karten und Festlegung von Protokollen dienen. Um Ausgänge am FPGA zu sparen, sind die Leitungen $BUSMODE\_3$ und $BUSMODE\_4$ an Lötbrücken gelegt, da sie auch während einer Kommunikation fest gesetzt bleiben.

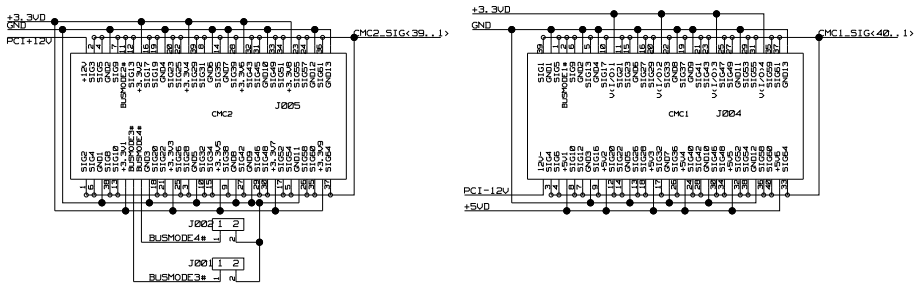


Abbildung 3.13: Belegung der CMC-Stecker 1 und 2

3.5.2 CMC-Stecker mit selbst definierter Belegung

Die CMC-Stecker 3 und 4 unterliegen keinen Normierungen, da sie an Stellen angebracht sind, an denen der CMC-Standard keine Verbindungen vorsieht. Deshalb kann die Belegung gemäss Absatz 3.3.3 frei gewählt werden. Da die beiden Stecker nicht nur für LVTTL sondern auch für LVDS-Signale benutzt werden sollen, müssen die Spezifikationen der kontrollierten Impedanz eingehalten werden. Die Nähe je zweier Leitungen, die ein LVDS-Paar bilden, ist zwar im LVTTL-Modus von Nachteil, da dann zwei unterschiedliche Signale aufeinander übersprechen können, jedoch muss dies zugunsten der guten LVDS-Eigenschaften der Leitungen in Kauf genommen werden. LVDS-Paare werden an nebeneinander liegende Anschlüsse geführt und durch Masseleitungen getrennt. An CMC-Stecker 3 liegt der externe Oszillatoreingang der PLL, um Signale von einer Tochterplatine einspeisen zu können. Außerdem werden freie Anschlüsse benutzt, um die diversen Spannungsquellen der PCI-Karte auch auf Tochterplatinen nutzen zu können. CMC-Stecker 4 hat neben dem LVDS-Bus auch eine Seite, die rein analogen Signalen vorbehalten ist. Masseanschlüsse sorgen für eine Abschirmung zu den schnellen digitalen Signalen

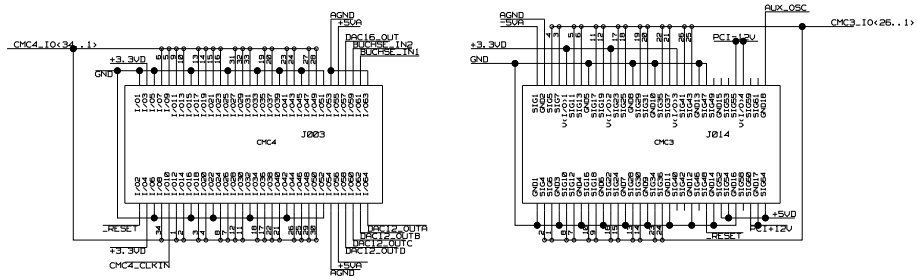


Abbildung 3.14: Belegung der CMC Stecker 3 und 4

3.6 Bauteile zur analogen Ein/Ausgabe

Auf der Analogseite der Karte finden sich neben den DACs und dem ADC auch Bauteile, die für eine gute Qualität der Analogsignale notwendig sind. Die Netzteile für analoge Versorgungsspannungen, die von den digitalen getrennt sind, befinden sich auf der Vorderseite der Platine im oberen Bereich des *IO-Spaces*. Von dort sind die Versorgungsspannungen flächig unter dem gesamten Bereich verlegt, der bis hin zu den CMC-Steckern reicht.

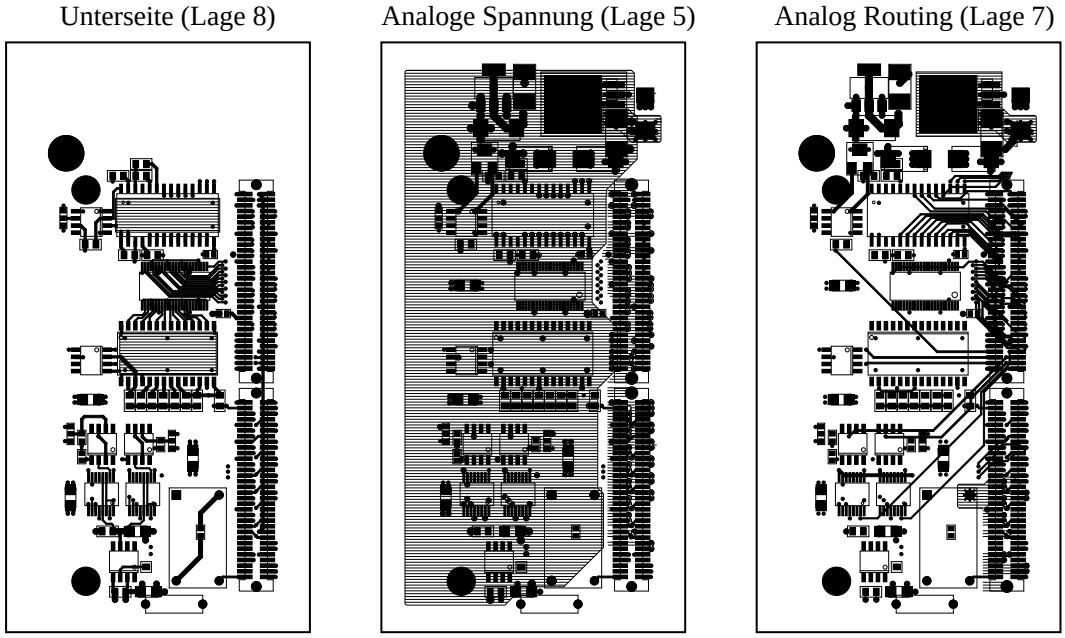


Abbildung 3.15: Analogteil von *Darkwing*

Verdeutlicht wird die Aufteilung des Analogteils durch Abbildung 3.15, in der links die Ansicht aller Bauteile auf der Unterseite zu sehen ist. Zur Orientierung sind die beiden CMC-Stecker 3 und 4 eingezeichnet. Die Mitte zeigt die flächige Verlegung der analogen +5 V-Versorgungsspannung und rechts wird das Routing der analogen Signale zum CMC-Stecker 4 und der digitalen DAC/ADC-Busse zum FPGA gezeigt.

Auf der Unterseite ist oben der 16 Bit DAC, in der Mitte der ADC mit dem Ausgangsbuffer, und im unteren Teil befinden sich die beiden 12 Bit DACs mit je einem Ausgangstreiber und einer gemeinsamen Spannungsreferenz. Ganz unten im Analogteil befindet sich eine Induktivität, die eine gefilterte Verbindung zwischen der analogen und digitalen Massefläche herstellt.

3.6.1 12 Bit-DACs

Die zwei Bausteine vom Typ [MAX5104] enthalten je zwei 12 Bit-DACs mit einer Einstellzeit von 12 μs. Die digitale Datenübertragung geschieht über ein serielles Programmierinterface mit drei Leitungen. Ist das *Chip Select* $\overline{CS}$ aktiv, werden Daten

am Anschluss DIN synchron zu einem Takt $SCLK$ durch ein Schieberegister geleitet und ausgelesen, sobald $\overline{CS}$ wieder deaktiviert wird. Das Ende des Schieberegisters ist an Anschluss $DOUT$ kontaktiert, so dass die zwei Bausteine zu einem langen Schieberegister zusammengesetzt sind und Konfigurationsdaten für alle DACs als ein langes Datenwort durch dieses Register geschoben werden. Beim Deaktivieren von $\overline{CS}$ übernehmen dann beide gleichzeitig den aktuellen Wert des Schieberegisters und interpretieren ihn als Kommando.

In dem 16 Bit langen Befehlswort jedes DACs kommen zuerst drei Bits, die den Befehl kodieren, dann 12 Bit für einen DAC-Datenwert und ein Schlussbit, das immer *low* ist. Das übermittelte Datenwort kann einem der beiden internen DACs zugeordnet werden und wird entweder sofort aktiv, wenn es ins DAC-Register geschrieben wird oder zwischengespeichert, wenn es ins Input-Register geschrieben wird. Sollen beide DACs synchron arbeiten, können erst beide Werte nacheinander in das Input-Register geschrieben werden und dann gleichzeitig die Input-Register in die DAC-Register übernommen werden.

Soll die Genauigkeit der DACs nicht schlechter sein als ein halbes LSB[19], so kann der DAC einen Widerstand von 1 kΩ und eine Kapazität von 100 pF treiben. Sollen jedoch niederohmigere Lasten versorgt werden, so muss der Ausgang durch einen externen Operationsverstärker gepuffert werden. Im Schaltplan in Abbildung 3.16 sind bei je einem der Bausteine ein Operationsverstärker an einem Ausgang angeschlossen. Die explizite Auswahl richtet sich nach der konkreten Anwendung, falls der standardmässig vorgesehene Typ LM6361 [OP] nicht genügen sollte.

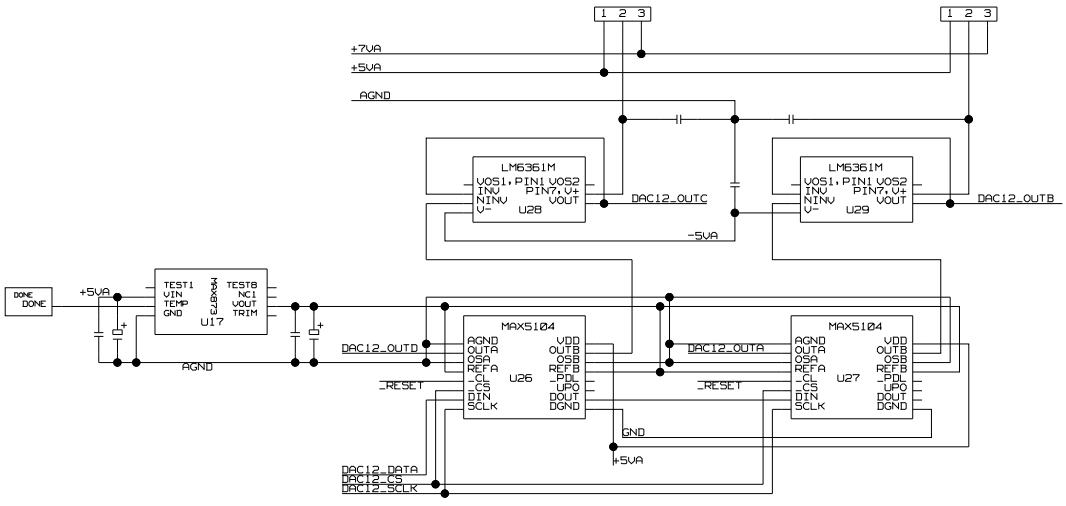


Abbildung 3.16: Beschaltung der 12 Bit DACs

Beide Bausteine benötigen eine externe Referenzspannung, die die Mitte des Regelungsbereiches zwischen der Offsetspannung am Anschluss OS und dem Ausgang OUT definiert. Hier findet ein [MAX873] Verwendung, der 2.5 V mit einer Genauigkeit von ± 1.5 mV bereitstellt. Die endgültige Kalibrierung der Ausgangs-

[19] least significant bit

spannung muss bei den 12 Bit-DACs per Software erfolgen. Durch die Messung der Ausgangsspannung bei Sollwerten kann eine Korrelation zwischen programmiertem Spannungswert und gemessener Spannung hergestellt werden.

3.6.2 16 Bit-DAC

Als 16 Bit-DAC wird ein Baustein vom Typ [AD768] benutzt, der eine Einstellzeit von 25 ns besitzt. Die Ausnutzung der höheren Genauigkeit stellt höhere Ansprüche an das Layout und die Qualität der analogen Versorgungsspannungen, als die 12 Bit-DACs. Der AD768 hat auf der einen Seite einen Analogteil, dessen Genauigkeit möglichst gut ausgenutzt werden soll, und auf der anderen Seite einen Digitalteil, der 16 parallele Datenleitungen treibt und damit hochfrequente Störungen erzeugt. Aus diesem Grund ist eine Massefläche unter den Baustein auf der Lage 8 gelegt, die die digitalen Leitungen auf Lage 7 abschirmt. Die Leitungen liegen eng nebeneinander und gehen horizontal sofort in den Digitalteil, um Störungen der Analogspannungen zu minimieren.

Der AD768 hat auf der Digitalseite neben den 16 Dateneingängen nur noch einen Takteingang, der mit einer steigenden Flanke definiert, wann die Daten vom Bus gelesen werden. 10 ns nach Gültigkeit der Daten kann der Takt aktiviert werden und nach weiteren 35 ns hat die Ausgangsspannung den eingestellten Wert angenommen. Die minimale Wartezeit des Taktsignals ist mit 10 ns also wesentlich kürzer als die Verarbeitungszeit im DAC. Falls die Spannungsdifferenzen von einem zum nächsten Ausgangswert nicht zu groß sind und eine mittlere Regelzeit etwas kleiner als der absolute Maximalwert angenommen wird, kann eine maximale Arbeitsfrequenz von 30 MHz erreicht werden.

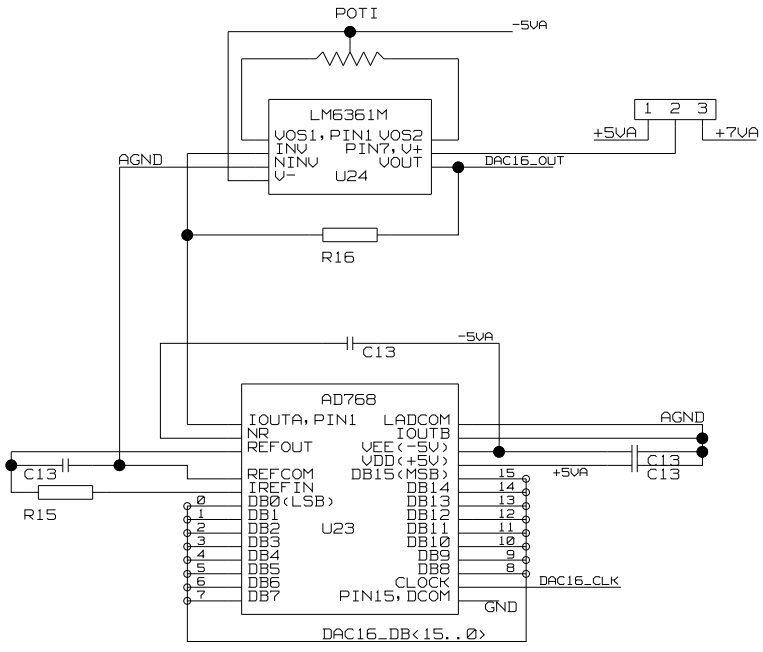


Abbildung 3.17: Beschaltung des 16 Bit-DACs

Da der AD768 keinen direkten Spannungsausgang hat, sondern den digitalen Wert in einen Strom konvertiert, muss die Spannung extern erzeugt werden. Dazu wird ein Operationsverstärker benutzt, der über einen Referenzwiderstand (R16) rückgekoppelt ist. Dies zeigt Abbildung 3.17. Am Ausgang des Operationsverstärkers stellt sich so die Spannung ein, die die Spannungsdifferenz zwischen den Eingängen minimiert. Für einen Spannungsbereich von 4 V wird ein 200 Ω Widerstand verwendet, und der absolute Offset kann durch einen regelbaren Widerstand an den Offsetanschlüssen des Operationsverstärkers reguliert werden.

Die Referenz wird hier nicht extern erzeugt, sondern der Baustein hat eine 2.5 V-Referenzspannungsquelle eingebaut. Da der DAC aber für einen Stromausgang ausgelegt ist, wird ein Referenzstrom von 5 mA benutzt, der über den externen Widerstand (R15) von 500 Ω erzeugt wird und wieder zurück in den Baustein fließt. Dies legt den Maximalwert der Ausgangsspannung fest.

3.6.3 ADC

Als ADC wird ein Baustein der ADS800 Reihe benutzt [ADC]. Je nach benötigter Geschwindigkeit kann ein ADS800 mit 40 MHz, ADS801 mit 25 MHz oder ADS802 mit 10 MHz Abtastrate eingesetzt werden, die ansonsten aber identisch sind. Diese haben eine Auflösung von 12 Bit und ein paralleles Interface. Um einem Bereich von 0.25 bis 4.25 V abzudecken, muss der Baustein 5 V Versorgungsspannung haben, und die Ausgangslogik ist TTL. Um die LVTTL-Eingänge des FPGAs nicht zu beschädigen, muss daher ein Buffer zwischengeschaltet werden, der die Signale in LVTTL konvertieren kann. Dies dient nicht nur der Spannungsanpassung, sondern wirkt sich auch positiv auf die analoge Qualität des ADCs aus, da die direkten Ausgangsleitungen kurz sind und das Treiben der Leitungen auf die lange Distanz zum FPGA durch einen Buffer erledigt wird.

Um die hohe Abtastrate zu ermöglichen, arbeitet der ADC intern mit einer Pipeline-Architektur, das heißt, die Konvertierung erfolgt stufenweise. Der Baustein bekommt eine kontinuierliche Taktung, und es dauert sechs Zyklen, bis eine Konvertierung abgeschlossen ist. Jede Stufe gibt nach einem Zyklus das Zwischenergebnis an die nächste Stufe weiter und kann im folgenden Zyklus neue Daten bearbeiten. Damit kann in jedem Zyklus jede Stufe arbeiten, und nach jeweils sechs Zyklen liegen die Daten am Ausgang an.

Die Beschaltung des ADCs zeigt Abbildung 3.18. Es gibt eine Analogseite, auf der alle Spannungszuführungen mit Keramikkondensatoren geglättet sind. Dem Eingang am Anschluss IN und den Referenzspannungen top $REFT$, bottom $REFB$ und common mode CM sind ebenfalls Keramikkondensatoren parallelgeschaltet, um Rauschen zu unterdrücken. Da nur ein Eingangspotential gegen Masse abgetastet werden soll, ist der invertierte Eingang $\overline{IN}$ an CM verbunden, um wie in [ADC] beschrieben den Eingangsbereich des ADCs voll auszuschöpfen.

Durch einen analogen Schalter [MAX4544] am Eingang des ADCs kann zwischen zwei Eingangsleitungen umgeschaltet werden. Dies ermöglicht es, ohne manuelles Umstecken von Verbindungen wahlweise den einen oder anderen Kanal zu digita-

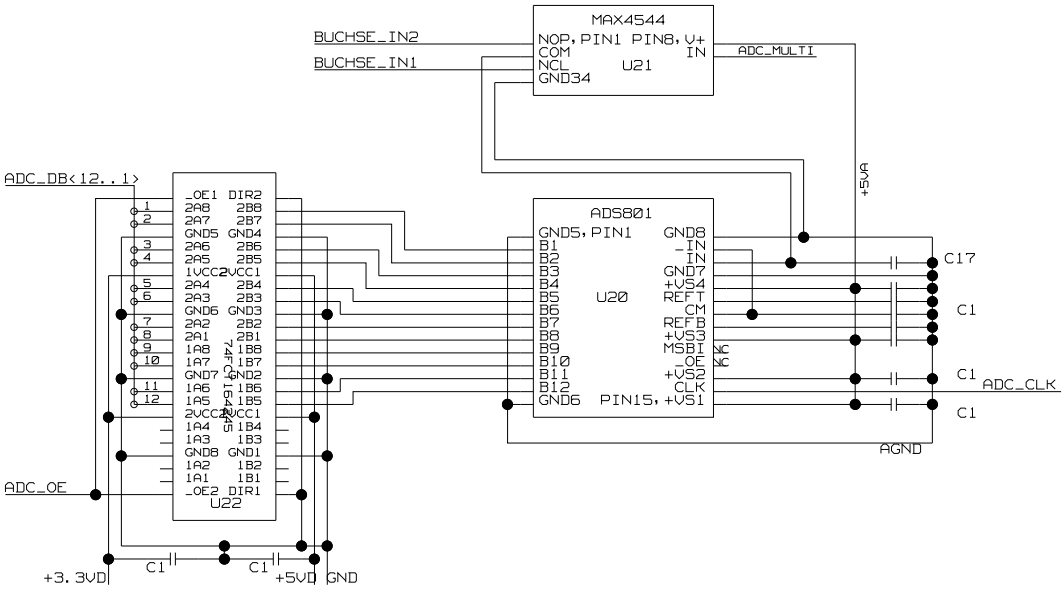


Abbildung 3.18: Beschaltung des ADCs

lisieren. Mit einer Schaltzeit von 30 ns ist der Schalter zwar zu langsam, um beim Multiplexen der zwei Eingänge an einen ADC die volle Abtastrate des ADCs auszuschöpfen, jedoch ist die Überwachung beider Kanäle mit geringer Bandbreite möglich, und die volle Geschwindigkeit steht bei Bedarf einem Kanal voll zur Verfügung. Der Baustein ist so gewählt, dass er mit einer hohen Übertragungsbandbreite keinen Einfluss auf das spektrale Aussehen des Signales hat.

Der bidirektionale 5 V zu 3.3 V übersetzende Treiberbaustein [FCT164245] wird hier ausschliesslich unidirektional benutzt, da nur die Ausgangsdaten vom ADC zum FPGA gebuffert werden. Die richtungsdefinierenden Signale DIR sind daher fest auf Masse gelegt, und mit dem *Output Enable* $\overline{OE}$ kann das Treiben der Leitungen gesteuert werden. Diese Funktion ist essentiell, wenn bei der Konfiguration des FPGAs alle IO-Treiber auf *high* gelegt werden. Es stellt sicher, dass die zwischen ADC und CPLD geteilten Konfigurationsleitungen vom Buffer nicht getrieben werden, und die Kommunikation zwischen CPLD und FPGA nicht gestört wird. Im Betrieb mit dem ADC ist der Buffer schnell genug, um die gewünschten Übertragungsraten zu leisten.

3.7 Spannungsversorgung

Die Grundlage für das Funktionieren der zu entwickelnden Schaltung bildet die korrekte Stromversorgung, die in der funktionalen Betrachtungsweise bisher noch nicht erwähnt wurde. Bestandteil der PCI-Norm ist die Bereitstellung von 5 V und 3.3 V, den Spannungspegeln für TTL und LVTTL, mit bis zu 25 Watt Gesamtleistung, aus denen die Stromversorgung für die gesamte Karte bezogen wird. Außerdem gibt es 12 V bei 0.5 A und -12 V bei 0.1 A für besondere Anwendungen. Alle anderen Spannungen müssen lokal durch Netzteile erzeugt und verlust- und störungsarm zu den Bauteilen geführt werden. Dies sind 1.8 V und 2.5 V für den FPGA und sowohl +5 V als auch -5 V für die analogen Baugruppen.

3.7.1 Grundlagen

Der Stromverbrauch digitaler CMOS[20]-Bauteile ist stark abhängig von der Anzahl der aktiven internen Komponenten. Bei hoher Aktivität wechseln die internen Komponenten häufig die Zustände und es wird in kurzer Zeit eine große Anzahl von Ladungen benötigt, um die Eingangskapazitäten der Transistoren und Leitungen umzuladen. Dabei kommt es zu großen Stromschwankungen, die von der Anzahl der Schaltvorgänge abhängen, und bei Bausteinen mit vielen Transistoren und hohen Taktfrequenzen sind besondere Vorkehrungen zu treffen, um diese Stromspitzen abzufangen.

Die Stromzuführung zu den Bausteinen erfolgt im Allgemeinen über die entsprechende Versorgungsspannungsfläche im Inneren der Platine. Dazu wird ein Via benötigt, das einen Kontakt von der inneren Lage an die Oberfläche herstellt und an eine Leitung zum Bauteil angeschlossen ist. Dieser Übertragungsweg ist nicht ideal, sondern hat neben einem Widerstand auch eine Induktivität und Kapazität pro Leitungslänge, die von der Geometrie des Leiters abhängt. Flächige Leiter besitzen eine niedrige Induktivität und eine vergleichsweise hohe Kapazität, während eine schmale Leiterbahn eine geringe Kapazität und eine hohe Induktivität hat.

Die Induktivität der Leitung kann dazu führen, dass lange dünne Leiterbahnen zu einem Baustein nicht genügend schnell die entsprechende Stromänderung aufbringen, die zum Umladen der internen Logik benötigt wird, und dies reduziert die Spannung am Baustein. Deshalb sind der Spannungszuführung Kondensatoren parallel geschaltet, die ausreichend lokale Ladungen bereit stellen, um schnell auf Stromspitzen zu reagieren. In direkter Nähe zu den Spannungsanschlüssen der Bausteine sind Keramikkondensatoren angebracht, die die hochfrequenten Stromschwankungen puffern. [Xil:Power] empfiehlt die Verwendung von Tantalkondensatoren, um die Ladeströme für die Keramikkondensatoren jeweils im Umkreis von 3 cm zur Verfügung zu stellen. Einige Tantalkondensatoren großer Kapazität, die in der Platzierung unkritisch sind, gleichen schließlich globale Stromschwankungen aus.

[20] complementary metal oxide semiconductor

3.7.2 Spannungsversorgung der LVDS-Treiber des FPGAs

In einem FPGA sind die Versorgungsspannungen für die interne Logik und die Ausgangstreiber getrennt, denn die Spannungsversorgung der IO-Treiber hängt von der Art der zu treibenden Logik ab. Für die meisten Komponenten wird LVTTL mit 3.3 V benutzt, während LVDS-Treiber eine Spannung von 2.5 V benötigen [Xil:Data]. 2.5 V sind nicht am PCI-Bus verfügbar und müssen aus 3.3 V erzeugt werden. Der FPGA soll zwei IO-Bänke mit LVDS betreiben, zur Versorgung genügt nach [Xil:Est] ein Netzteil, das auf 0.5 A ausgelegt ist.

Für den Fall kleiner Differenz zwischen Eingangs- und Ausgangsspannung und kleiner Verlustleistung wie hier sind Linearregler geeignet. Die Funktionsweise beruht auf einem Operationsverstärker als Regler, der die Differenz zwischen einer Referenz und der Ausgangsspannung zur Ansteuerung eines Transistors benutzt. Durch die Beschaltung als *Emitterfolger* fließt genau soviel Strom durch den Transistor, wie nötig ist, um die geforderte Spannung über einer Ausgangslast aufrecht zu erhalten. Dabei kann die rückgekoppelte Spannung zum Operationsverstärker durch einen Spannungsteiler justiert und somit die Ausgangsspannung geregelt werden.

Die Benutzung eines Transistors als Stellglied bedeutet, dass der Widerstand des Transistors so geregelt wird, dass der Spannungsabfall zwischen Ein- und Ausgangsspannung über dem Transistor liegt. Zur Aufrechterhaltung der Regelung muss dieser Spannungsabfall einige Volt betragen, bei modernen *low drop*-Reglern einige hundert mV. Die Verlustleistung berechnet sich als Produkt aus Spannungsabfall und Ausgangsstrom und wird voll vom Transistor in Wärme umgewandelt. Aus diesem Grund muss eine ausreichende Wärmeabfuhr über das Gehäuse gewährleistet werden, die zumeist den limitierenden Faktor bei der Benutzung eines Linearreglers darstellt.

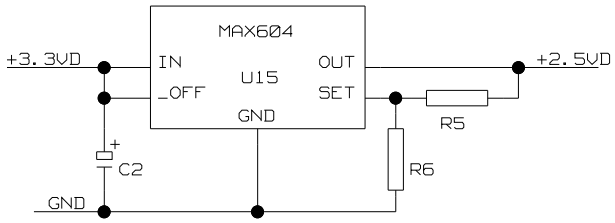


Abbildung 3.19: Linearregler MAX604

Der hier benutzte Regler [MAX604] vergleicht die Rückkopplung gegen 1.2 V und so benötigt man einen Spannungsteiler (R5, R6 in Abbildung 3.19), um die Ausgangsspannung auf die gewünschten 2.5 V einzustellen. Die maximale Verlustleistung beträgt für dieses Netzteil $(3.3 - 2.5)\,V \cdot 0.5\,A = 0.4\,W$, die als Wärme über das SO-8 Gehäuse und die Platine abgeführt wird.

3.7.3 Spannungsversorgung des FPGAs

Die Leistungsaufnahme der internen Logik eines FPGAs [Xil:Power] lässt sich abschätzen durch die Formel $P = C \cdot U^2 \cdot f$, die die Leistung P als das Umladen der

internen Komponenten mit der Gesamtkapazität C bei der Spannung U und Wechselfrequenz f darstellt. Da die Spannung quadratisch eingeht, versuchen Hersteller, die Versorgungsspannung weiter zu senken. Andererseits ist man bestrebt, die Logik mit immer höheren Taktfrequenzen laufen zu lassen, was die Leistungsaufnahme erhöht. Die Anschaulichkeit dieser Formel darf nicht darüber hinweg täuschen, dass die Kapazität und Anzahl der internen Komponenten nicht pauschalisierbar sind und die Leistungsaufnahme außerdem von der Programmierung und der Platzierung der Logik im FPGA abhängt. Daher greift man auf heuristische Methoden zurück, um eine Abschätzung der Leistungsaufnahme zu erhalten. Im Falle der hier verwendeten FPGAs der Virtex-E Reihe[21] erhält man eine Abschätzung durch den *Virtex Power Estimator*. [Xil:Est]

Für die Abschätzung der maximalen Belastung eines FPGAs vom Typ XCV300E im hier beschriebenen Testsystem gingen als Eckdaten ein:

- S- und SD-RAM Ansteuerung: 100 - 130 MHz, 40% der CLBs
- LVDS: 100 MHz, 40% der CLBs
- Local Bus und interne Register: 40 MHz, 20% der CLBs

Diese Werte für ein maximal großes Design bei hoher Geschwindigkeit sind weit über der Normalbelastung, aber die Realisierung selbst einer solchen Konfiguration soll nicht an der Stromzufuhr scheitern müssen. Dies führt zu einer geschätzten maximalen Leistungsaufnahme von 16 W für die interne Versorgungsspannung.

Um die Anforderungen gemäß [Xil:Power] und [Xil:Data] zu erfüllen, wird ein Netzteil benötigt, das einen maximalen Ausgangsstrom von 9 A bei 1.8 V liefert. Als Eingangsspannung dienen die 5 V, die vom PCI-Bus mit ausreichender Leistung zur Verfügung gestellt werden. Der Fehler der Ausgangsspannung soll selbst bei einem schnellen großen Lastwechsel jederzeit unter 5% liegen und die Verlustleistung gering sein. Diese Bedingungen würden bei einem Linearregler zu einer Verlustleistung von ca. 19 W führen und erfordern daher die Benutzung eines Schaltregler-Netzteils, dessen Funktionsweise im folgenden erläutert wird.

Funktionsprinzip

Die prinzipielle Arbeitsweise eines Schaltreglers basiert auf der kontrollierten Energiespeicherung in einer Spule. Mit einer Frequenz von einigen hundert kHz wird über Schalter die Energieaufnahme und -Abgabe einer Spule von einem Regelbaustein gesteuert. Die elektrischen Eigenschaften einer idealen langen Spule lassen sich aus den Maxwell-Gleichungen herleiten und durch die folgenden Formeln beschreiben:

$$B = \mu_0 I n \qquad\qquad W = \frac{1}{2} I^2 L \tag{3.3}$$

B ist die Stärke des induzierten Magnetfeldes bei fester Windungszahl n, Permeabilitätskonstante μ_0 und Strom I durch die Spule. Die gespeicherte Energie W ist proportional zur Induktivität L und dem Quadrat des Stromes I. Durch einen hohen

[21] XCV300-E, XCV400-E und XCV600-E

Stromfluss kann man viel Energie in der Spule speichern, auf der anderen Seite darf das Magnetfeld einen kritischen Punkt nicht überschreiten, nämlich den Sättigungswert des ferromagnetischen Spulenkerns.

Durch eine Änderung des Stromflusses in der Spule wird eine Spannung induziert, die nach der Lenzschen Regel der Ursache entgegen gerichtet ist. Idealisiert wird eine Spule dargestellt durch eine reine Induktivität und einen Serienwiderstand. Schaltet man eine Spannung U_0 über dieser Spule ein, so ist der Spannungsabfall in der Spule aufgeteilt in U, das über dem Widerstand R abfällt und U_{ind}, das den Induktionsterm darstellt. Diese Spannung U entspricht einem Ohmschen Widerstand R bei gegebenen Strom I.

$$U = U_0 + U_{ind} = U_0 - L\dot{I} = RI \tag{3.4}$$

Diese Differentialgleichung beschreibt die dynamischen Eigenschaften und die Lösung ergibt einen exponentiellen Stromanstieg beim Einschalten, der beschrieben wird durch

$$Einschalten : I = \frac{U_0}{R}(1 - e^{-t\frac{R}{L}}) \tag{3.5}$$

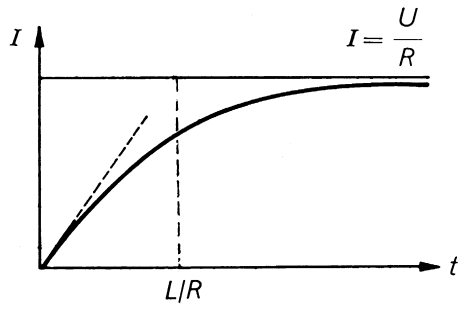


Abbildung 3.20: Lösung der Differentialgleichung (3.4) für das Einschalten

Eine schnelle Stromänderung erfolgt wie in Abbildung 3.20 dargestellt im Bereich $t < \frac{L}{R}$, wo der Strom noch fast linear mit der Zeit steigt und noch nicht gegen den festen Endwert $\frac{U}{R}$ konvergiert.

Um die Eigenschaften der Spule für ein Netzteil zu nutzen, geht man wie folgt vor: Die Spule ist auf der einen Seite durch zwei Schalter entweder gegen die Eingangsspannung oder Masse geschaltet und auf der anderen Seite an die Ausgangskondensatoren angeschlossen. Schließt man den oberen Schalter in Abbildung 3.21, beginnt ein Stromfluss zum Laden der Ausgangskondensatoren, der durch die Spule hindurch geht. Die Induktivität begrenzt den Ladestrom, wobei der Schaltregler einen Sonderfall der allgemeinen Spulendifferentialgleichung darstellt. Da die Ein- und Ausgangsspannung als konstant anzunehmen sind, muss auch der Induktionsterm in Formel 3.4 konstant sein. Daher ist auch $\dot{I}$ konstant und der Strom durch die Spule steigt oder fällt immer linear. Die Induktionsspannung der Spule ist dabei gleich dem Spannungsabfall zwischen Ein- und Ausgangsspannung.

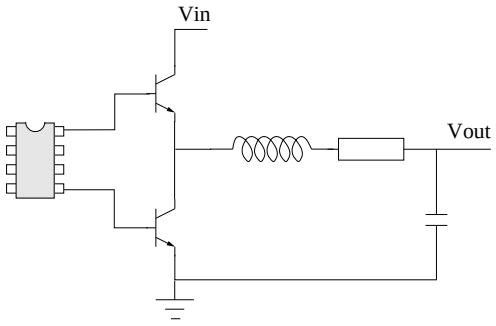


Abbildung 3.21: Prinzipieller Aufbau eines Schaltreglers

Ist der maximale Induktionsstrom erreicht, der über einen Messwiderstand ermittelt wird, wird der Eingangsschalter wieder geöffnet und die Spule durch den unteren Schalter zwischen Masse und Ausgang kontaktiert. Dabei verändert sich die Polung der Spule, und da die Spule versucht, der Stromänderung entgegen zu wirken, wird Energie aus dem Magnetfeld abgegeben. Dabei bleibt die Stromrichtung bestehen und die Ausgangskondensatoren werden weiter geladen. Die Speicherung der Energie im ersten Zyklus und Abgabe im zweiten ermöglicht einen hohen Wirkungsgrad. Bevor die Spule ihre Energie aufgebraucht hat und beginnen würde, Strom aus den Ausgangskondensatoren zu entnehmen, werden die Schalter wieder geöffnet und ein neuer Zyklus kann beginnen.

Auswahl der Komponenten

Ein Regelbaustein für die beschriebene Anwendung ist der Typ LTC1735, der die Ausgangsspannung mit einem Spannungsteiler und den Spulenstrom über einem Messwiderstand[22] misst, um die Ansteuerung der Spule zu optimieren. Als elektronische Schalter dienen MOSFETs.

Die Leistung des Netzteils ist prinzipiell nur von den Parametern der Komponenten abhängig: der Induktivität und Maximallast der Spule, der maximalen Strombelastbarkeit der MOSFETs, dem seriellen Scheinwiderstand (ESR[23]) der Kondensatoren und natürlich dem Platinenlayout, da die Leiterbahnen diesen hohen Strömen standhalten müssen.

Im Hinblick auf die maximale Ausgangslast von 8 bis 10 A und die kurzen Schaltzeiten fiel die Wahl der MOSFETs auf den Typ Si4886DY [Si4886], der die Ausgangslast selbst als Dauerstrom schalten könnte[24]. Der geringe Widerstand von ca. 10 mΩ im geschalteten Zustand zwischen Drain und Source wird für die hohen Ströme ebenso benötigt wie eine Schaltzeit im Bereich von 0.1 μs für die hohe Frequenz des Reglers.

Für die Wahl der Ausgangskondensatoren ist vor allem der ESR von Bedeutung. Das einfachste Ersatzschaltbild für einen realen Kondensator ist die Kapazität in

[22] engl.: *sense resistor*
[23] equivalent/effective series resistance
[24] Die mittlere Belastung ist wesentlich kleiner als der gepulste Maximalstrom

Serie mit einem Widerstand. Benutzt man Kondensatoren zur Glättung der Ausgangsspannung, wird Spannungsrauschen zu Strom durch den Kondensator. Der ESR limitiert damit nach dem Ohmschen Gesetz den Strom, mit dem ein Kondensator Ladung aufnehmen oder abgeben kann, also die Geschwindigkeit, mit der er auf Spannungsänderungen reagiert.

Abschließend ist zu bemerken, dass mit der heutigen Technologie Kondensatoren mit entsprechendem ESR und Kapazität nicht ohne weiteres verfügbar sind. Tantalkondensatoren haben eine ausreichende Kapazität, aber einen ESR von 50 bis 100 mΩ. Keramikkondensatoren besitzen einen wesentlich geringeren ESR, aber auch eine geringere Kapazität. Die Anforderungen an die Ausgangskondensatoren nach [LTC1735] sind $ESR < 2.2 \cdot R_{sense}$ und $C > 1/(8fR_{sense})$. Daher müssen Tantalkondensatoren mit kleinem ESR benutzt werden.

Design

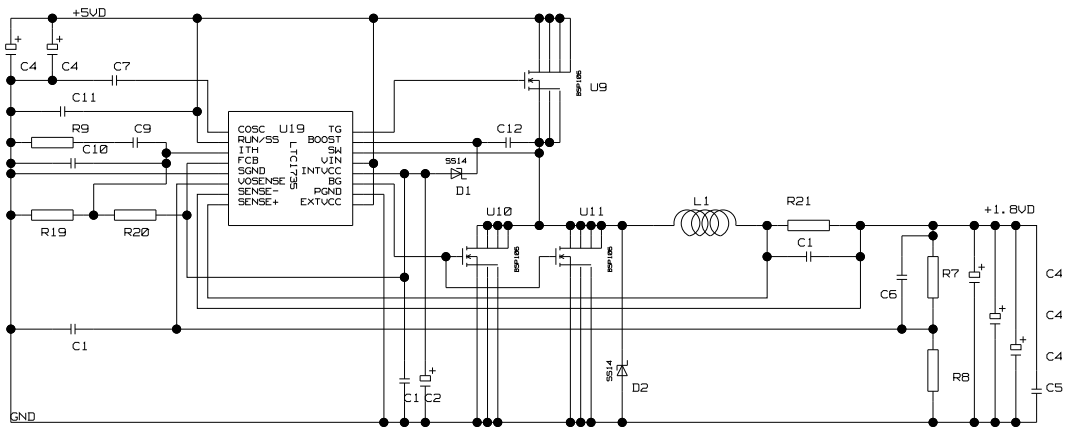


Abbildung 3.22: Schaltplan des 1.8 V Reglers

Strompfad Im Schaltplan dargestellt in Abbildung 3.22 ist der zentrale Teil der Schaltregler. Auf der rechten Seite sieht man den schon beschriebenen Hochstrom-Pfad mit den MOSFETs, der Spule und den Ausgangskondensatoren. Für die Messung der Ausgangsparameter gibt es einen Spannungsteiler und einen Messwiderstand R_{sense} (R21).

Ein normaler Arbeitszyklus des Reglers beginnt wie beschrieben mit dem Schließen des MOSFETs U9. Der Kondensator C12 wurde durch Anheben der Leitung *SW* vorgeladen und ist mit der 100-fachen Kapazität des MOSFET-Gates groß genug, um den oberen MOSFET zu *boosten*. Das heisst, der Regler schaltet die Spannung über dem Kondensator zur internen Schaltspannung hinzu und kann den MOSFET mit fast doppelter Spannung betreiben und somit den Gate-Drain-Widerstand verringern, um hohe Stromdurchlässigkeit zu erreichen.

Eine Besonderheit des Designs ist der Messwiderstand R_{sense}, über dem der Spulenstrom gemessen wird. Er ist so zu wählen, dass auf der differentiellen Messleitung

bei maximalem Ausgangsstrom eine Spannung von 50 mV anliegt [LTC1735]; dies entspricht einem Widerstand von 6 mΩ bei 9 A. In der Literatur findet man Beschreibungen, wie ein solch kleiner Widerstandswert bei so hoher Last mit hinreichender Genauigkeit zu erreichen ist. Es ist möglich, Messwiderstände als Leiterbahnen zu verwirklichen, was zu einer guten Genauigkeit führt aber wegen des hohen Stromes in diesem Fall zu viel Platz benötigen würde. Die tatsächliche Realisierung erfolgt über eine Drahtbrücke aus Silberdraht, deren Widerstand man durch die Länge des Drahtes regulieren kann. [LTC1735] schlägt sogar vor, dass man jedes Netzteil individuell an der bestückten Platine ausmisst und den Wert von R_{sense} kalibriert.

Analogteil Auf der linken Seite des Schaltplans 3.22 befinden sich die Bauteile, die nicht im Nutzstrompfad liegen und ausschließlich der Einstellung der Regelung dienen.

C7 ist ein Kondensator, der durch seine Entladecharakteristik die Oszillatorfrequenz des Reglers festlegt. Eine Kapazität von 30 bis 50 pF entspricht einer Frequenz zwischen 100 und 550 kHz.

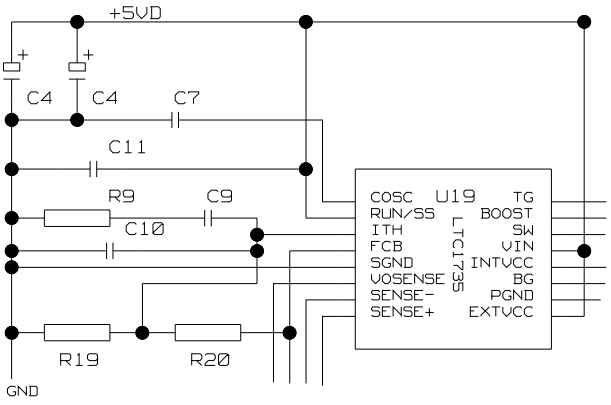


Abbildung 3.23: Analogteil des Reglers

C11 wird als Kurzzeittimer benutzt, um den Pin RUN/SoftStart zu beschalten. Wird RUN/SS auf Masse gezogen, schaltet der Regler aus; überschreitet die Spannung an RUN/SS eine Schwelle von 1.5 V, so schaltet der Regler wieder ein. Durch eine interne Stromquelle wird C11 geladen, was dazu führt, dass der Regler erst startet, wenn ausreichend Zeit bestand, um den Kondensator über die Schwellenspannung hinaus zu laden[25] (Verzögerungszeit ca. $1.25\,\mathrm{s}/1\,\mu F$).

Die Beschaltung am Pin I_{TH}, dem Kompensationspunkt der internen Überstrom-Fehlererkennung, bestimmt mit R9, C9 und C10 die Charakteristik des Abschaltemechanismus. Hierdurch wird festgelegt, bei welchem Ausgangsstrom ein Zyklus beendet wird und wie bald nach dem Ende eines Zyklus ein neuer beginnen kann. Zusammen mit Pin FCB[26], der verschiedene Betriebsmodi im Bereich kleiner Lasten auswählt, kann so das Verhalten des Regelbausteins kontrolliert werden.

[25] daher der Name SoftStart
[26] forced continous / burst

R19 und R20 stellen einen Spannungsteiler dar, mit dem die Vorspannung an Pin I_{TH} eingestellt und der Betriebsmodus *active voltage positioning* gewählt werden kann. Dies ist eine Anpassung des Reglers für Prozessor-Stromversorgungen. Dabei wird davon ausgegangen, dass der Prozessor direkt vor sich weitere Puffer-Kondensatoren hat und schnell zwischen großem und kleinem Stromverbrauch wechselt, weil er je nach Programmierung gerade viele oder wenige Transistoren schalten muß. Diese Lastwechsel sind unter Umständen schneller, als die Regelschleife darauf antworten kann, denn der Regler braucht einige Zyklen, bis er die Ansteuerung der Spule wieder auf die richtige Ausgangslast angepasst hat. Würde der Regler die Spannung konstant auf 1.8 V halten, würde ein schneller Lastanstieg dazu führen, dass die Pufferkondensatoren den notwendigen Strom liefern und dabei an Spannung verlieren, bis der Regler wieder genügend Ladung bringt. Umgekehrt würde bei einem Rückgang des benötigten Stromes der Regler zu lange auf hoher Leistung arbeiten und die Kondensatoren zu hoch laden. Um dies zu vermeiden, nutzt der Regler die Toleranz der Eingangsspannung des Prozessors von typischerweise 0.1 V aus. Bei Vollast liefert der Regler eine niedrigere Ausgangsspannung, damit ein Übersteuern der Spannung beim Abfallen der Ausgangslast eine höhere Toleranz nach oben hat, bis der Regler wieder stabilisiert ist. Analog dazu liefert der Regler bei kleiner Last eine höhere Spannung, damit die Toleranz für das Untersteuern im Falle eines Lastanstieges ausgenutzt werden kann.

Die Eingangskondensatoren sollen eine ausreichende Menge lokaler Ladungen zum Energieübertrag in die Speicherspule zur Verfügung stellen. Der ESR sollte klein sein, damit die Kondensatoren genügend hohe Ströme zur Spule bereitstellen können, dies leisten schon zwei parallelgeschaltete Tantalkondensatoren.

Eine genaue Bestückung für die angestrebten Leistungsdaten ist aus dem Bestückungsplan im Anhang D.1 ersichtlich. Die Reglerfrequenz beträgt mit $C7 = 43\,pF$ ca. 280 kHz. Damit bleibt der Regler nach Formel 3.4 mit der Einschaltzeit des oberen MOSFETs unter der Zeitkonstante $L/R \hat{=} 5\,\mu s$ der Spule und somit im linearen Bereich der Stromkennlinie.

Layout

Auf einige Besonderheiten im Layout wurde bereits hingewiesen: Die Strompfade vom Eingang über den oberen MOSFET durch die Spule zum Ausgang und von Masse über die unteren MOSFETs durch die Spule zum Ausgang müssen breit genug sein, um die Ströme ohne zu großen Widerstand und Erwärmung leiten zu können. Dazu werden die Leistungsbauteile möglichst nahe zusammen gesetzt und durch breite Leiterbahnen und eine große Anzahl von Vias verbunden. Dies verdeutlicht Abbildung 3.24.

Die Masse des Strompfades wird mit einigen Störungen verunreinigt werden, da die MOSFETs die hohen Ströme mit hoher Frequenz vom einen auf den anderen Pfad schalten. Damit diese Schwankungen des Massepotentiales möglichst wenig andere Bauteile beeinflussen, wurde darauf geachtet, dass lokale Rückflußpfade für den Strom vorhanden sind. Das heißt, dass der Stromfluß nicht über einen langen Pfad geführt ist, entlang dessen andere Bauteile liegen, die ein ungestörtes Massepotential

benötigen, sondern dass die Störungen auf den kurzen Weg des jeweiligen Pfades beschränkt sind und der jeweilige Stromkreis geschlossen ist. Dies begrenzt Störungen und Schwankungen des Massepotentiales auf lokale Bereiche, denn zu anderen Baugruppen fließen dann keine Ströme eines Hochstrompfades sondern nur kleine Ausgleichsströme.

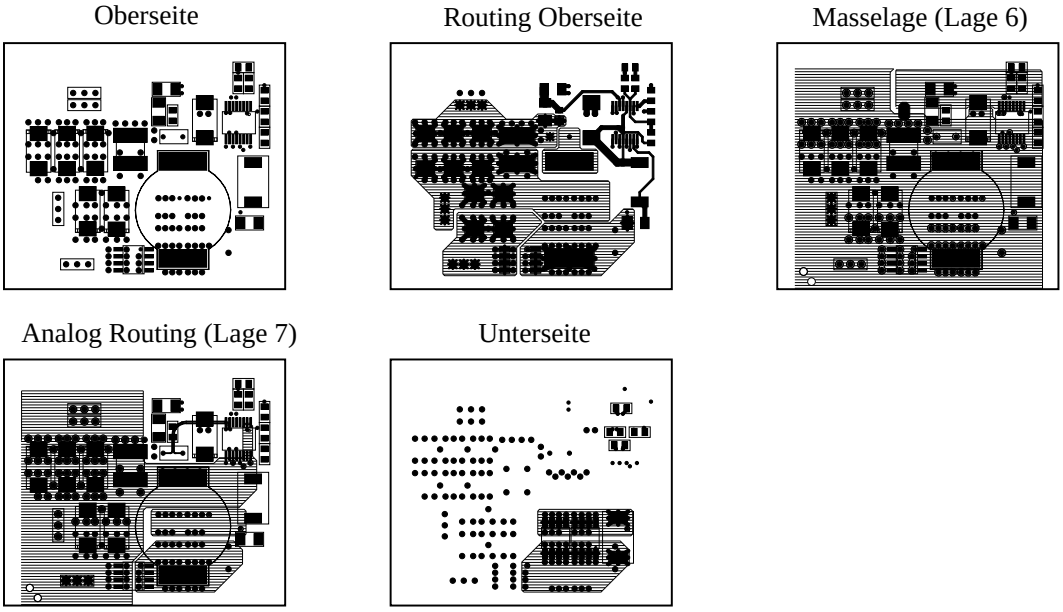


Abbildung 3.24: Layout des 1.8 V Netzteils

Dies ist der Grund, warum die Massefläche unter dem Analogteil des Reglers geteilt ist. Eine schmale Verbindung für die kleinen Ausgleichsströme hat eine Impedanz und wirkt somit als Tiefpass. Damit wird die Masse als Bezugspunkt für Regelschaltungen sauberer als die digitale Masse, an der durch das schnelle Schalten viel Rauschen zu erwarten ist. Außerdem gibt eine einzige Verbindung die Möglichkeit, einen definierten Punkt zu bestimmen, auf dessen Potential die Analogmasse liegen soll. Dieser Abgriff ist in der Nähe der MOSFETs an Masse denkbar schlecht, da hier das größte Rauschen zu erwarten ist. Der sinnvolle Berührungspunkt von Analogteil und Hochstromteil ist der Spannungsteiler, da hier die Messung der Ausgangsspannung für die Regelung gemacht wird und alle anderen Regelkreise sich darauf beziehen.

Besonders erwähnenswert ist noch der Abgriff der Spannung über R_{sense}. Eine kleine Spannung von maximal 50 mV soll in direkter Nachbarschaft einer hohen Strom führenden Spule, die mit 300 kHz betrieben wird, über Leiterbahnen zum Regelbaustein gebracht werden, ohne das Ergebnis der Messung zu verfälschen. Dazu benutzt man ein differentielles Leitungspaar, um Gleichtaktstörungen zu unterdrücken. Im vorliegenden Fall wurde auf gleiche Länge der Leitungen, abgerundete Ecken und minimalen Abstand geachtet. Auch der symmetrische Abgriff, der auf der analogen Lage 7 zu sehen ist, ist eine Folge dieser Überlegungen.

Experimente

Im Vordergrund der Untersuchungen stehen zwei Eigenschaften des Netzteils: die dauerhafte Stromversorgung bis 9 A im Zusammenhang mit dem Wirkungsgrad und das Verhalten bei einem schnellen Lastwechsel. Dazu wurde ein Testaufbau gewählt, bei dem als Verbraucher zwei regelbare Drahtwiderstände von 5 und 0.3 Ω parallelgeschaltet waren. Dies erlaubte das Einstellen eines Laststroms von 0.3 bis 15 A in guter Genauigkeit. Um das Netzteil ausgiebig testen zu können, wurde es auf einer eigenständigen Platine mit einem möglichst originalgetreuen Layout[27] gefertigt und ausgemessen.

Dauerstrom Zunächst werden Ströme und Spannungen am Eingang und Ausgang gemessen, während die Ausgangslast von 1 bis 10 A durchfahren wird. Man misst sowohl, welchen Ausgangsstrom das Netzteil liefert, bevor es abschaltet, als auch die Genauigkeit, mit der es die Ausgangsspannung regeln kann. Der Quotient aus Ausgangs- und Eingangsleistungen gibt außerdem den Wirkungsgrad, auf den später eingegangen wird.

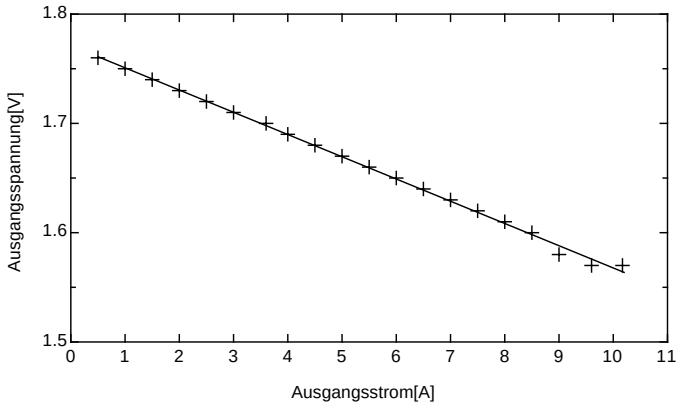


Abbildung 3.25: Ausgangsspannung

Abbildung 3.25 zeigt die Ausgangsspannung bei verschiedenen Ausgangsströmen. Man sieht einen leichten Abfall, und der gesamte Umfang der Spannungsschwankung beträgt 0.2 V, das entspricht einem Fehler um den Mittelwert (1.7 V) von fast 6% und liegt damit an der Grenze der Toleranz des FPGAs. Der Mittelwert von 1.7 statt 1.8 V erklärt sich aus einer falschen Bestückung des Spannungsteilers und kann leicht korrigiert werden. Der große Fehler abhängig vom Strom, der für einen elektronischen Regler untypisch ist, zeigt die Charakteristik des *active voltage positioning* Modus.

Abbildung 3.26 zeigt den Wirkungsgrad des Reglers in zwei unterschiedlichen Betriebsmodi. Die obere Messreihe zeigt den *Burst*-Modus. Dies bedeutet, dass im Falle eines kleinen Ausgangsstroms die Ausgangskondensatoren durch mehrere Zyklen (*Burst*) voll geladen werden und dann mehrere Ladezyklen ausgelassen werden, solange die Kondensatoren der Last gewachsen sind. Reicht die Kapazität nicht aus,

[27] Das Layout der Testplatine *Megavolt* findet sich in Anhang F

um wenigstens zwei Zyklen zu überbrücken, wechselt der Regler in den Modus kontinuierlichen Arbeitens (*continous mode*), in dem jeder Zyklus benutzt wird, um die Kondensatoren zu laden.

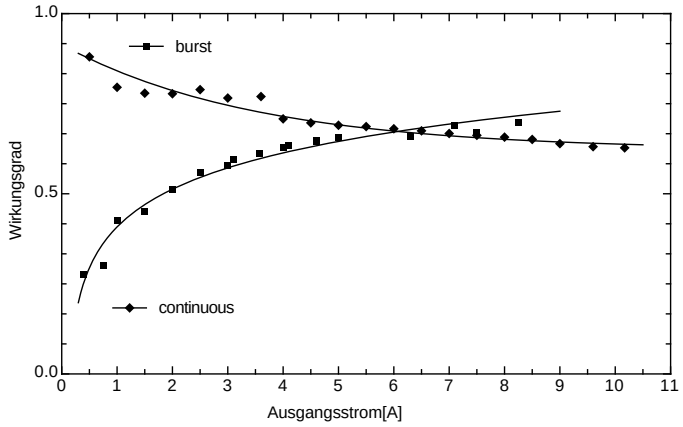


Abbildung 3.26: Wirkungsgrad

Dieses Umschalten der Betriebsmodi hat eine direkte Auswirkung auf den Wirkungsgrad. Der *Burst*-Modus verdeutlicht, dass im unteren Lastbereich ein Wirkungsgrad von über 80 % erreicht wird, indem Zyklen ausgelassen werden. Die untere Kurve zeigt den Regler im erzwungenen kontinuierlichen Modus (FCB an Masse), in dem jeder Zyklus angefangen wird, selbst wenn es nicht nötig wäre. Die überschüssige Energie muss dann durch ein Entladen der Ausgangskondensatoren mit den unteren MOSFETs wieder entnommen werden und vermindert als Verlust den Wirkungsgrad. Ist FCB *high*, so arbeitet der Regler im *Burst*-Modus und schaltet eigenständig in den *continous*-Modus, hier sichtbar bei ca. 4 A.

Die Messungen unter Gleichstrombedingungen zeigen, dass der Regler funktioniert und sowohl eine geregelte Ausgangsspannung von 1.8 V als auch Ströme bis zu 10 A mit einem Wirkungsgrad von über 65 % liefert.

Lastwechsel Von besonderem Interesse ist das Verhalten des Reglers bei schnellen Lastwechseln. Wie bereits ausgeführt, kann ein schnell getakteter FPGA unter Umständen schnellere Lastwechsel erzeugen, als der Regler erfassen kann. Deshalb wurde der Testaufbau verändert, um den beschriebenen *active voltage positioning* Modus auch unter realen Bedingungen zu beobachten.

Um den direkten Vergleich zu den optimalen Werten aus [LTC1735] zu haben, wurde versucht, die dortige Abbildung 9 bzw. 10 nachzumessen. Dazu wurde ein Lastwechsel von 0.3 auf 9 A erzeugt, indem einer der beiden Drahtwiderstände durch ein MOSFET zuschaltbar zum anderen gemacht wurde.

In Abbildung 3.27 ist bei geöffnetem MOSFET nur R1 als Verbraucher angeschlossen und es fließt der Strom I_1. Bei geschlossenem MOSFET wird R2 parallel geschaltet und die Last addiert sich zu $I_1 + I_2$. Der MOSFET wird über einen Frequenzgenerator angesteuert, der Rechteckimpulse variabler Länge mit langer Pause

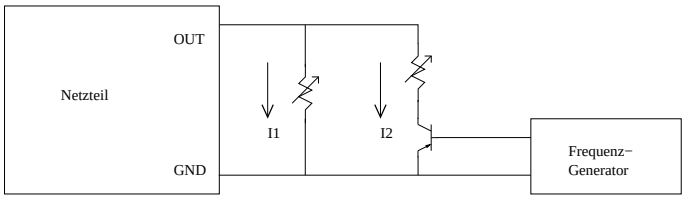


Abbildung 3.27: Erzeugung eines schnellen Lastwechsels

dazwischen erzeugt. So erhält man einen niedrigen konstanten Strom, der mit kurzen Hochstromphasen überlagert ist.

Die Abbildungen 3.28 und 3.29 zeigen auf Kanal 1 die Ausgangsspannung. Als Triggersignal wurde auf Kanal 2 das Schaltsignal des MOSFETs gelegt. Abbildung 3.28 ist die genaue Nachstellung der Abbildung 10 in [LTC1735], die eine gute Übereinstimmung des vorliegenden Layouts mit dem Datenblatt zeigen. Von Interesse sind aber auch längere Impulse wie in Abbildung 3.29 dargestellt.

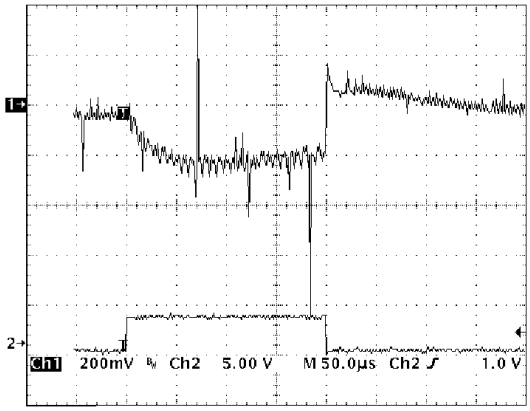


Abbildung 3.28: Lastwechsel Antwort 1

Man erkennt, dass das Verhalten dem *active voltage positioning* entspricht, das ein Absinken der Spannung im Falle des geschalteten MOSFETs, also hoher Last, bewirkt. Der Regler nutzt einen Bereich von 200 mV aus, der nach [Xil:Data] gerade in den Spezifikationen des FPGAs liegt. Die hochfrequenten Störungen auf der Spannung, die hier mit *Megavolt* gemessen wurden, sind nicht kritisch, da sie auf *Darkwing* durch die vielen Kondensatoren direkt am FPGA ausgeglichen werden.

3.7.4 Filterung für den Analogteil

Durch die benutzten Schaltregler, durch schnell getaktete Chips und nicht zuletzt die Störungen, die durch den PCI-Bus kommen, unterliegen die digitalen Spannungsversorgungen großen Störungen. Die digitalen Bauteile bleiben davon unberührt, für die Qualität der analogen Baugruppen jedoch spielen Schwankungen der Versorgungsspannungen eine Rolle, da sie die Referenz für Analogsignale bilden. Daher sollten

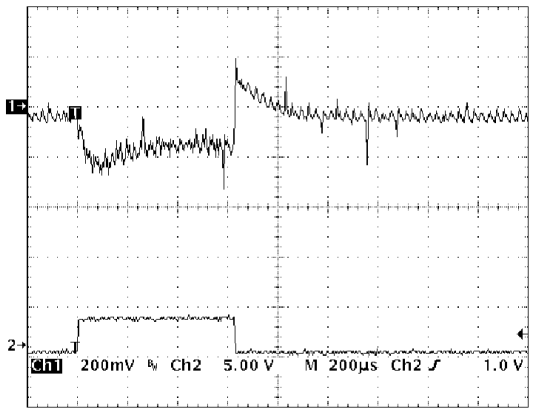


Abbildung 3.29: Lastwechsel Antwort 2

die analogen Spannungen (wie auch das Massepotential) von den digitalen getrennt sein, was teilweise sogar innerhalb eines Chips schon realisiert ist. So bleiben die analogen Strompfade unbehelligt von Störungen der digitalen Ströme. Darüber hinaus benötigen die Operationsverstärker der DACs nicht nur +5 V sondern auch -5 V, um eine Ausgangsspannung von 0 bis +5V zu erzeugen.

Um die analogen ±5 V mit geringem Rauschen zu erzeugen, kann man Linearregler benutzen (Abbildung 3.30, rechts), die wie der MAX604 sehr einfach zu beschalten sind. Der Positiv-Regler LT1963 und der Negativ-Regler LT1175 sind sogenannte Low-Dropout-Regler, denen 0.3 bzw 0.7 V Spannungsdifferenz zwischen Eingang und Ausgang genügen [LT1175], [LT1963]. Beim LT1963 erfolgt die Justierung der Ausgangsspannung wieder durch einen Spannungsteiler (R1, R2), der LT1175 ist als -5 V-Festspannungsregler erhältlich. Außer Ein- und Ausgangskondensatoren sind wiederum keine weiteren Bauteile nötig. Rechts im Schaltplan Abb. 3.30 sieht man Blockkondensatoren, die auf dem Analogteil verteilt für eine weitere Glättung sorgen.

Da Linearregler Eingangsspannungen benötigen, die einige Volt über den Ausgangsspannungen liegen, aber wegen der Verlustleistung auch nicht zu hoch sein sollten, werden +8 V bzw. -7 V aus +5 V (digital) erzeugt. Dazu werden zwei Schaltregler benutzt, die in Abb. 3.30 links zu sehen sind und über RC-Glieder bzw. eine Induktivität die erhöhten Spannungen an die Linearregler liefern. Der LT1172 erzeugt +8 V und der MAX764 ist ein invertierender Regler auf -7 V. Beide Regler funktionieren wieder nach dem Prinzip des Schaltreglers und benötigen Spulen, um die Energie zu speichern [LT1172],[MAX764]. Linearregler können in diesem Fall nicht eingesetzt werden, da sie nur überschüssige Energie über den Widerstand eines Transistors verbrauchen können, während beim aufwärts Regeln die verlustarme Speicherung der Energie in einer Spule und die Zurückgewinnung in einen elektrischen Strom benötigt wird. Wieder legen externe Spannungsteiler (R10, R11 und R13, R14) die Ausgangsspannungen fest.

Die gewählte Zusammenstellung der Regler liefert theoretisch maximal 1.5 A bei +5V und 300 mA bei -5 V. Benötigt werden ca. 1A positiv und 200 mA negativ, so

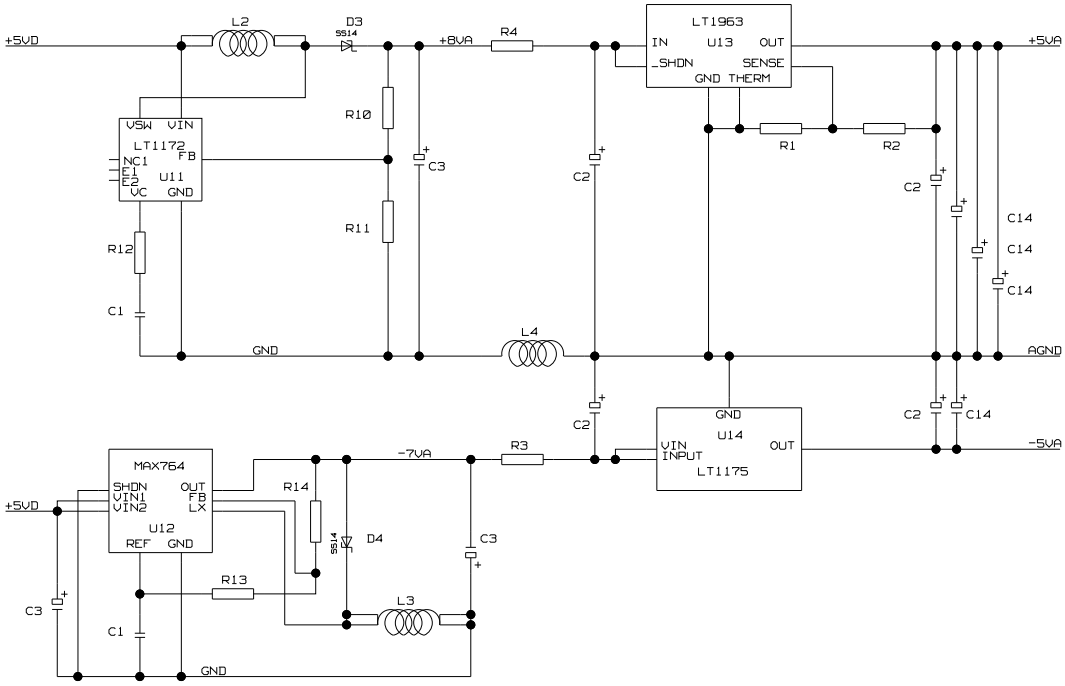


Abbildung 3.30: Netzteile für die analogen Bausteine

dass der positive Regler noch einigen Spielraum nach oben hätte. Wenn von diesen reduzierten Leistungsdaten ausgegangen wird, können aber platzsparende Gehäusetypen verwendet werden, da bei reduzierter Leistung auch die Wärmeabfuhr über das Gehäuse weniger gut sein muß.

3.8 Bestückung und Inbetriebnahme

Nach der Fertigstellung des Layouts und der Produktion der Platine müssen die Bauteile aufgelötet werden. Dies erfolgt in Abschnitten, um das Funktionieren einzelner Teile sicherzustellen, bevor weitere Baugruppen hinzugefügt werden. Bei der Inbetriebnahme des ersten Prototyps wurden die einzelnen Baugruppen besonders intensiv untersucht, um die Qualität der Signale beurteilen zu können. Dabei wurden auch Fehler im Layout verbessert, Optimierungen in der Bestückung vorgenommen und Bauteile kalibriert.

3.8.1 Reihenfolge der Bestückung

Zunächst werden die *ball grid arrays* bestückt, da dies in einem speziellen Verfahren erfolgen muss. Die Gehäuse des PLX-Chips und FPGAs haben auf der Unterseite je eine Matrix aus Lötkontakten, auf denen sich ein Tropfen Lötzinn befindet. Mit Hilfe eines speziellen Mikroskops werden die Gehäuse über den dafür vorgesehenen Kontakten auf der Platine positioniert und abgesetzt. Ein Gel aus Lötflussmittel lässt die Gehäuse zunächst haften, bis die Zinntropfen in einem Ofen verflüssigt werden und eine feste Verlötung herstellen. Dies erklärt, warum zu diesem Zeitpunkt noch keine anderen Bauteile aufgelötet sein dürfen, denn sie würden im Ofen wieder entlötet werden. Diese Technik erlaubt das Verlöten von mehreren hundert Kontakten gleichzeitig mit einer hohen Zuverlässigkeit und stellt sicher, dass die teuren Bauteile keinen Schaden beim Bestücken nehmen.

In einem zweiten Schritt werden die Bauteile bestückt, die den digitalen Kern darstellen, also CPLD, PLL, EEPROM, S-RAMs und alle passiven Bauteile, die in diesem Stadium wichtig sind. Beim Test dieser Minimalbestückung wird die Versorgungsspannung des FPGAs von einem externen Netzteil geliefert, und es wird überprüft, ob der FPGA konfigurierbar ist und das S-RAM vollständig ansprechbar.

Ist die Karte in diesem Stadium funktionsfähig, kann das 1.8 V-Netzteil bestückt und getestet werden. Je nach Anwendung muss der Messwiderstand individuell angepasst und die Überstromabschaltung geprüft werden. Danach erfolgt die Bestückung der vier Netzteile für die Analogspannungen und Prüfung der Ausgangsspannungen und Leistungen.

Zuletzt werden die analogen Komponenten aufgebracht, die sich auf der Rückseite der Platine befinden. Mittels externer Messgeräte werden die erzeugten Spannungen mit Sollwerten verglichen und die DACs kalibriert. Nachdem diese letzten Messungen vorgenommen wurden, ist die Karte vollständig einsatzbereit, und mit einer entsprechenden Adapterplatine kann ein ASIC zum Test angeschlossen werden.

3.8.2 Korrekturen

Ergebnis der ersten Bestückung ist die Korrektur zweier Fehler im Schaltplan bzw. im Layout. Am Schaltregler LT1172 (U11) fehlen zwei Verbindungen zu Masse. Die Anschlüsse 6 und 8 ($E1$ und $E2$) sind an die Emitter der internen Transistoren

angeschlossen und müssen geerdet werden, um maximale Leistung zu erreichen. Dies kann nach der Bestückung korrigiert werden, indem ein Draht vom Masseanschluss 1 um den Chip herum geführt wird.

Ein zweiter Fehler im Layout ist die Orientierung des Linearreglers LT1175 (U14). Die Belegung der Anschlüsse ist auf der Platine gespiegelt zu der des Bauteils. Da es sich bei dem SOT-223 aber um ein kleines spiegelsymmetrisches Gehäuse handelt, ist es unproblematisch, bei der Bestückung das Bauteil auf der Oberseite liegend einzulöten.

3.9 Aufnahme der zu testenden ASICs

Ein wesentlicher Bestandteil des Testsystems ist die Erweiterbarkeit durch eine aufsteckbare Tochterplatine, die die Adaption an einen zu testenden ASIC darstellt. Die einfachste Form ist eine Verlängerung der Signale des CMC-Steckers 4 auf eine Kontaktierung nach außen.

Gosalyn ist der Projektname einer solchen Platine, die die Abwärtskompatibilität zu den bisher verwendeten Trägerplatinen für ASICs herstellt. Verwirklicht sind aber auch kompliziertere Adaptionen, wie z.B. *Hoverboard*, das die LVDS-Signale von den CMC-Steckern 3 und 4 mit kontrollierter Impedanz zum Testchip leitet und Bauteile für die Auslese mit einem Logik-Analysator beinhaltet. Durch Verwendung der CMC-Stecker 1 und 2 besteht außerdem die Möglichkeit, kompliziertere Digitallogik auf einer vollen CMC-Karte anzusteuern.

3.9.1 Gosalyn

Gosalyn ist eine rein passive Tochterplatine, die über dem *IO-Space* liegt und bis zu den CMC-Steckern 3 und 4 reicht. Der CMC-Stecker 4 genügt, um die Belegung des bisher benutzten Steckers[28] zu befriedigen, die bündig mit dem Gehäuse des PCs abschließt. Die zu testenden ASICs befinden sich dann einzeln auf einer von außen steckbaren Trägerplatine.

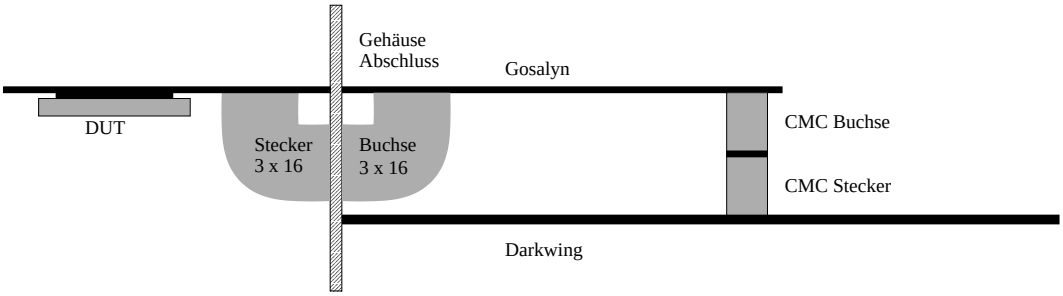


Abbildung 3.31: Tochterplatine *Gosalyn* mit Chipträger

Der Schaltplan ist eine triviale Abbildung vom CMC-Stecker 4 auf die 3x16 Buchse und wird durch Tabelle G.1 im Anhang wiedergegeben.

Neben dem Signalrouting sind auf Gosalyn die Versorgungsspannungen +3.3 V digital und +5 V analog verfügbar, je durch einen Tantalkondensator gepuffert.

Der CMC-Stecker 3 wird für die Beschaltung der 3x16 Buchse nicht benötigt, ist jedoch für die Stabilität vorgesehen. Dies gibt auch die Möglichkeit, den externen Oszillatoreingang der PLL zu benutzen, der auf Gosalyn an eine LEMO-Buchse gelegt ist. Abbildung 3.32 zeigt die *Darkwing* zugewandte Seite von Gosalyn.

[28] 3-reihige Stiftleiste mit je 16 Kontakten

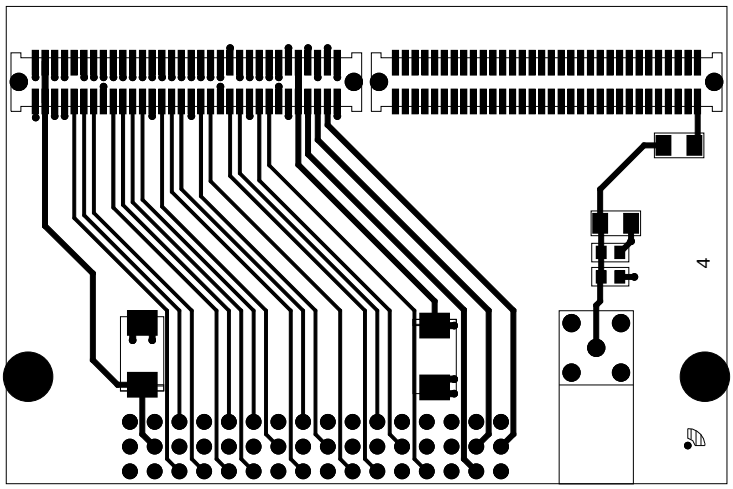


Abbildung 3.32: Unterseite von Gosalyn

Kapitel 4

Messungen

Nach der Inbetriebnahme der ersten Prototypen können Vergleiche zwischen den Karten gezogen werden und das Verhalten im Normalbetrieb charakterisiert werden. Dazu gehören die Überprüfung der Signalintegrität ausgewählter Leitungen, und der Vergleich der erreichbaren Fähigkeiten mit theoretischen Werten. Im Rahmen dieser Arbeit konnte keine Statistik aufgestellt werden, jedoch Messungen an einem Prototypen vorgenommen werden, die kritische Punkte des Designs untersuchen.

4.1 Netzteile

Bei den Netzteilen ist von Interesse, ob die Regelung funktioniert und die Nominalleistung erreicht wird. Dies ist bei allen Netzteilen der Fall, wobei der Leistungstest des 1.8 V-Netzteils besonders hervorzuheben ist. Die bisher geladenen FPGA-Konfigurationen nutzten nur zwischen 30 und 50 % der CLBs und führten kleinere Tests oder Experimente aus, die keine große Belastung darstellten. Auf einer Karte mit bestücktem 1.8 V-Netzteil ist eine Messung der Stromaufnahme des FPGAs nicht mehr möglich. Einfache S-RAM Tests wurden mit einer externen Spannungsquelle ausgeführt und so konnte für diesen Fall eine Stromaufnahme der internen-FPGA Logik von 100 mA gemessen werden. Anspruchsvollere Tests wie die der LVDS-Übertragung wurden unter Verwendung des bestückten Netzteils durchgeführt und führten nur zu einer leichten Erwärmung des FPGAs. Um die Stabilität des 1.8 V-Netzteils auch bei hoher Belastung zu prüfen, wurde daher mit der Parallelschaltung von Lastwiderständen zum FPGA das Netzteil bis an seine Leistungsgrenze belastet und verifiziert, dass der FPGA ordnungsgemäß arbeitet.

Der zweistufige Aufbau der analogen Netzteile ist vorgesehen, um das Rauschen auf den analogen Versorgungsspannungen zu minimieren. Sowohl die Schalt- als auch die Linearregler funktionieren mit der gewünschten Genauigkeit und Leistung. Durch eine Anpassung der Drossel L5 und der Kondensatoren auf dem Analogteil ist es möglich, die Qualität der analogen Versorgungsspannungen zu optimieren.

4.2 Taktsignal

Weitere Tests bestätigen die Funktion des lokalen Busses und des S- und SD-RAMs mit 50 MHz Taktfrequenz. Mit dem S-RAM werden Datenraten von 80 MByte/s beim Lesen und über 100 MByte/s beim Schreiben erreicht. Es ist möglich, die PLL zu programmieren und verschiedene Taktfrequenzen zu erzeugen.

Für ein einwandfreies Funktionieren der synchronen Logik ist es notwendig, dass das Taktsignal gut definierte Pegel und Flanken aufweist. Deshalb wurde beim Layout der beiden Taktungen von der PLL darauf geachtet, möglichst kurze Verbindungen herzustellen. Beide Signale gehen zunächst durch den CPLD, und werden dann an den FPGA bzw. PLX geführt. Abbildung 4.1 zeigt das Signal $FPGA\_CLKA$ mit 50 MHz. Man erkennt, dass die Signalzustände sehr gut definiert und die Signalflanken steil sind.

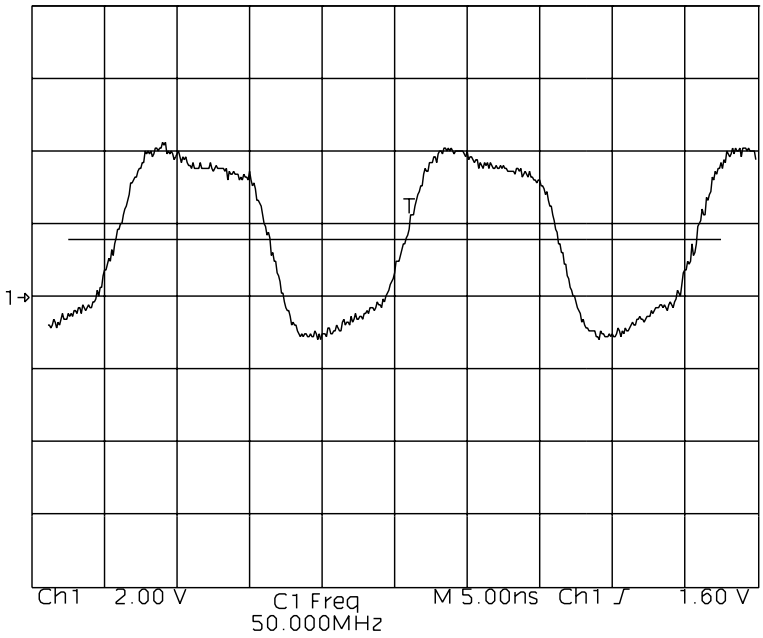


Abbildung 4.1: Taktsignal für den lokalen Bus $FPGA\_CLKA$ bei 50 MHz

4.3 LVDS

Die LVDS-Fähigkeit der Karte wurde mit der Ansteuerung des ASICs ANNA getestet, wobei dieser auf das *Hoverboard* aufgesteckt wird, dass die notwendigen Terminierungen und Laufzeitanpassungen der Leitungen bereitstellt. Mit einem differentiellen Tastkopf können einzelne Signale analog auf einem Oszilloskop dargestellt werden. Für den Anschluss eines Logik-Analysators zur gleichzeitigen Darstellung aller Bussignale, konvertieren außerdem Buffer auf der Adapterplatine die Signale von LVDS nach TTL.

Abbildung 4.2 zeigt die Aufnahme des LVDS-Taktsignals, das vom FPGA getrieben wird, am ANNA Chip aufgenommen mit einem differentiellen Tastkopf. Das Signal schwingt leicht über, wobei die gute Qualität bei 100 MHz darauf schließen lässt, dass auch höhere Frequenzen verwirklicht werden können.

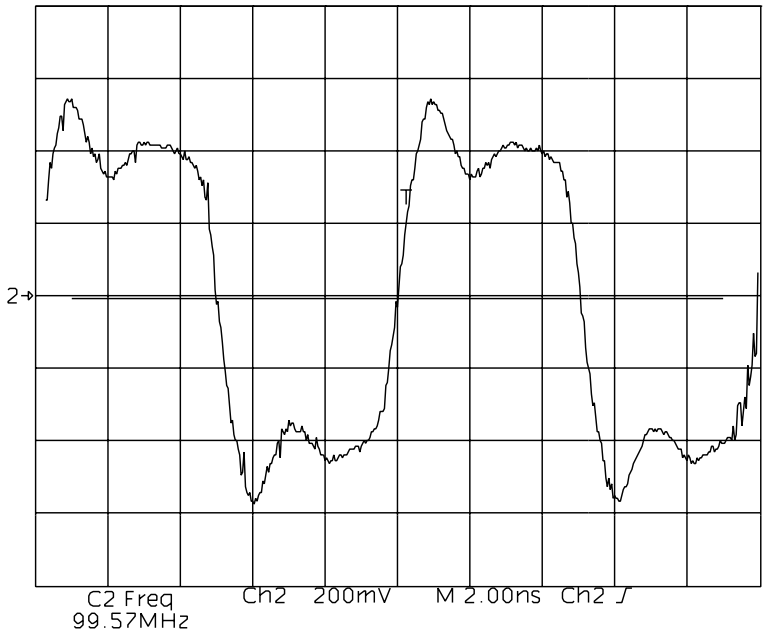


Abbildung 4.2: LVDS Taktsignal am empfangenden ASIC

Um bei solch hohen Frequenzen und langen Leitungen die Synchronität der Taktung sicherzustellen, muss bei einer bidirektionalen Kommunikation das Taktsignal vom jeweils sendenden Baustein erzeugt werden (Quellensynchronität). Es gibt zwei unabhängige Taktleitungen, die jeweils für eine Kommunikationsrichtung benutzt werden. Die Datenleitungen werden bidirektional benutzt, wobei bei einem Richtungswechsel durch Pausen sichergestellt wird, dass eine Leitung nicht von beiden Bausteinen gleichzeitig getrieben wird. Abbildung 4.3 zeigt den Wechsel zwischen Lesen und Schreiben von zwei 8 Bit Worten Daten und drei Adressbits. $CLKin$ ist das Taktsignal, das ständig vom FPGA erzeugt wird, während $CLKout$ nur vom ASIC erzeugt wird, wenn Daten zum FPGA gesendet werden. Diese Abbildung ist die Messung einer realen Kommunikation mit einem Logik-Analysator, der eine maximale Abtastrate von 4 ns besitzt. Daher können Laufzeitunterschiede der Signale nicht besser aufgelöst werden und alle Flanken erscheinen hier synchron.

Nach der Markierung G1 ist ein Takt zu sehen, in dem Daten vom FPGA gelesen werden. Danach ist ein halber Takt Pause, in dem die Leitungen unbenutzt sind, bevor die Taktung des ASICs einsetzt, um Daten zum FPGA zu senden. Danach folgen drei Taktzyklen, in denen sowohl zur steigenden als auch zur fallenden Flanke Daten übertragen werden. Vor der Markierung G2 ist wieder ein Takt, in dem die Leitungen nicht getrieben werden, bevor Daten vom FPGA empfangen werden.

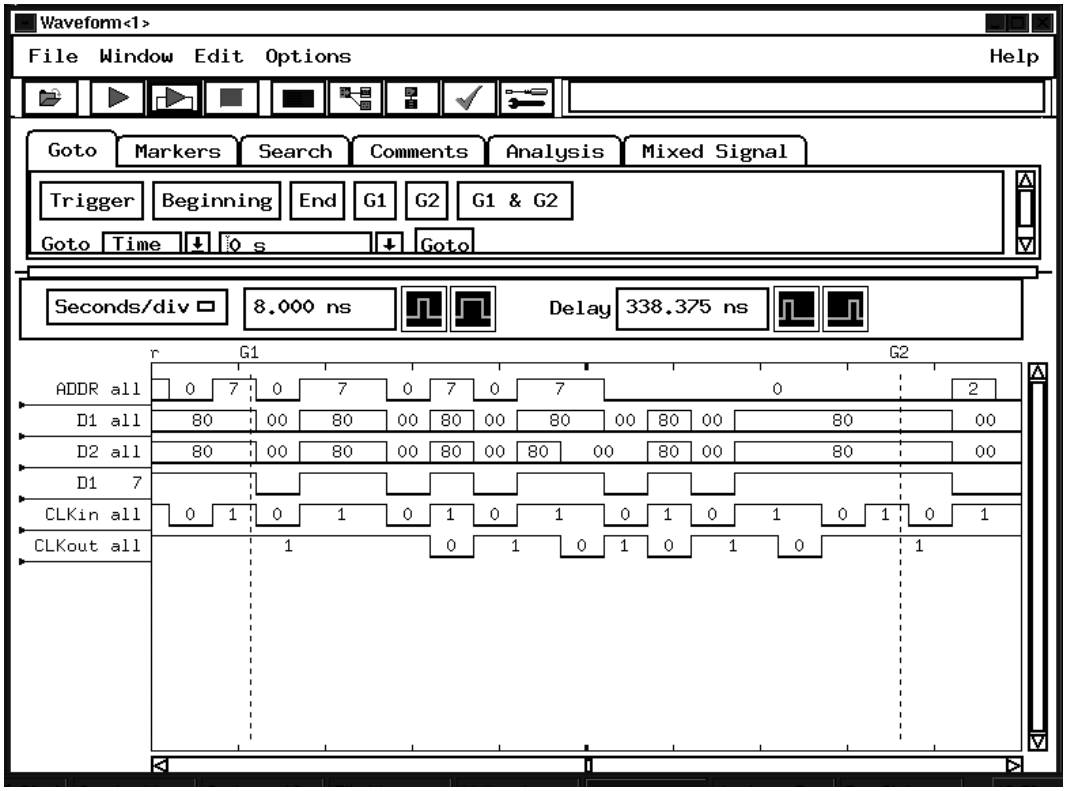


Abbildung 4.3: Timing-Messung von bidirektionalem Bus-LVDS

Zusätzlich zu den Taktsignalen und Datenworten ist der Zustand des Bits D7 aufgetragen, der gleichzeitig mit einem hochauflösenden Oszilloskop aufgenommen wurde. Hier ist statt dem idealisierten logischen Zustand die genaue Beschaffenheit des Signals zu erkennen. In Abbildung 4.4 ist die oben genannte Markierung G1 als Triggerpunkt T zu erkennen. Zunächst folgt der halbe Takt, in dem das Signal nicht getrieben wird und auf 0 zurückfällt. Dann ist ein Treiben der Leitung durch den FPGA an der hohen Amplitude von 400 mV auf *high* und *low* zu erkennen. Das Signal wird dann vor dem Richtungswechsel nicht getrieben und gleitet langsam auf 0 zurück. Mit der kleineren Amplitude von 200 mV erscheinen daraufhin die Signale, die vom ASIC ausgegeben werden.

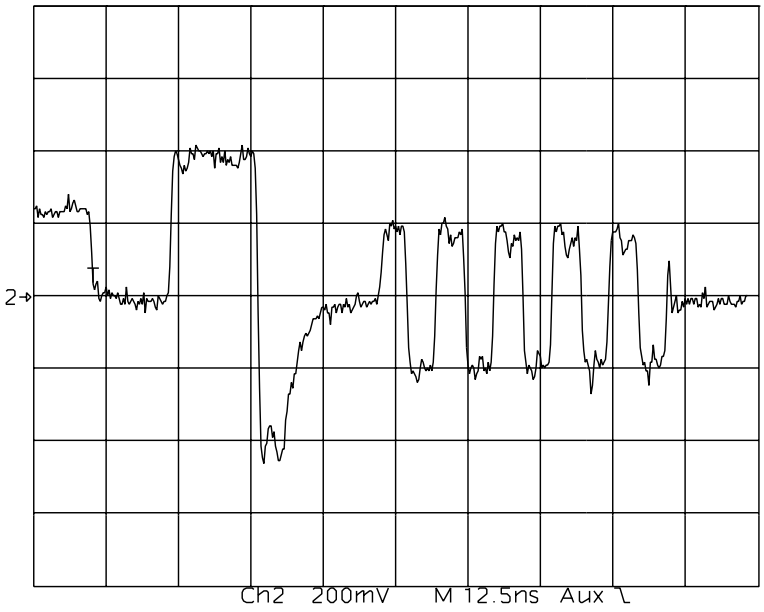


Abbildung 4.4: LVDS-Datenleitung D7 bei bidirektionaler Benutzung

4.4 16 Bit-DAC

Bei der Charakterisierung des 16 Bit-DACs stehen zwei Größen im Vordergrund. Zum einen ist die erreichbare Auflösung zu bestimmen und zum anderen die Geschwindigkeit, die vom verwendeten Operationsverstärker abhängt.

Voraussetzung für eine Bestimmung der Auflösung ist eine rauscharme Spannungsversorgung der Bausteine. Da eine genaue Untersuchung des Rauschverhaltens und eine Optimierung bisher nicht erfolgte, ist dies aber noch nicht gegeben. Das Rauschen pflanzt sich auf die Ausgänge des DACs und Operationsverstärkers fort und so ist keine bessere Genauigkeit möglich. Durch Mittelung der Ausgangsspannung kann zwar ein genauerer Wert gemessen werden, doch diese Ergebnisse liegen weit unter der erwarteten Genauigkeit und sind nicht aussagekräftig, solange die Versorgungsspannungen keine akzeptable Qualität aufweisen.

Auch das dynamische Verhalten ist aus den genannten Gründen nur begrenzt auswertbar. Abbildung 4.5 zeigt die Messung der Ausgangsspannung mit veränderlichen Werten. Bei der benutzen einfachen Software wurden die Werte direkt vom PCI-Bus auf den DAC geschrieben und nicht aus dem lokalen RAM gelesen. Deshalb konnte die maximale Geschwindigkeit des DACs nicht erreicht werden. Man sieht jedoch, dass bei einer Terminierung von 50 Ω der Operationsverstärker schnell genug ist, um in 20 ns 1 V Spannungshub zu leisten. Dies erfüllt voll die Erwartungen, die an die Erzeugung analoger Testmuster gestellt wurden.

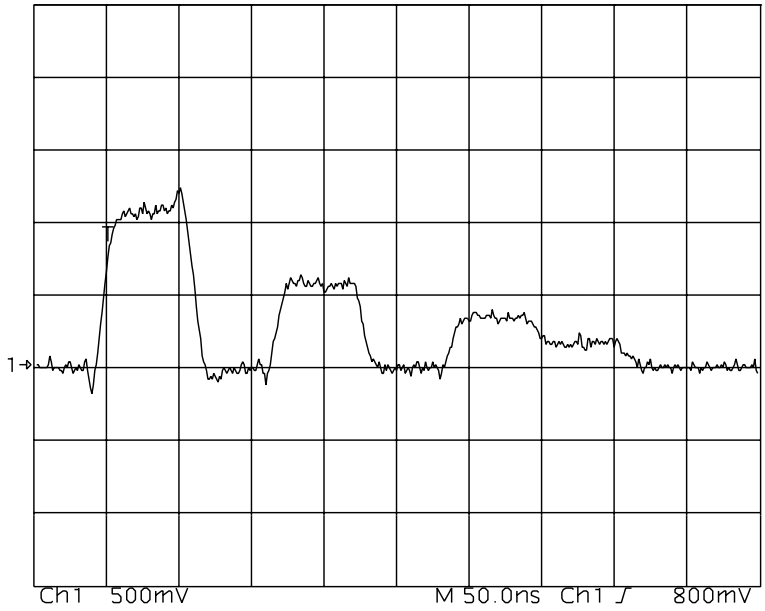


Abbildung 4.5: Ausgabe des 16 Bit-DACs

Kapitel 5

Software

In der Übersicht (Kapitel 2) wird die Trennung der Software in zwei Teile erläutert. Der Teil der Software, der das Experiment auf PC-Ebene darstellt, bildet eine projektspezifische Benutzeroberfläche, die eine Interaktion mit der Hardware ermöglicht. Gemeinsame Randbedingung ist der Zugriff auf die Hardware über die Betriebssystemtreiber, die durch [Jungo] bzw. [Bau01] definiert werden.

In diesem Kapitel wird auf den anderen Teil der Software eingegangen, der das Verhalten der programmierbaren Bauteile beschreibt, die auf der Karte zusammenwirken. Auf die Konfiguration des EEPROMS und damit des PLX sowie den VHDL-Code des CPLDs wird in Absatz 3.2 eingegangen. Hier sollen grundlegende Strukturen einer FPGA Programmierung erläutert werden. Dies führt zur Einführung einer Simulationsumgebung und der Erstellung eines VHDL-Modells am Beispiel des FPTA-Chips.

5.1 Allgemeiner-FPGA Code für *Darkwing*

Die Programmierung des FPGAs bildet die Schnittstelle zwischen der PC-Software und den Bauteilen auf dem Board. Die Ansteuerung der Bauteile erfolgt nicht nur rein elektrisch durch den FPGA, sondern es werden auch Kommunikationsprotokolle durch den FPGA eingehalten, die für die PC-Software nicht sichtbar sind. Es können voneinander unabhängige *state machines* ablaufen, die auf Signale von außen reagieren und jeweils bestimmte Teile der Funktionalität übernehmen.

Im VHDL-Code werden Register definiert, die lokale Speicher darstellen. Dies geschieht durch eine Deklaration der Art:

```
signal control_reg    : std_logic_vector(31 downto 0);
signal write_reg      : std_logic_vector(31 downto 0);
signal read_reg       : std_logic_vector(31 downto 0);
signal adc_reg        : std_logic_vector(15 downto 0);
signal dac_reg        : std_logic_vector(15 downto 0);
```

Dabei entstehen 32 bzw 16 Bit breite Register, die intern geschrieben und gelesen werden können. Damit ist aber auch ein Zugriff von außen leicht realisierbar,

denn eine einfache *state machine* kann zu den Flanken einer Taktung die Daten von externen Anschlüssen lesen und in ein Register übernehmen. Beispielhaft dafür ist das Einlesen von ADC-Werten, das unabhängig von anderen Programmteilen ständig synchron zur Taktung `adc_clk` Werte von den Datenleitungen `adc_data` in das Register `adc_reg` lädt.

```
process (adc_clk)
begin
   if adc_clk'event and adc_clk = '1' then
      adc_reg(15 downto 0) <= adc_data (15 downto 0) ;
end process;
```

Entsprechend kann die Ansteuerung des parallelen DACs auch über ein Register erfolgen, indem die *state machine* sensibel auf die Änderung des Registerinhaltes ist und die Taktung und Daten an die externen Anschlüsse gibt. Soll der DAC gesperrt sein, bis der analoge Ausgang den gewünschten Wert angenommen hat, müssen unter Umständen Wartezyklen eingeschoben werden. Ein Zugriff auf den DAC von anderen Modulen aus erfolgt dann durch Schreiben eines Wertes in das Register `dac_reg`.

Die Datenübermittlung zu den seriellen DACs kann ähnlich realisiert werden, allerdings muss die *state machine* das parallel empfangene Datenwort serialisieren und Takt für Takt ein Bit in das Schieberegister[1] der DACs schreiben. Dies ist leicht zu implementieren. Da die serielle Taktung langsam ist und sogar ausbleiben kann, bis das nächste Datenbit übertragen wird, kann die Serialisierung der Daten schon in der PC-Software erfolgen, und die Daten werden bitweise über den lokalen Bus in den FPGA geschrieben. So können ein Register und die Serialisierungslogik im FPGA gespart werden, wobei eine Übertragung von einzelnen Bits in 32 Bit PCI-Worten die Effizienz des Busses verringert.

Im Falle des Datenaustauschs mit dem lokalen Bus existiert ein umfangreicheres Modul, das die Kommunikation mit dem PLX abwickelt und die gemultiplexten Daten des lokalen Busses nach einem Schreibzyklus auf den FPGA als Adress- und Datenwort zur Verfügung stellt. Eine *state machine* reagiert auf das Signal `pclk` und schreibt die Daten in das durch `address` selektierte Register. Eine vereinfachte Version des VHDL-Codes verdeutlicht dies.

```
process (reset, pclk)
begin
   if reset='1' then
      control_reg   <=(others=>'0');
      write_reg     <=(others=>'0');
   else
      if pclk'event and pclk = '1' then -- on rising edge of pci-trigger
         case address is
            when writeadr=>
               write_reg <= data_from_pci (31  downto 0) ;
            when controladr=>
               control_reg <= data_from_pci (31  downto 0) ;
            when others => NULL ;
         end case;
```

[1] siehe Absatz 3.6.1

```
        end if;
      end if;
  end process ;
```

An dieser Stelle soll nicht die genaue Funktionsweise aller weiteren Module, insbesondere der RAM und PLX-Ansteuerung, im Detail erklärt werden. Durch die einfachen Beispiele ist aber klar, dass *state machines* entworfen werden können, die das Protokoll auf Bauteilebene kapseln und Daten über Register austauschen. Wichtig ist dabei, dass Signale jeweils nur von einem Prozess getrieben werden, damit kein undefinierter Zustand eintritt. Daher ist auch ein direkter Zugriff von allen auf einen externen Baustein zu vermeiden, und die Kommunikation mit jedem einzelnen Baustein ist ausschließlich durch ein bestimmtes Modul auszuführen, das die Protokolle einhält und eine Verwaltung der Ressource beinhaltet.

Die Arbitrierung der Zugriffe auf dieses Modul ist im einfachsten Fall durch ein zusätzliches Signal zu realisieren, das den Benutzungsstatus einer Ressource anzeigt. Ist das Signal deaktiviert, kann ein Modul die Ressource reservieren, und das Signal wird gesetzt. Ein zweites Modul wird vor dem Zugriff das Statussignal lesen und warten, bis die Reservierung aufgehoben ist. Ist der erste Vorgang abgeschlossen, wird das Statusbit gelöscht, und das zweite Modul kann die Ressource reservieren.

Diese einfache Verwaltung, bei der immer ein Modul die volle Kontrolle über eine Ressource hat, entspricht einer Serialisierung paralleler Anfragen. Sie ist nur sinnvoll, falls eine Ressource größtenteils von einem Modul benutzt wird und nur selten Konflikte mit anderen Modulen auftreten. Wird eine Ressource ständig von verschiedenen Modulen benötigt, kann es passieren, dass ein Modul die Kontrolle behält und die Ressource niemals für andere Prozesse frei gibt. Außerdem kann es notwendig sein, Prioritäten einzuführen, damit gewisse Prozesse vorrangigen Zugriff erhalten, weil deren Ausführung nicht verzögert werden kann.

Für die Verwaltung des RAMs wird ein anspruchsvolleres Steuerungsmodul entworfen, das zum einen intern eine einheitliche Ansteuerung von S- und SD-RAM erlaubt und zum anderen die parallele Bearbeitung verschiedener Anfragen leistet. Das RAM muss unter Umständen ständig sowohl für DMA-Zugriffe über den lokalen Bus als auch für Zugriffe vom Hauptmodul, als auch für automatisierte analoge Daten Ein- und Auslese erreichbar sein. Diese Parallelität wird erreicht durch die Vergabe von *Ports*, von denen jeder eine ständige Verbindung zum RAM-Modul darstellt. Durch Steuerleitungen kann jeder Port einen RAM-Zugriff anmelden und das RAM-Modul serialisiert die Anfragen je nach Anzahl und Priorität der Ports. Außerdem übersetzt es das gemeinsame interne Protokoll in die spezifische Ansteuerung der Bauteile und übernimmt administrative Aufgaben, wie das Umschalten zwischen den Bausteinen oder das Auffrischen[2] des SD-RAMs.

Neben den unveränderlichen Modulen zur Ansteuerung der fest installierten Hardware gibt es ein Modul, das den zu testenden ASIC ansteuert. Dieses Modul ist das eigentliche Hauptmodul jedes einzelnen Experimentes, da es die Funktionalität und den Ablauf des Tests festlegt. Im einfachsten Fall stellt dieses Modul die Kommunikation mit dem ASIC bereit, die von der PC-Software benutzt wird, um Daten

[2] Siehe Abschnitt 3.4.2

mit dem Chip auszutauschen. Im Hinblick auf Evolutionsexperimente mit konfigurierbaren mixed-signal ASICs kann ein solches Modul neben den grundsätzlichen Kommunikationsaufgaben aber auch Teile des genetischen Algorithmus ausführen und beispielsweise die Bewertung eines Individuums in Hardware ausführen. Der lokale Speicher auf dem Testsystem gibt die Möglichkeit, eine ganze Generation von Individuen in den Speicher zu laden, den ASIC mit jeder Konfiguration zu testen und eine Bewertung aller Individuen durchzuführen. Obwohl Mutation und Selektion in der Aufbauphase noch in PC-Software stattfinden, ist anzustreben, auch diese in das Hauptmodul des FPGA Codes einzubinden und so ganze Evolutionsexperimente vollkommen autonom auf einer *Darkwing*-Karte durchzuführen. Die PC-Software kann mehrere solcher Systeme in einem Rechner integrieren und eine Verwaltung der Experimente vornehmen.

5.2 FPGA-Code zur Ansteuerung des FPTAs

Beim ersten Entwurf eines solchen ASIC-spezifischen Moduls steht ungeachtet der späteren Komplexität die Implementierung der grundlegenden Kommunikation. Diese ist gegeben durch Timing-Diagramme wie in Abbildung 5.1 dargestellt. Am Beispiel eines Schreibzugriffs auf eine Spalte des internen S-RAMs des FPTAs soll gezeigt werden, wie ein Timing-Diagramm in VHDL-Code umgesetzt wird. Eine ausführliche Beschreibung des FPTA-Chips würde an dieser Stelle zu weit führen und kann [FPTA] entnommen werden. Von Bedeutung ist hier nur, dass der ASIC ein Feld von 256 konfigurierbaren Transistorzellen enthält. Jede Zelle hat vier S-RAM Einheiten mit je 6 Bit Speicherkapazität, die die Konfiguration der Zelle repräsentieren. Die Speichereinheiten aller Zellen sind so miteinander verbunden, dass sie nach außen einen homogenen Speicher darstellen, der 64 Spalten und 96 Reihen hat. Zum Lesen des Speichers wird eine Spalte in einen Zwischenspeicher gelesen und dann in Päckchen von je 6 Bit auf die Datenleitungen gegeben, während beim Schreiben erst durch 6 Bit Päckchen eine Spalte im Zwischenspeicher zusammengesetzt wird, die dann durch einen Schreibbefehl als ganze Spalte ins Konfigurations-RAM übernommen werden kann.

Im linken oberen Block in Abbildung 5.1 ist das Timing für das Schreiben von 6 Bit Päckchen gezeigt, wobei dieser Block 16 mal mit verschiedenen Adressen und Daten wiederholt wird, bis eine Spalte zusammengesetzt ist. AR_EN signalisiert die Übernahme der Daten in den Zwischenspeicher. Soll die Spalte aus dem Zwischenspeicher in das RAM übernommen werden, gibt die Adresse die Spalte, und AW_EN signalisiert die Aktivierung der Speicherung.

Da VHDL-Code grundsätzlich parallel umgesetzt wird, wird eine synchrone Kommunikation durch eine *state machine* dargestellt, die jeden einzelnen Taktzyklus des Timing-Diagramms enthält. Eine Darstellung der entsprechenden *state machine* als gerichteter Graph ist in Abbildung 5.2 sichtbar. Der Anfangszustand der *state machine* ist IDLE, und zu jeder steigenden Flanke des Taktsignals kann ein Übergang in einen neuen Zustand stattfinden. Durch das Kontrollregister wird die *state machine* in einen der beiden Zweige geleitet, die dann automatisch ablaufen, bis IDLE wieder

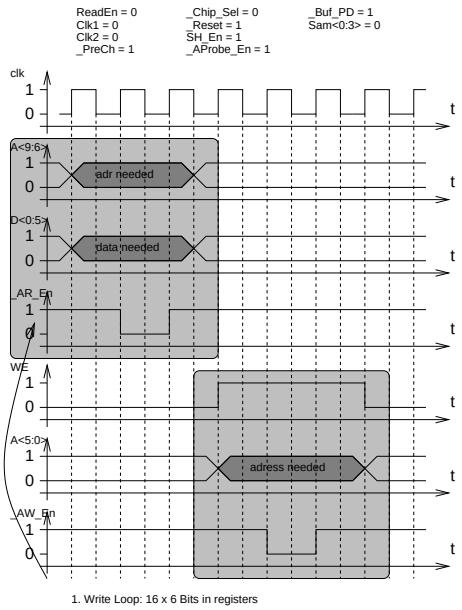


Abbildung 5.1: Timingdiagramm eines FPTA-Schreibzugriffs

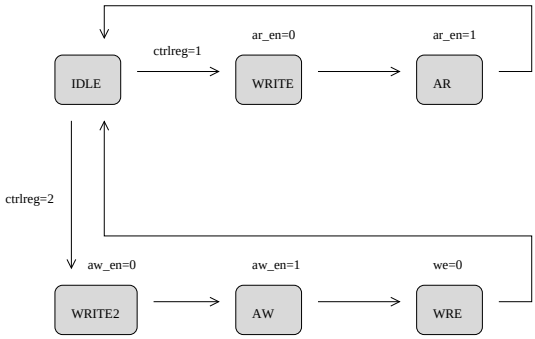


Abbildung 5.2: *state machine* eines FPTA-Schreibzugriffs

erreicht ist. Die Bezeichnungen über den Zuständen zeigen, welche Signale gesetzt werden.

Die Konfiguration einer Spalte kann nun von der PC-Software über Registerzugriffe gesteuert werden. Daten und Adressen werden vor Beginn einer Aktion in die entsprechenden Register geschrieben. Durch Schreiben des Kontrollregisters kann eine Aktion gestartet werden, die wie beschrieben abläuft. Für das Schreiben einer ganzen Spalte muss die PC-Software also in einer Schleife für alle 16 Reihenadressen Daten schreiben und dann die gesamte Spalte schreiben. Eine zweite Schleife führt dies für alle Spalten aus und so kann der ganze Konfigurationsspeicher geschrieben werden. Die Hardware im FPGA stellt also die Grundeinheiten von Kommunikationsblöcken zur Verfügung, die dann von der PC-Software zu kompletten Transaktionen zusammen gefügt werden.

Dieses Konzept ist auch für alle anderen Aktionen des FPTAs verwirklicht. Neben dem Schreiben des Konfigurationsspeichers gibt es auch die Möglichkeit, den Speicher auszulesen, was ebenfalls in Spalten und Zeilen aufgeteilt ist. Außerdem gibt es Funktionen zur Konfiguration von IO-Buffern und analogem Routing.

Abschließend ist zu bemerken, dass die Wahl der Kommandoblöcke sich wie gezeigt auf möglichst elementare Blöcke beschränkt. Für den ersten Test des FPTAs sollte es die Möglichkeit geben, jede Aktion in möglichst kleiner Granularität aufzurufen, damit Fehler leichter gefunden und alle erdenklichen Timing-Diagramme erzeugt werden können. Der Übergang zwischen PC-Software und FPGA-Software ist aber fließend in dem Sinne, dass viele Aufgaben von beiden Teilen gleichermaßen bewältigt werden können. Als Beispiel dient hier wieder das Schreiben des Konfigurationsspeichers, wo eine Schleife über 64 Spalten von 16 Schreibvorgängen von je 6 Bit für eine volle Konfiguration benötigt wird. Die kleinsten notwendigen Einheiten sind die beiden beschriebenen Teile für Zeilen- und Spaltenselektion. Die doppelte Schleife, die bisher in der PC-Software abläuft, kann später durch einen entsprechenden Prozess in Hardware ersetzt werden, der die Konfigurationsdaten nicht Stück für Stück mit Kommandos über den lokalen Bus erhält, sondern mit der Übergabe der Adresse der kompletten Konfigurationsdaten im lokalen RAM der Karte automatisch eine komplette Konfiguration vornimmt. So können nach der Testphase eines ASICs kompliziertere Standardabläufe als Prozesse im VHDL-Code implementiert werden, was die Stabilität verbessert. Die PC-Software kann dann nur noch funktionsfähige Routinen der Hardware aufrufen und nicht mehr eigene, unter Umständen unsinnige, Kombinationen von Kommandoblöcken bilden.

Weiterhin kann nach der Inbetriebnahme und Ausmessung des ersten Chips auch das Evolutionsexperiment auf diesem System stattfinden. Es ist kein triviales Problem, zu entscheiden, welche Teile des dazu notwendigen genetischen Algorithmus in Software oder Hardware implementiert werden. Die Umsetzung umfangreicherer Algorithmen in Hardware bedeutet einen großen Aufwand, der den Rahmen dieser Arbeit sprengen würde.

5.3 Simulationsmodell für den FPTA

Um eine *state machine* für den FPGA zu testen, wird eine Testumgebung [3] für den VHDL-Code angelegt, der wiederum eine *state machine* in VHDL ist, die das zu testende Modul umschließt. Es werden relevante Testvektoren erzeugt, und die Reaktion des Moduls simuliert, so dass die erzeugten Signale in Timing-Diagrammen angezeigt und überprüft werden können.

Eine *testbench* für einen VHDL-Code des FPGAs auf *Darkwing* muss die Kommunikation mit den anderen Bauteilen simulieren können. Der CPLD und die PLL werden ausschließlich vom PC über den lokalen Bus angesprochen und sind für eine FPGA-Programmierung „unsichtbar“. Der lokale Bus als Kommunikation zwischen PLX und FPGA muss aber simuliert werden, damit das FPGA-Modul Befehle entgegen nehmen kann. Dazu dient ein Modul, das Daten aus einer Datei liest und im

[3] engl: testbench

Protokoll des lokalen Busses an die entsprechenden Signale des FPGA-Moduls legt. Für die Kommunikation mit dem RAM gibt es von den Herstellern entsprechende Module, die Timing-Informationen der realen Chips enthalten und in die *testbench* eingebunden werden können. Das Verhalten der analogen Bauteile bedarf keiner Simulation, da das Timing hier nicht bidirektional ist und unkritisch.

Die Teile des FPGA-Moduls, die sich mit der Kommunikation mit fest verdrahteten Bauteilen auf *Darkwing* beschäftigen, können für alle Projekte in gleicher Form benutzt werden. Um die Kommunikation mit dem zu testenden ASIC zu simulieren, muss aber ein Simulationsmodell für den jeweiligen Chip verfasst werden. Je nach gewünschter Detailtreue enthält es die digitale Kommunikation, Speicherung von Daten im ASIC und Simulation interner Datenmanipulation. Dabei können die beim Design des ASICs zu Grunde gelegten Kommunikationsprotokolle und Zeitkonstanten angenommen werden oder sogar Messwerte von getesteten ASICs berücksichtigt werden.

Ein einfaches Modell des FPTAs ist als VHDL-Code in Anhang B angegeben. Das beschriebene Modell enthält keine analoge Funktionalität sondern ausschließlich die digitale Kommunikation zur Konfiguration und Auslese der S-RAM Zellen des ASICs. Zur Erstellung dieses Modells wurden keine Messwerte eines bestehenden Chips verwendet, sondern Timing-Diagramme der theoretischen Kommunikation in eine *state machine* übertragen.

5.3.1 Interne Signale und Speicher

Das Entity `fpta_model` enthält alle digitalen Signale, die zur Ansteuerung des FPTA-Chips benötigt werden und definiert somit die Schnittstelle.

```
entity fpta_model is
   port (
        fpta_a    :in std_logic_vector (9 downto 0) ;
        fpta_d    :inout std_logic_vector (5 downto 0) ;
        fpta_sam  :in std_logic_vector (3 downto 0) ;
        readen    :in std_logic ;
        clk1      :in std_logic ;
        clk2      :in std_logic ;
        prech     :in std_logic ;
        chip_sel  :in std_logic ;
        resetf    :in std_logic ;
        sh_en     :in std_logic ;
        aprobe_en :in std_logic ;
        buf_pd    :in std_logic ;
        ar_en     :in std_logic ;
        we        :in std_logic ;
        aw_en     :in std_logic ;

        -- clk for simulation only
        clk       :in std_logic ;
   ) ;
end fpta_model ;
```

Um die S-RAM Zellen, die die Konfiguration des FPTAs beinhalten, zu simulieren, wird die interne Aufteilung des Speichers in 64 Spalten zu je 96 Bit durch 64 Register vom Typ `fpta_word` realisiert. So können Konfigurationsdaten an die entsprechende Stelle im simulierten Speicher geschrieben werden und stehen zu einem späteren Zeitpunkt zur Auslese zur Verfügung:

```
type fpta_word is array (natural range <>) of std_logic_vector (95 downto 0) ;
signal fpta_sram  :fpta_word (63 downto 0) ;
```

Die Funktionen `my_word` und `my_row`, die im vollständigen Quellcode in Anhang B stehen, dienen der internen Decodierung von binär empfangenen Spalten- und Zeilenadressen in Dezimalzahlen, die als Argumente für den Zugriff auf Elemente des Feldes in VHDL benötigt werden. An dieser Stelle wird besonders deutlich, dass dieses Modul nicht für die Instantiierung in einer Hardware geschrieben ist, sondern für den Ablauf einer Simulation auf einem PC. Die Einführung von Integer-Variablen und for-Schleifen wäre nicht dazu geeignet, in Gatterlogik umgesetzt zu werden.

5.3.2 Grundfunktionen für die Kommunikation

Der Prozess `stateloop` enthält die *state machine*, die die digitale Kommunikation interpretiert. Abbildung 5.3 zeigt eine Darstellung des Automaten als gerichteten Graphen.

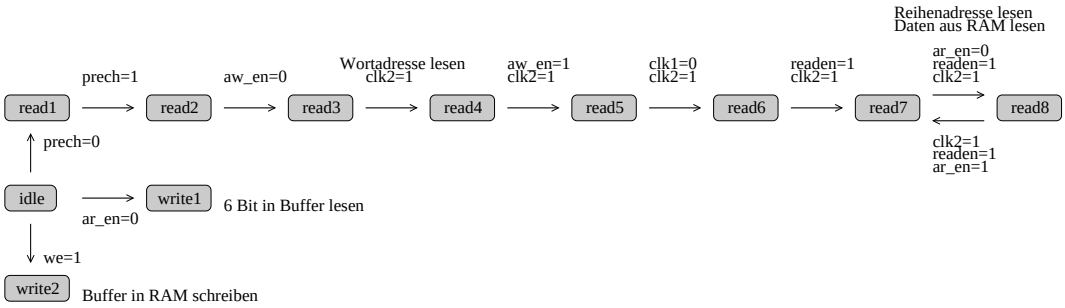


Abbildung 5.3: *state machine* des FPTA-Modells

Eingezeichnet sind immer die notwendigen Signaländerungen, um in den neuen Zustand zu gelangen. Werden diese Bedingungen nicht erfüllt, fällt der Automat immer in Zustand IDLE zurück.

Die beiden Schreibzustände repräsentieren die zunächst voneinander unabhängigen Zeilen bzw. Spalten. Beim Schreiben in eine Zeile wird in den Zwischenspeicher geschrieben, während beim Schreiben einer Spalte immer der aktuelle Inhalt des Spaltenzwischenspeichers in das Speicherfeld geschrieben wird. Während das Schreiben durch die AW_EN und AR_EN Leitungen ein kurzes Protokoll hat, ist das Lesen eine lange zusammenhängende Kette von notwendigen Signalen. Zur Verdeutlichung der Abfolge der Steuersignale ist in Abbildung 5.4 das Timing-Diagramm dargestellt,

das mit der state machine übereinstimmt. Man sieht zuerst die Selektion der Spalte und dann das mehrmalige Auslesen von 6 Bit Päckchen aus dem Spaltenspeicher.

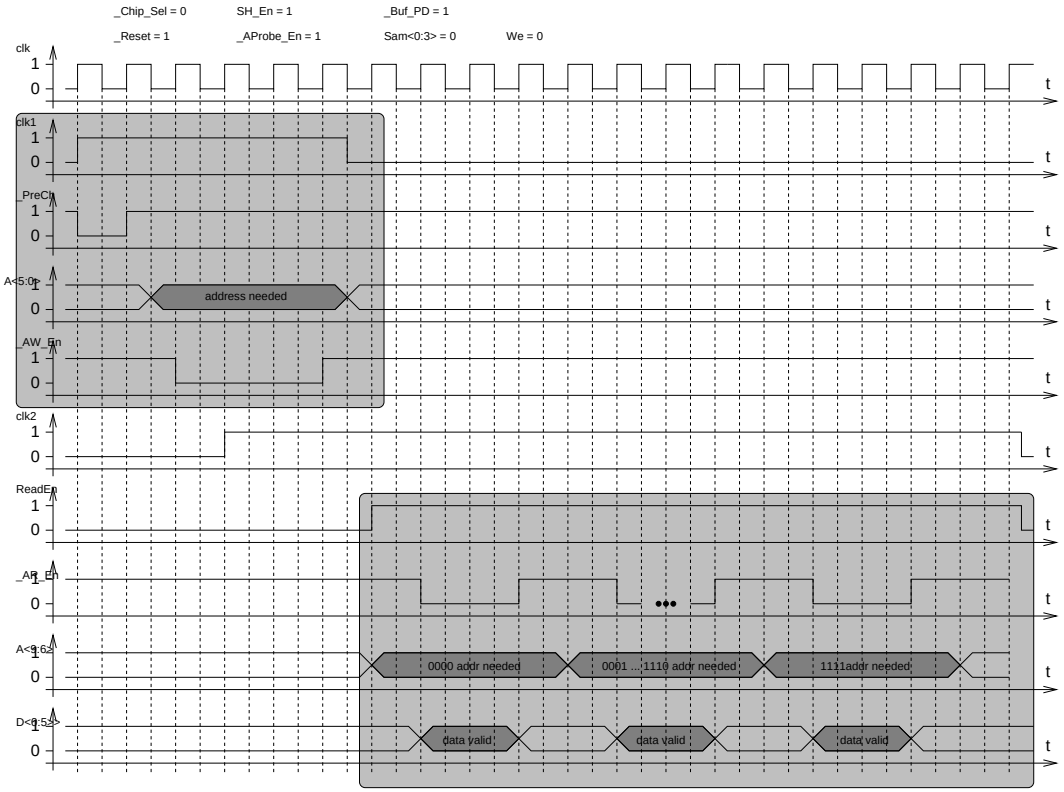


Abbildung 5.4: Timing-Diagramm eines FPTA-Lesezugriffs

Die Abbildung der theoretischen Timing-Diagramme in einem VHDL-Code ist trivial. Es sind bisher noch keine Eigenschaften des echten ASICs eingegangen, wie Setup-Zeiten oder das Verhalten bei falscher Ansteuerung. Auch das analoge Verhalten bleibt vollkommen außer Acht. Dennoch bietet dieses Modell eine Hilfe beim Erstellen des Testmoduls und die Möglichkeit der Fehlersuche in einer Simulation. Je nach Anforderungen an die Simulationsumgebung kann auf dieses Modell aufgebaut werden, und es können nach und nach Eigenschaften des realen Chips hinzugefügt werden.

Kapitel 6

Zusammenfassung

Am Beginn dieser Arbeit stand die Aufgabe, ein Testsystem für konfigurierbare gemischt analog/digitale ASICs zu entwerfen. Dieses sollte sowohl flexibel genug sein, um in der Erprobungsphase eines Experimentes neue Chips in Betrieb zu nehmen und Fehler analysieren zu können, als auch in der Messphase mächtig genug, um kompliziertere Aufgaben wie etwa einen genetischen Algorithmus selbstständig ausführen zu können.

Die in Kapitel 1 definierten Anforderungen führten zur Entwicklung einer PCI-Karte mit einem Virtex-E FPGA als Hauptkomponente, wobei durch die gewählte Gehäuseform eine Auswahl zwischen verschieden großen Bausteinen ermöglicht wird und damit eine Anpassung des FPGAs an die Komplexität der Aufgabenstellung. Die Benutzung eines PCI-*Masterchips* zur Anbindung an den PCI-Bus ermöglicht die schnelle Kommunikation und das Ansprechen des FPGAs über DMA-Zugriffe. Die Logik des FPGAs wird in einer komfortablen VHDL-Umgebung programmiert und simuliert und ist über den CPLD jederzeit neu konfigurierbar. Mit der Ansteuerung von lokalem Speicher, analoger Peripherie und nicht zuletzt des zu testenden ASICs kann so auf ein und derselben Hardware ein Testsystem für verschiedene mixed-signal ASICs aufgebaut werden. Dies ermöglicht die Durchführung auch komplexer Testabläufe, ohne den Prozessor des PCs zu benutzen.

Neben den funktionalen Komponenten enthält die PCI-Karte auch alle Bauteile, die nötig sind, um deren elektrische Spezifikationen zu erfüllen. Dies sind die PLL für eine flexible Taktgenerierung, die 1.8 V und 2.5 V-Netzteile für den FPGA und die ± 5 V-Netzteile für die analogen Bauteile. Hinzu kommen Treiberbausteine und passive Komponenten. In Kapitel 3 wird erläutert, wie die insgesamt 232 Bauteile zu einem Schaltplan zusammengefügt und als komplexe Leiterplatte auf acht Lagen realisiert wurden. Die Anordnung der Bauteile lässt kurze Verbindungen zu, und die Aufnahme der zu testenden ASICs auf einer CMC-Karte bietet Offenheit für verschiedenste Erweiterungen. Durch den Lagenaufbau mit Potentialflächen in der Platine werden die analogen Bauteile auf der Rückseite vom digitalen Layout getrennt, um eine optimale Signalqualität zu erreichen.

Die Inbetriebnahme der ersten beiden Prototypen bestätigt die volle Funktionsfähigkeit aller Komponenten. Erste Messergebnisse zeigen sowohl die Qualität der

analogen Signale als auch exakte Signalformen ohne Artefakte von Reflexionen auf den digitalen Bussen. Die kontrollierte Impedanz der LVDS-Leitungen erlaubt eine bisher getestete Datenübertragungsrate von 200 MBit pro Sekunde pro Signal, und die Steilheit der Flanken verspricht weitere Geschwindigkeitssteigerungen beim Übergang zu schnelleren FPGAs.

Unter Verwendung der Adapterplatine *Gosalyn* konnten Experimente vom bisherigen Testsystem [Bli00] erfolgreich auf *Darkwing* portiert werden. Dabei können Trägerplatinen bisheriger ASICs auf *Gosalyn* aufgesteckt und mit wenigen Anpassungen der FPGA-Anschlusszuordnung im VHDL-Code in Betrieb genommen werden. Somit kann die bisherige Hard- und Software weiter benutzt werden. Dies gilt für die aktuellen Projekte der FPTA, EDDA und EVOOPT-ASICs. Darüber hinaus nutzt das ANNA-Projekt bereits die neuen Möglichkeiten der CMC-Stecker und hat mit dem *Hoverboard* eine eigene LVDS-fähige Adapterplatine für die Trägerplatinen. Zu den neuen Projekten zählt auch die Eingliederung einer SD-RAM Ansteuerung in das bestehende VHDL-Modul der S-RAM Ansteuerung, um eine intern einheitliche Kommunikation mit dem Speicher zu erhalten.

Da die bisher portierten Experimente nur wenig Platz im FPGA einnehmen, sind die Voraussetzungen gut, um durch Erweiterung der bestehenden Programme die neuen Funktionen sowohl des FPGAs als auch der externen Komponenten zu nutzen. Im FPTA-Projekt bedeutet dies den Ausbau der in Kapitel 5 entwickelten *state machines* derart, dass unter Benutzung des lokalen RAMs komplette Generationen von Individuen eines genetischen Algorithmus automatisch in den ASIC geladen und mit den Analogbauteilen getestet und bewertet werden. Voraussichtlich wird es mit den benutzbaren FPGAs auch möglich sein, den genetischen Algorithmus selbst im FPGA zu verwirklichen, und dies wird erst den erhofften Sprung in der Leistungsfähigkeit des Evolutionssystems bringen. Erst wenn es gelingt, für die Karte diese Stufe der Autonomie zu erreichen, wird die Geschwindigkeit des Experiments nicht mehr durch die Bandbreite des PCI-Busses und die Langsamkeit des PCs dominiert sondern kann lokal auf dem Testsystem voll mit der Geschwindigkeit der Hardware ablaufen und wesentlich höhere Evolutionsraten erzielen.

Mit dem Aufbau der vorliegenden PCI-Karte ist also die Möglichkeit geschaffen, ein Test- und Evolutionssystem für konfigurierbare gemischt analog/digitale ASICs aufzubauen, das die Vorteile von schneller Hardware und komfortabler PC-Software vereint. Die Anpassungsfähigkeit und Erweiterbarkeit stellen sicher, dass das hier entwickelte Testsystem auch für zukünftige Projekte von Nutzen sein wird.

Anhang A

EEPROM für PLX

```
9080 device ID
10b5 vendor ID
0680 class code
0002 class code, revision

4008 max lat, min grant
0100 irq A
1111 MSW mailbox 0
2222 LSW

3333 MSW mailbox 1
4444 LSW
ffc0 MSW loc. adr. spc. 0 range reg PCI->Local
0000 LSW 4 MB range reduced from 64 due to problems!!!

1000 MSW loc. adr. spc. 0 local base adr PCI->Local
0001 LSW enable!
1800 MSW loc. arbitration
0000 LSW

0060 MSW serial eeprom write protect
0200 LSW local misc: init done /endian reg:all little endian
ffff MSW expansion rom range
0000 LSW

0100  MSW exp. rom rmap
0000  LSW
4003  MSW local address space 0 region descriptor
#4203 MSW local address space 0 region descriptor with extended eeprom
0303  LSW

0000 MSW direct master range
0000 LSW
0000 MSW memory rng.
0000 LSW

0000 MSW i/o rng.
0000 LSW
0000 MSW master remap
0000 LSW

0000 MSW config address dir. master
0000 LSW
```

Anhang B

VHDL Modell für den FPTA

```
-- ------------------------------------------------------------
--
-- fpta model, J.Becker
--
--
-- chip version: fpta
-- for xpc104 board
--
-- ------------------------------------------------------------
-- including:
-- minimal ram write access
--


library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;--synopsys

library work;
use work.all;
use work.edda_global.all;

-- ------------------------------------------------------------ --
--      ENTITY fpta_model
-- ------------------------------------------------------------ --
entity fpta_model is
port(
  fpta_a :in std_logic_vector(9 downto 0);
  fpta_d :inout std_logic_vector(5 downto 0);
  fpta_sam :in std_logic_vector(3 downto 0);
     readen   :in std_logic;
     clk1 :in std_logic;
  clk2       :in std_logic;
  prech      :in std_logic;
  chip_sel   :in std_logic;
     resetf     :in std_logic;
     sh_en      :in std_logic;
  aprobe_en  :in std_logic;
  buf_pd     :in std_logic;
     ar_en      :in std_logic;
  we         :in std_logic;
     aw_en      :in std_logic;

  -- clk for simulation only
  clk : in std_logic
) ;
end fpta_model;



-- ------------------------------------------------------------ --
--      ARCHITECTURE declaration
-- ------------------------------------------------------------ --
architecture f_mod of fpta_model is

--fpta test storage
type  fpta_word is array (natural range <>) of std_logic_vector(95 downto 0) ;
signal  fpta_oneword  : std_logic_vector(95 downto 0) ;
signal  fpta_sram : fpta_word(63 downto 0) ;
type STATES is (IDLE, WRITE1, WRITE2, READ0, READ1, READ2, READ3, READ4, READ5, READ6, READ7, READ8) ;
signal ns : STATES;
```

```
--signal fpta_din : std_logic_vector(5 downto 0) ;
--signal fpta_dout : std_logic_vector(5 downto 0) ;

--------------------------------------------------------------------------------
-- FUNCTION :
--   TO_INT (oper : STD_LOGIC_VECTOR) return int
--     INPUT : A STD_LOGIC_VECTOR
--     INPUT : An integer value representing that STD_LOGIC_VECTOR
--------------------------------------------------------------------------------
    function my_word (oper: STD_LOGIC_VECTOR(9 downto 0) ) return INTEGER is
        variable temp : INTEGER := 0;
    begin
        for i in 0 to 5 loop
            if oper(i) = '1' then
                temp := temp + 2**i ;
            end if ;
        end loop;
        return temp;
    end my_word;

 function my_row (oper: STD_LOGIC_VECTOR(9 downto 0) ) return INTEGER is
        variable temp : INTEGER := 0;
    begin
        for i in 6 to 9 loop
            if oper(i) = '1' then
                temp := temp + 2**(i-6) ;
            end if ;
        end loop;
        return temp;
    end my_row;



begin

-- generate simulated fpta output data
stateloop : process (clk, clk1, clk2, prech, aw_en, ar_en, readen, we, resetf)
variable adress : integer ;
begin
if resetf='0' then
ns <= IDLE ;
fpta_d <= (others=>'Z');
else
case ns is
when IDLE =>
--fpta_d <= (others=>'Z');
if ar_en'event and ar_en='0' then
ns <= WRITE1 ;  -- get input buffer of one row (6 bit)
end if ;

if we'event and we='1' then
ns <= WRITE2 ;  -- store input buffer of one word (all rows) into fpta-ram
end if ;

if clk1'event and clk1='1' and prech'event and prech='0' then
ns <= READ1 ;  -- read out fpta-ram
end if ;

when WRITE1 =>
adress := my_row(fpta_a) ;
fpta_oneword(adress*6+5 downto adress*6) <= fpta_d(5 downto 0)  ;
ns <= IDLE ;
when WRITE2 =>
if aw_en'event and aw_en='0' then
   adress := my_word(fpta_a) ;
fpta_sram(adress) <= fpta_oneword ;
ns <= IDLE ;
end if ;
when READ1 =>
if  prech'event and prech='1' then
ns <= READ2 ;
end if ;
when READ2 =>
if  aw_en'event and aw_en='0' then
ns <= READ3 ;
end if ;
when READ3 =>
if  clk2'event and clk2='1' then -- get adress of word
adress := my_word(fpta_a) ;
fpta_oneword <= fpta_sram(adress)  ;
ns <= READ4 ;
end if ;
when READ4 =>
if  aw_en'event and aw_en='1' and clk2='1'then
ns <= READ5 ;
end if ;
when READ5 =>
```

```
if clk1'event and clk1='0' and clk2='1'then
ns <= READ6 ;
end if ;
when READ6 =>
if readen'event and readen='1' and clk2='1'then
ns <= READ7 ;
end if ;
when READ7 =>
if  readen='1' and clk2='1' and ar_en'event and ar_en='0' then  --readout loop (each 6 bits)
adress := my_row(fpta_a) ;
fpta_d(5 downto 0)  <= fpta_oneword(adress*6+5 downto adress*6) ;
ns <= READ8 ;
elsif readen='0' then ns <= IDLE ;
end if ;
when READ8 =>
if  readen='1' and clk2='1' and ar_en'event and ar_en='1' then
ns <= READ7 ;
end if ;
when others => ns <= IDLE ;
end case ;
end if ;
end process stateloop;

end f_mod;

-- ---------------------------------------------------------

configuration model of fpta_model is
for f_mod
end for;
end model;
```

Anhang C

Schaltplan

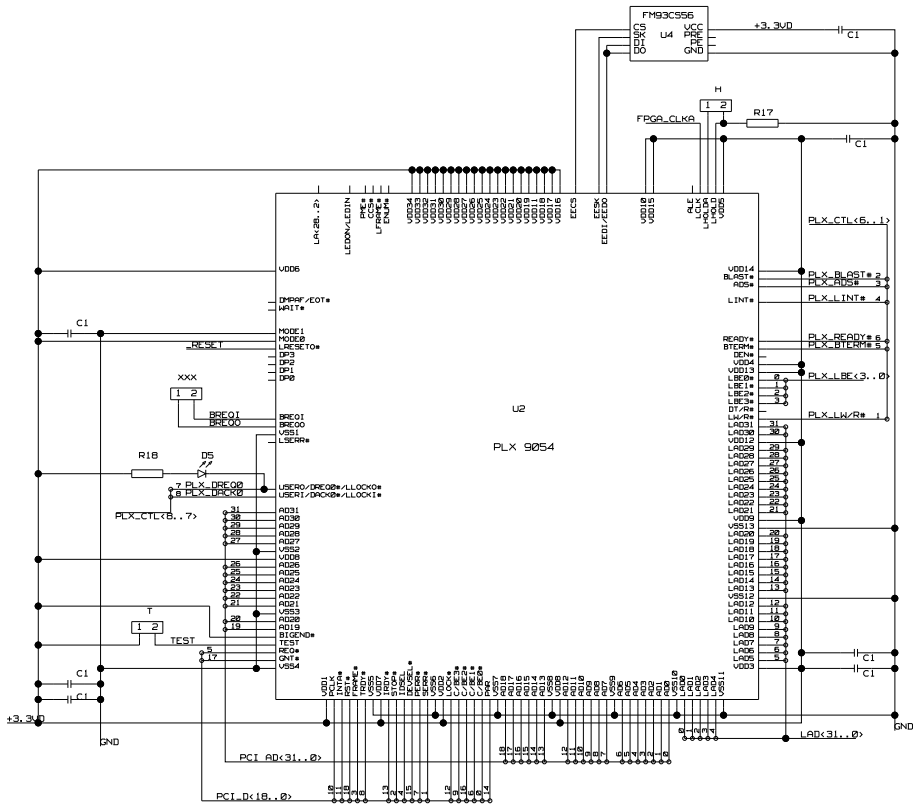
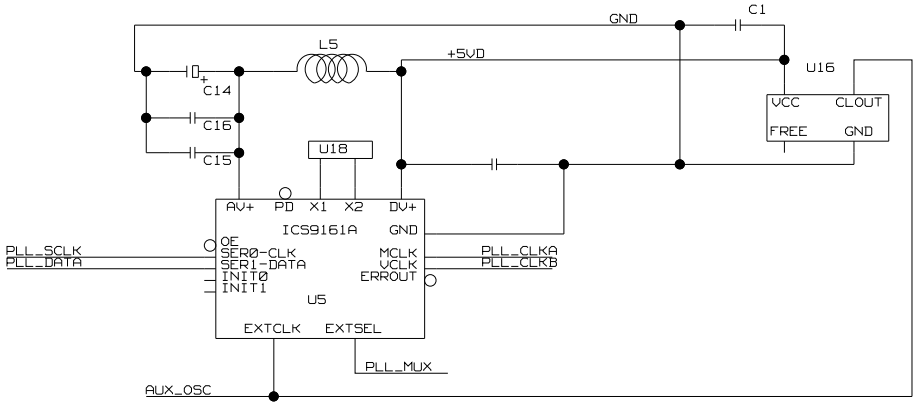


Abbildung C.1: Schaltplan PLX



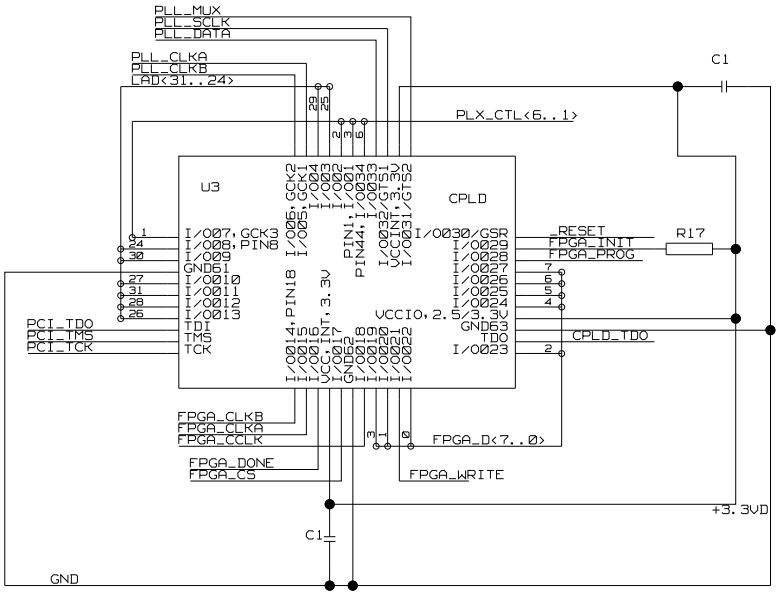


Abbildung C.2: Schaltplan CPLD und PLL

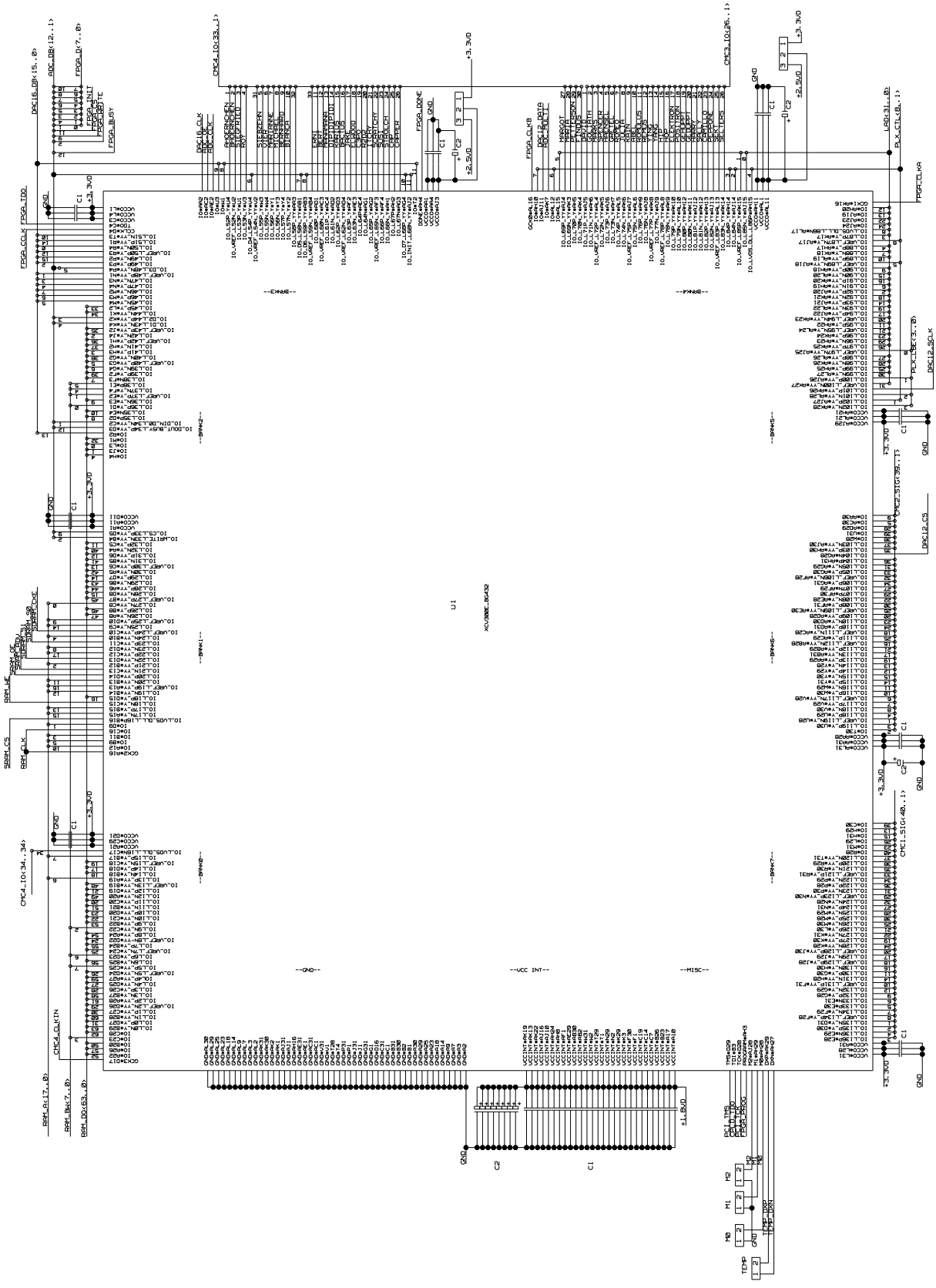


Abbildung C.3: Schaltplan FPGA

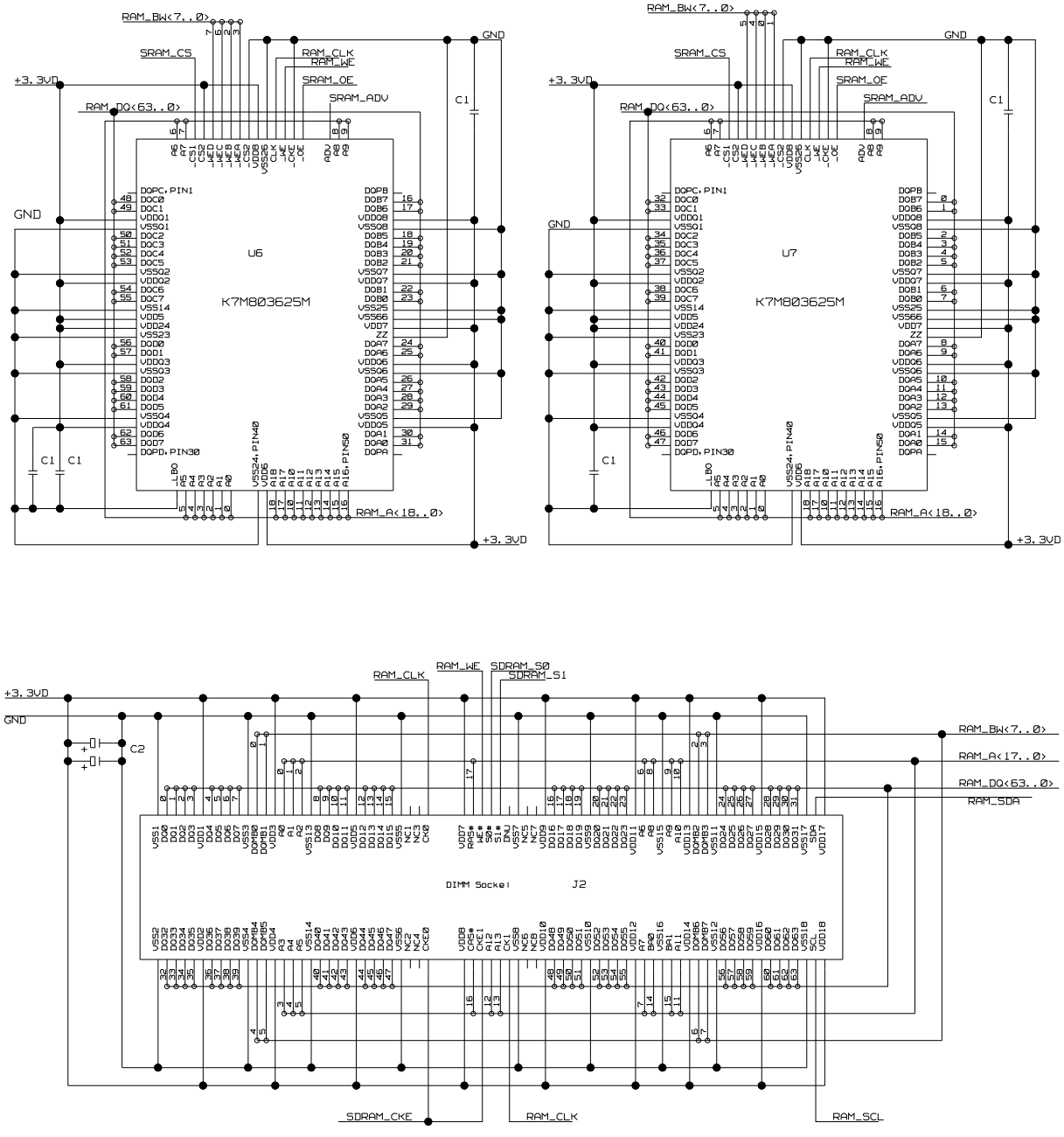


Abbildung C.4: Schaltplan RAM

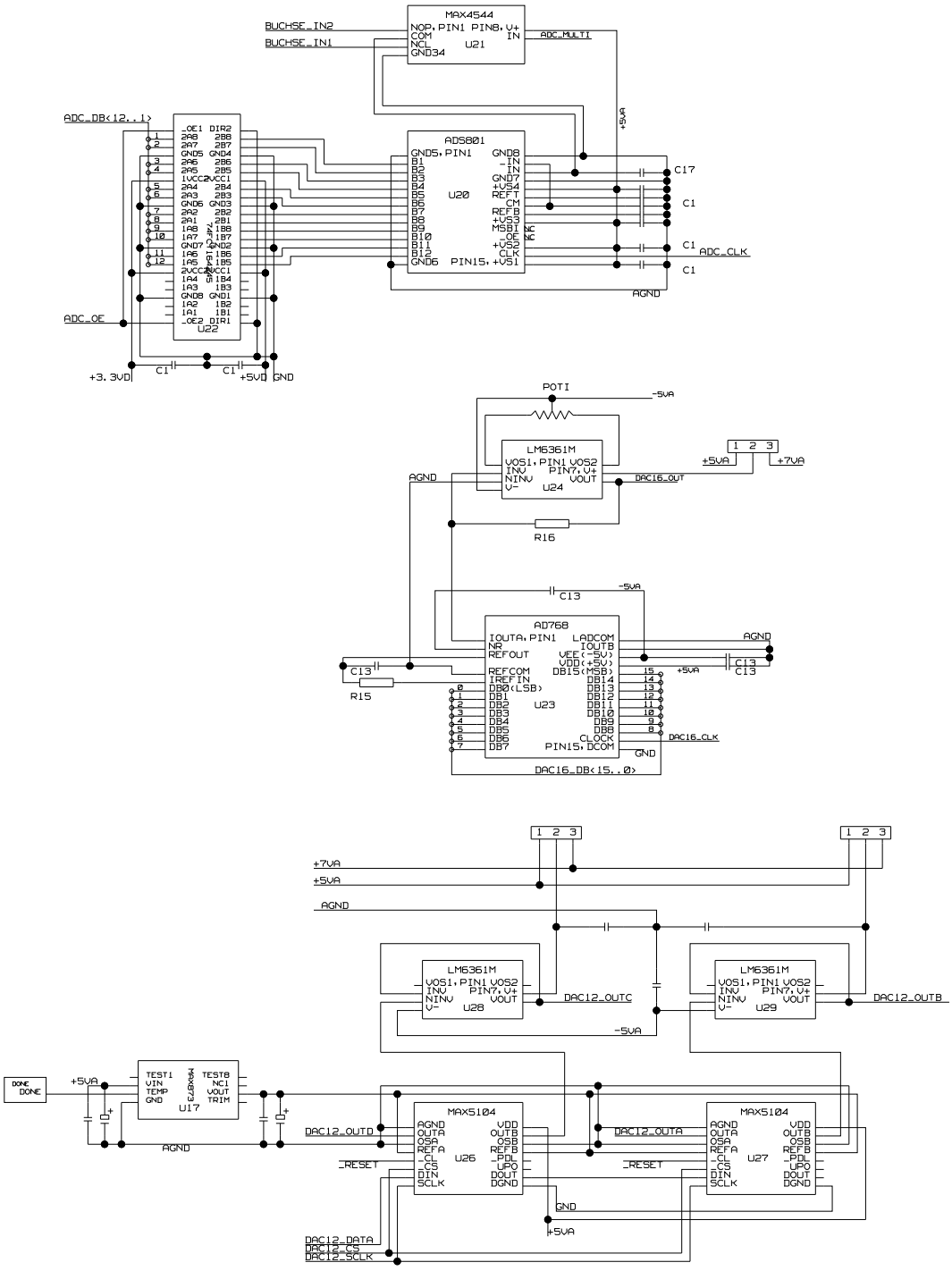


Abbildung C.5: Schaltplan Analogteil

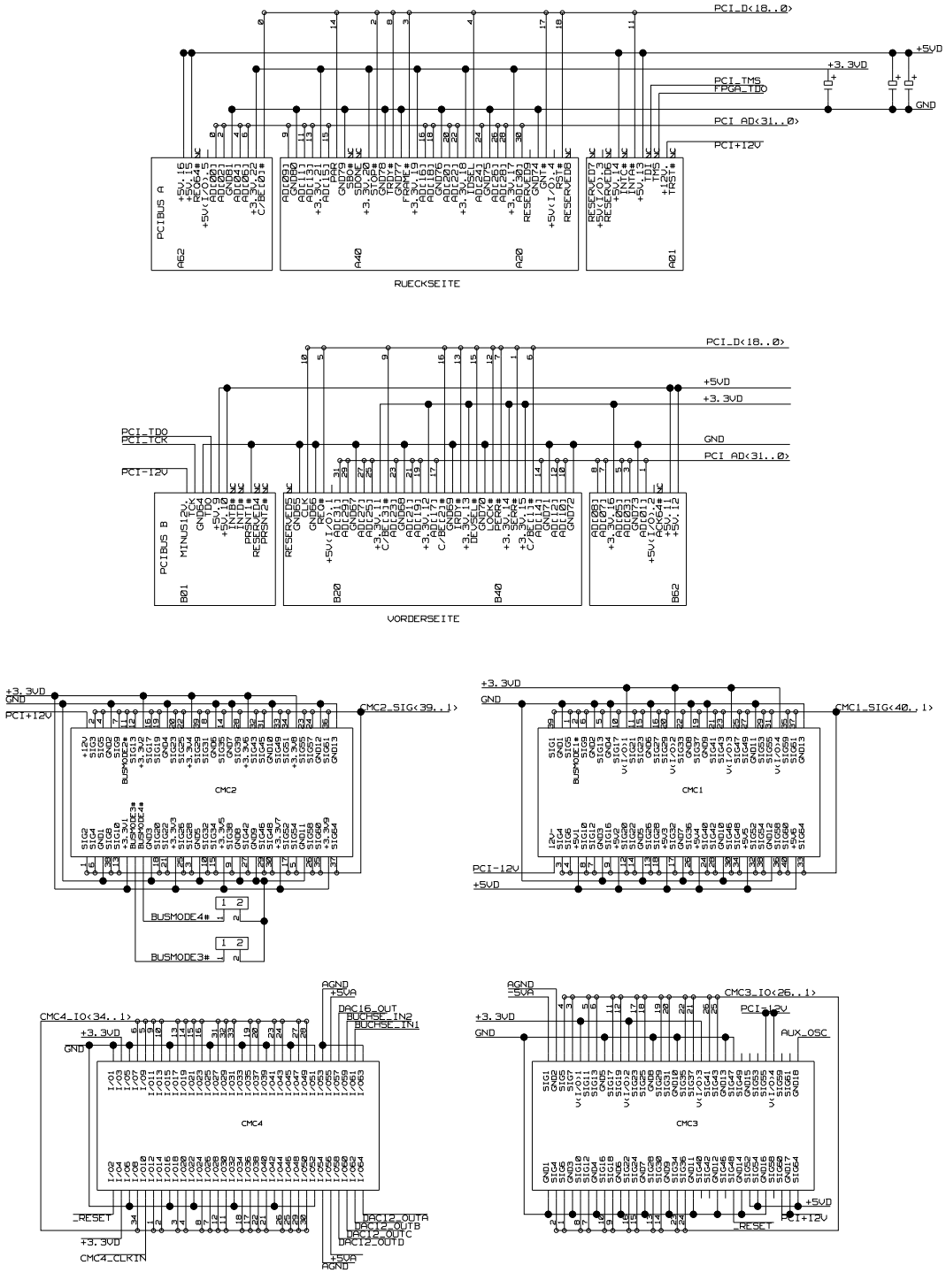


Abbildung C.6: Schaltplan Stecker

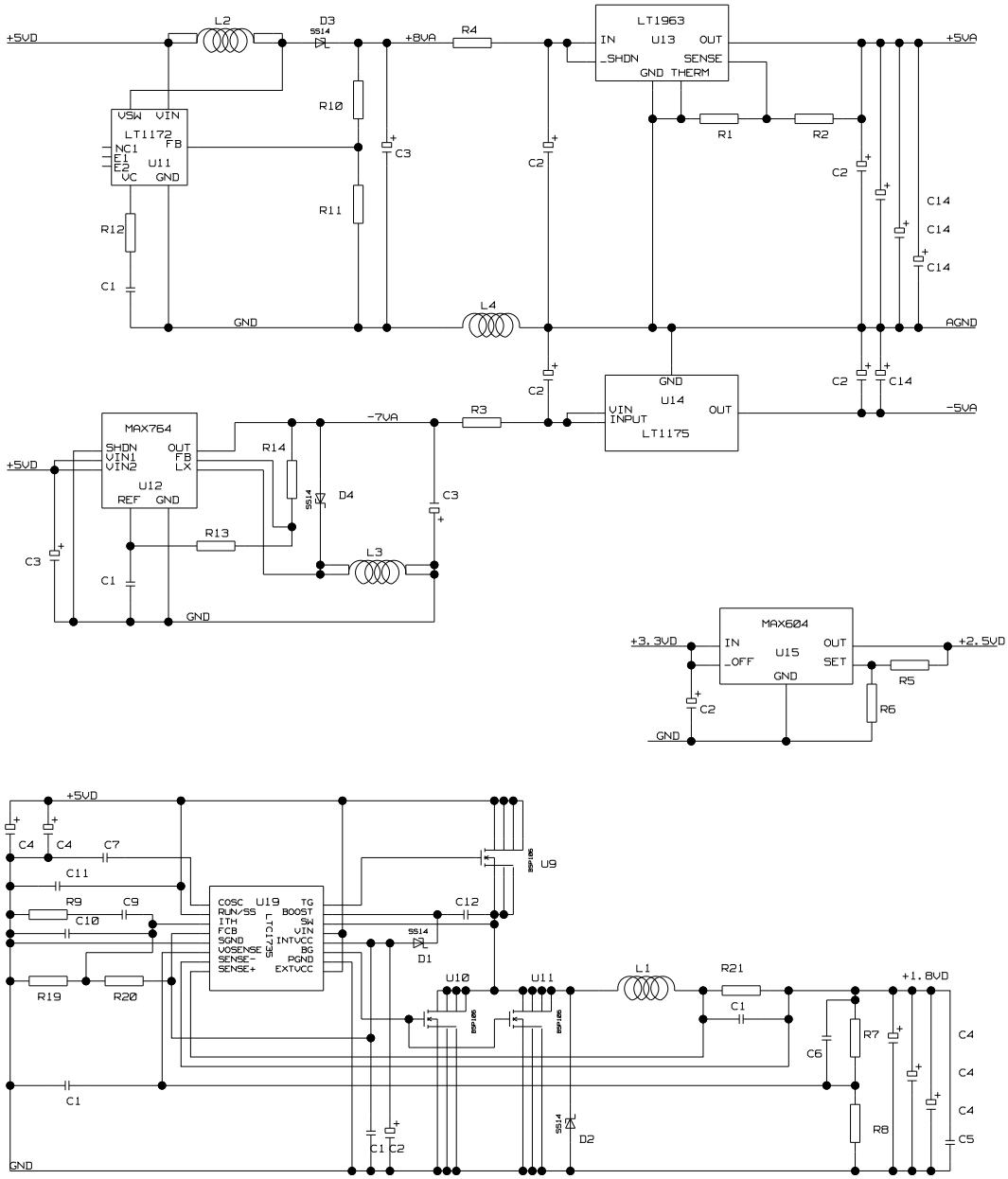


Abbildung C.7: Schaltplan Netzteile

Anhang D

Bestückung

Ref	JEDEC	Wert	Anzahl	Bemerkung
C1	SMD0603	0.1 μF, Keramik	ca. 60	Block Keramik
C2	CT7343_D	100 μF, 10V	16	Block Tantal
C3	CT7343_D	220 μF, 10V, low ESR	5	Schaltregler
C4	CT7343_D	330 μF, 6.3V, low ESR	3	1.8V-Regler
C5	SMD2220	47 μF, Keramik	1	Tayko Yuden
C6	SMD0603	100 pF	1	
C7	SMD0603	47 pF	1	
C8	SMD0805	0.1 μF	1	
C9	SMD0603	470 pF	1	
C10	SMD0603	220 pF	1	
C11	SMD0603	47 pF	1	
C12	SMD0603	1 nF	1	
C13	SMD0805	1 μF, Tantal	4	DAC16 (U23)
C14	CT3216_A	22 μF	7	Block analog
C15	SMD0603	0.1 nF	1	
C16	SMD0603	10 nF	2	
C17	SMD0805	22 pF	1	
R1	SMD0805	1.2 kΩ	1	LT1963 R1
R2	SMD0805	3.9 kΩ	1	LT1963 R2
R3	SMD2512	4.7 Ω, 1W	1	- 7V
R4	SMD2512	2.2 Ω, 1W	1	+ 8V
R5	SMD0805	5.6 kΩ	1	MAX604 R1
R6	SMD0805	5.1 kΩ	1	MAX604 R2
R7	SMD0805	160 kΩ	1	LTC1735 R2
R8	SMD0805	130 kΩ	1	LTC1735 R1
R9	SMD0805	33 kΩ	1	R-c
R10	SMD0805	6.8 kΩ	1	LT1172
R11	SMD0805	1.2 kΩ	1	LT1172

R12	SMD0805	1 kΩ	1	LT1172
R13	SMD0805	680 kΩ	1	MAX764
R14	SMD0805	150 kΩ	1	MAX764
R15	SMD0805	510 Ω	1	DAC16
R16	SMD0805	200 Ω	1	DAC16 OP
R17	SMD0805	2.2 kΩ	1	Pullup LHOLD, INIT
R18	SMD0805	150 Ω	1	LED Vorwiderstand
R19	SMD0805		1	Activ Voltage Position
R20	SMD0805		1	Activ Voltage Position
R21	SMD0805	5 mΩ, 10A	1	Drahtbrücke
P1		100 kΩ	1	Poti
D1	SMDfett	2A, Shottky	1	1.8V
D2	SMDfett	2A, Shottky	1	1.8V
D3	SMDfett	1A, Shottky	1	+8V
D4	SMDfett	1A, Shottky	1	-7V
D5	SMD0805	LED	1	
L1	DO022P-152HC	1.5 μH, 10A	1	1.8V
L2	CTX50-3	50 μH, 1A	1	+8V
L3	CTX50-2/1	50 μH, 0.5A	1	-7V
L4	radial	10 μH	1	GND-Drossel
L5	SMD0805	0.47 μH	1	Induktor PLL
U1	BG234	XCV300E - XCV600E	1	FPGA
U2	PBGA225	PLX-PCI9054-AB50-BI	1	PCI-Interface
U3	CLCC44	XC 9536/9572 XL	1	CPLD
U4	DIP8	NM93CS56	1	EEPROM
U5	SOIC16W	ICS9161A	1	PLL
U6	TQFP100	K7M803625M-QC85	1	RAM
U7	TQFP100	K7M803625M-QC85	1	RAM
U8	SOIC8	SI4886DY	1	MOSFETS
U9	SOIC8	SI4886DY	1	MOSFETS
U10	SOIC8	SI4886DY	1	MOSFETS
U11	SOIC8	LT1172CS8	1	+8V, 1.5A
U12	SOIC8	MAX764CSA	1	-7V, 250mA
U13	Q5	LT1963EQ	1	+5V, 1A
U14	SOT223	LT1175CST5	1	-5V, 250mA
U15	SOIC8	MAX604CSA	1	+2.5V, LVDS
U16	DIP14	Oszillator (optional)	1	
U17	SOIC8	MAX873ACSA	1	Referenz 2.5V
U18	HC49, SMD	16MHz	1	Quarz
U19	SSOP16	LTC1735CGN	1	1.8V Regler
U20	SOIC28W	ADS800U / ADS801U	1	ADC
U21	SOIC8	MAX4544CSA	1	Analog Schalter

U22	TSSOP48	74FCT164245	1	Translator Buffer
U23	SOIC28	AD768AR	1	DAC16
U24	SOIC8	LM6361M	1	OP für 16bit DAC
U26	SSOP16	MAX5104EEE	1	DAC12
U27	SSOP16	MAX5104EEE	1	DAC12
U28	SOIC8	LM6361M	1	OP für 12bit DACs
U29	SOIC8	LM6361M	1	OP für 12bit DACs
J1	3*2mm		2	Stifte
J2	SO DIMM 144	AMP390114-1	1	SD-RAM Sockel

Bestückungsplan

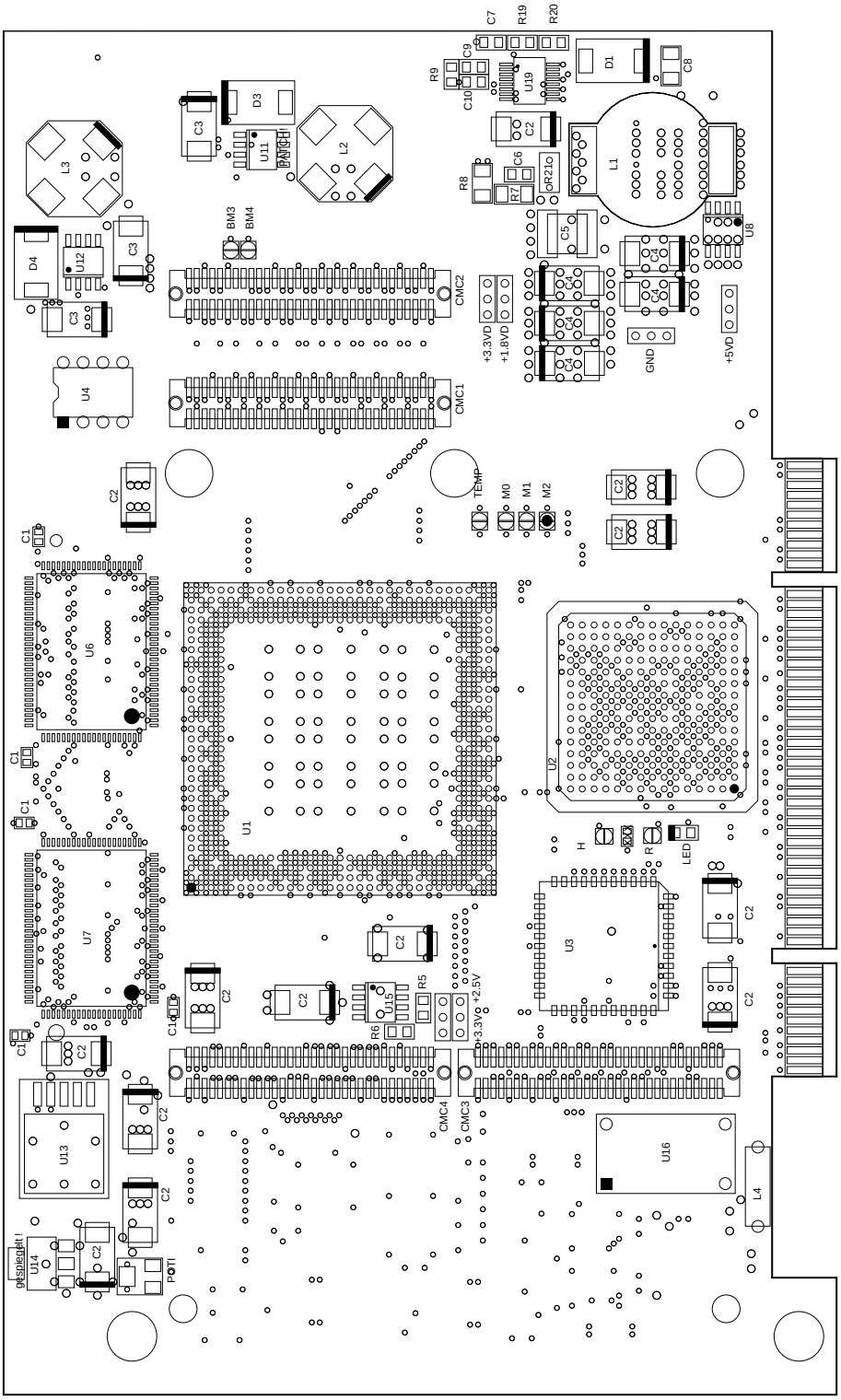


Abbildung D.1: Bestückungsplan Oberseite

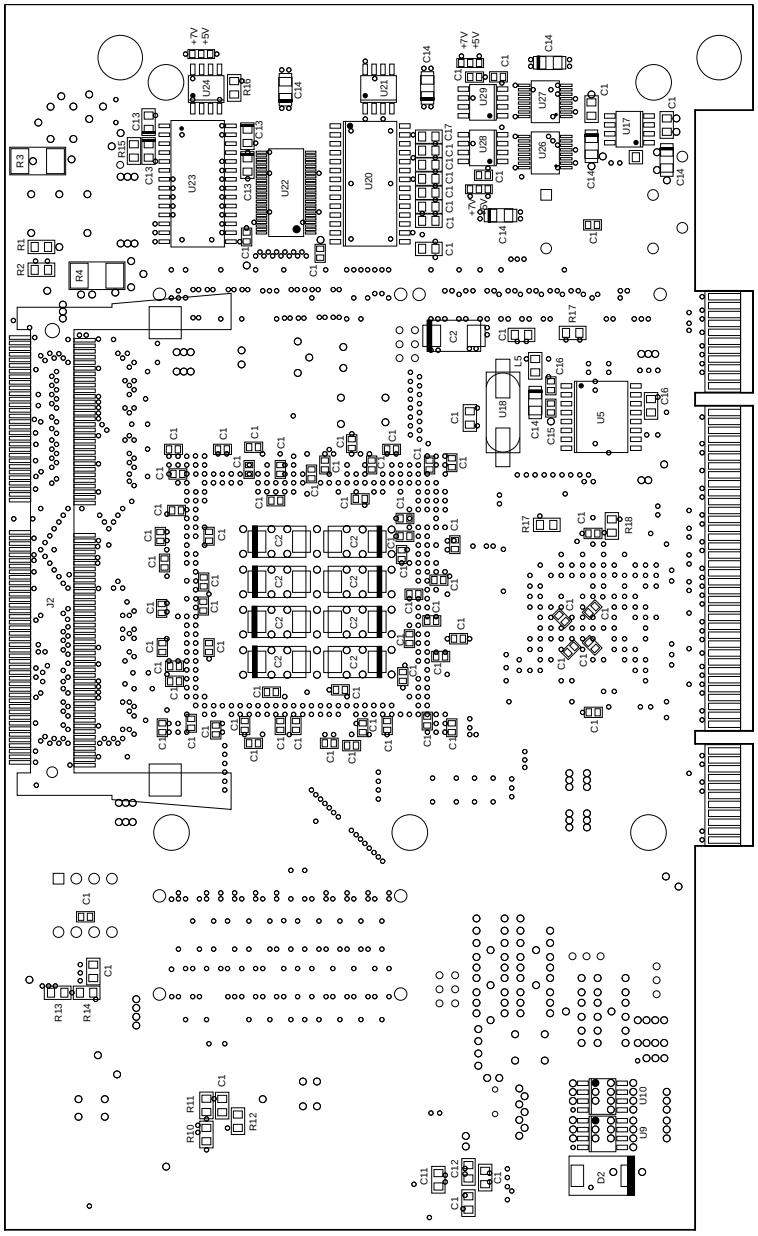


Abbildung D.2: Bestückungsplan Unterseite

Fotos

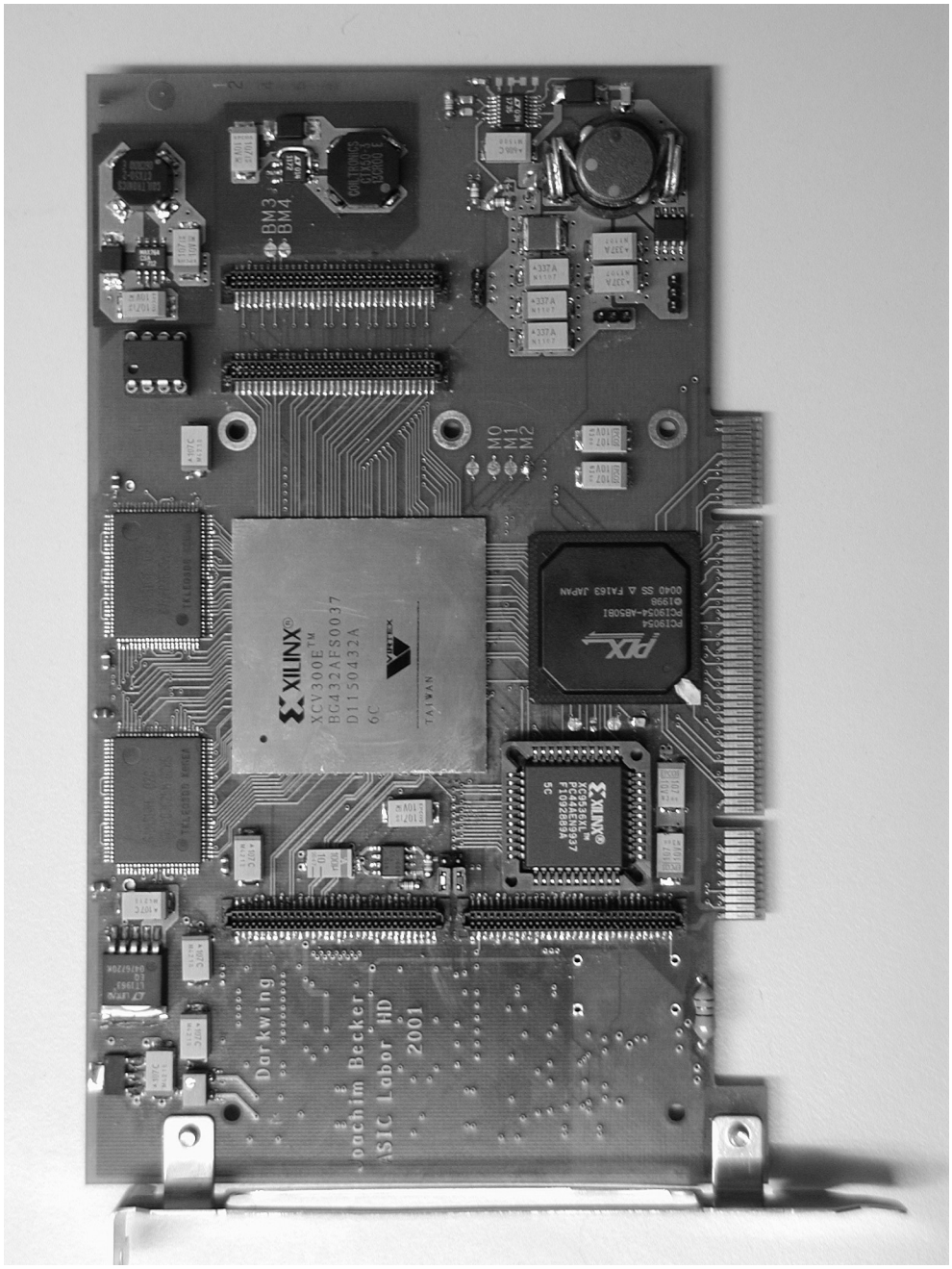


Abbildung D.3: Oberseite

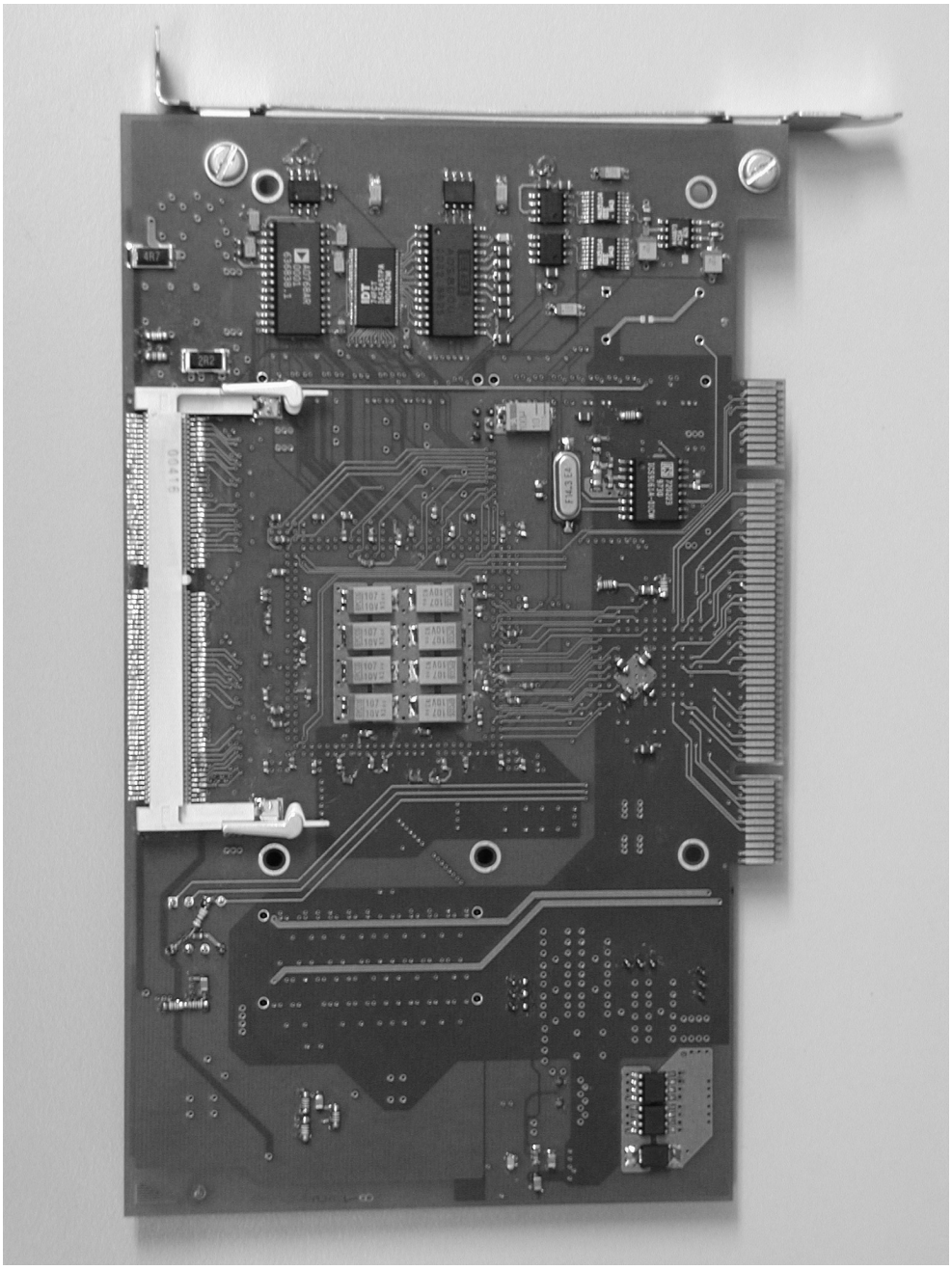


Abbildung D.4: Unterseite

Anhang E

Lagenpläne

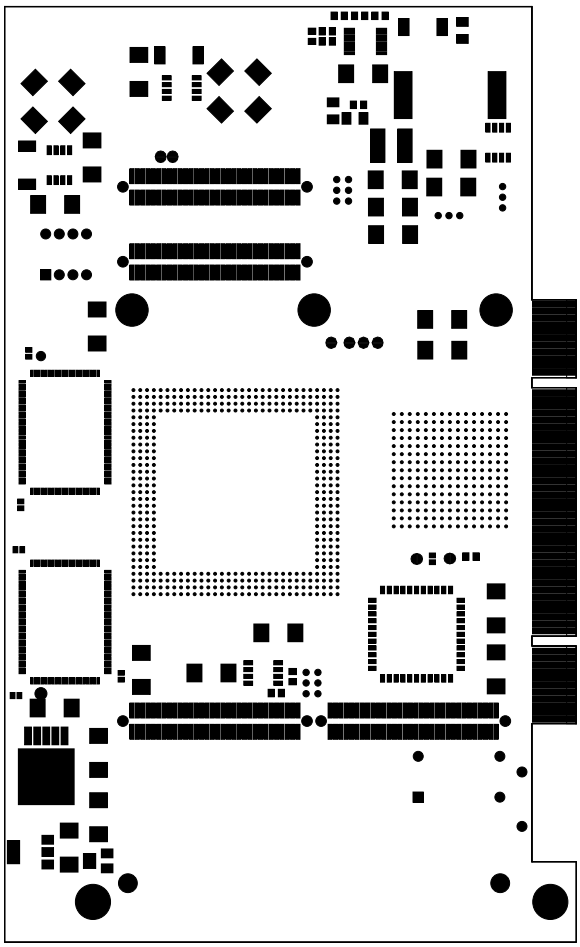


Abbildung E.1: Lötstopplack Oberseite

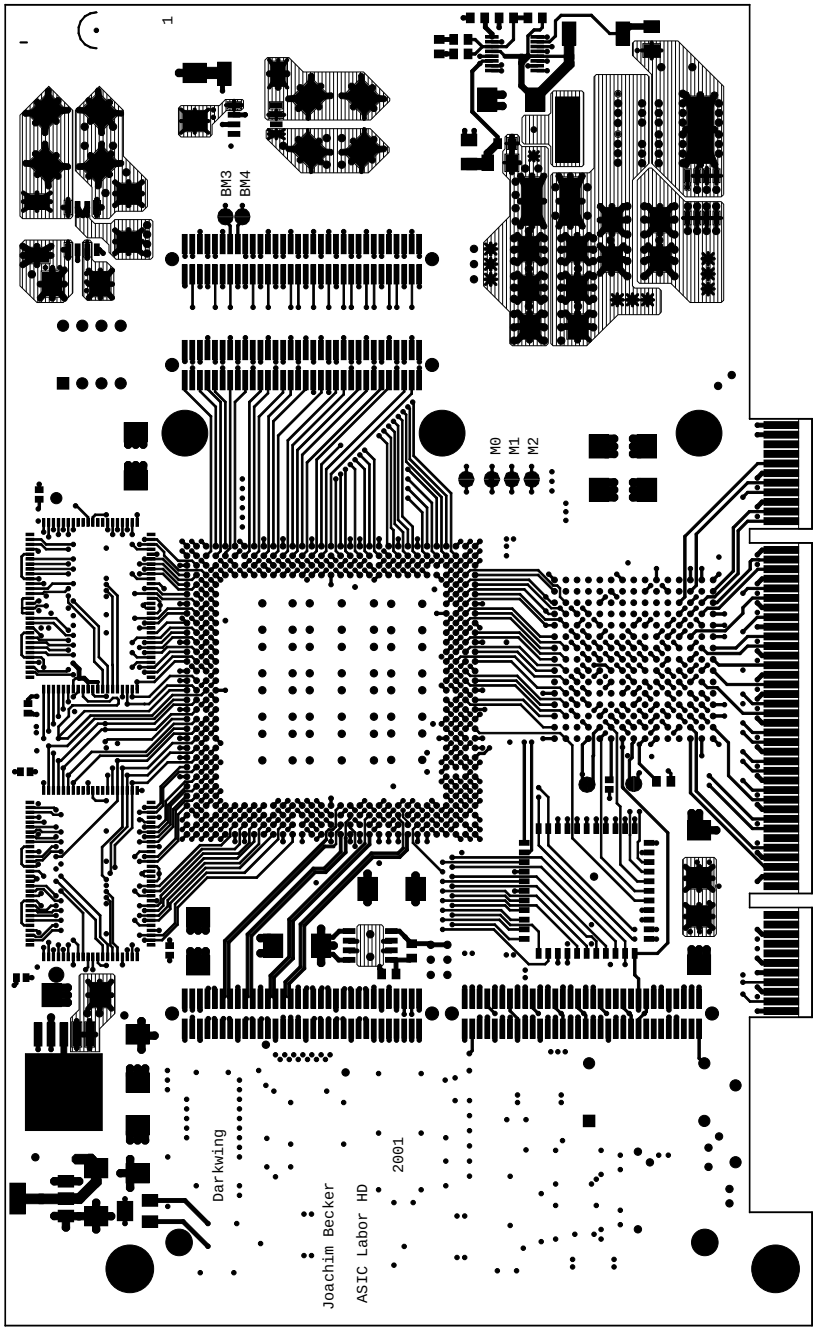


Abbildung E.2: Oberseite

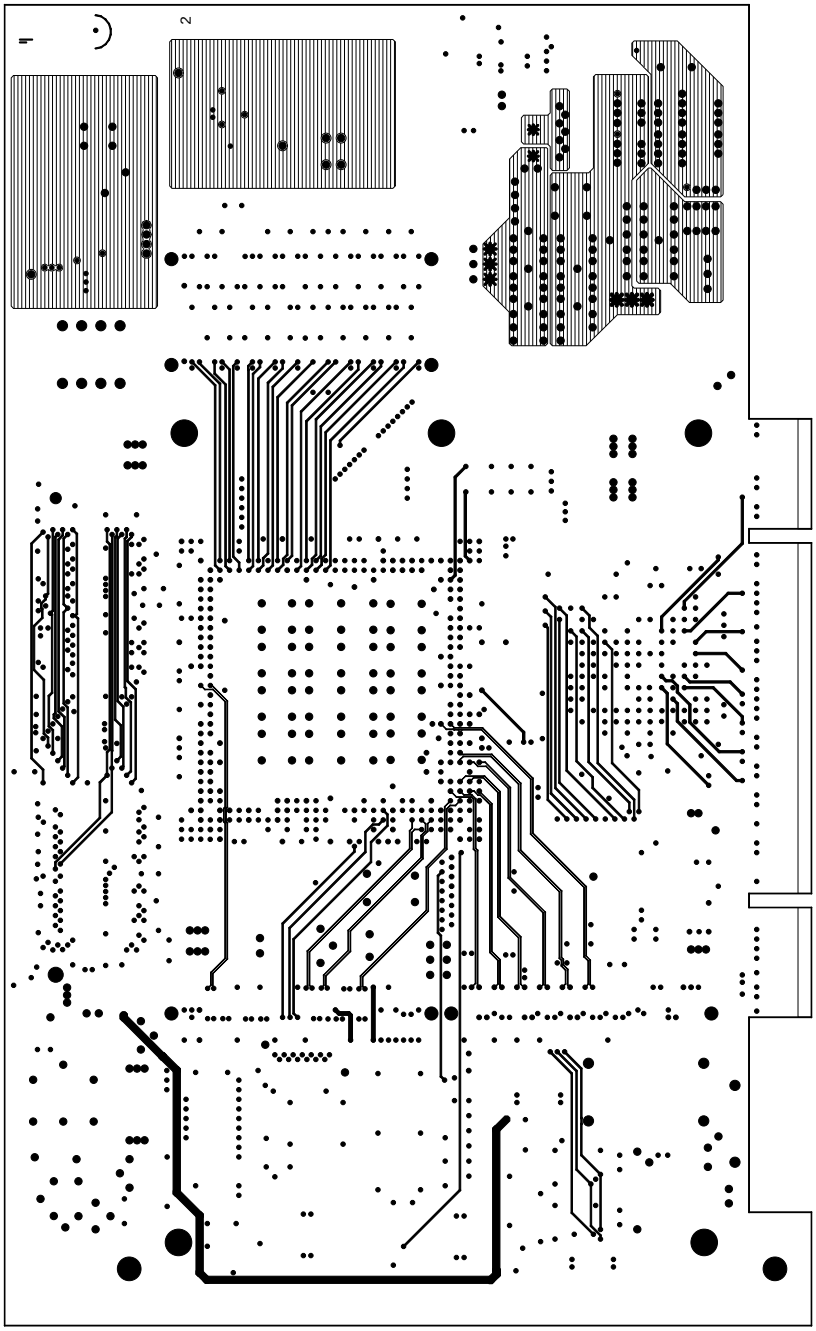


Abbildung E.3: Lage 2

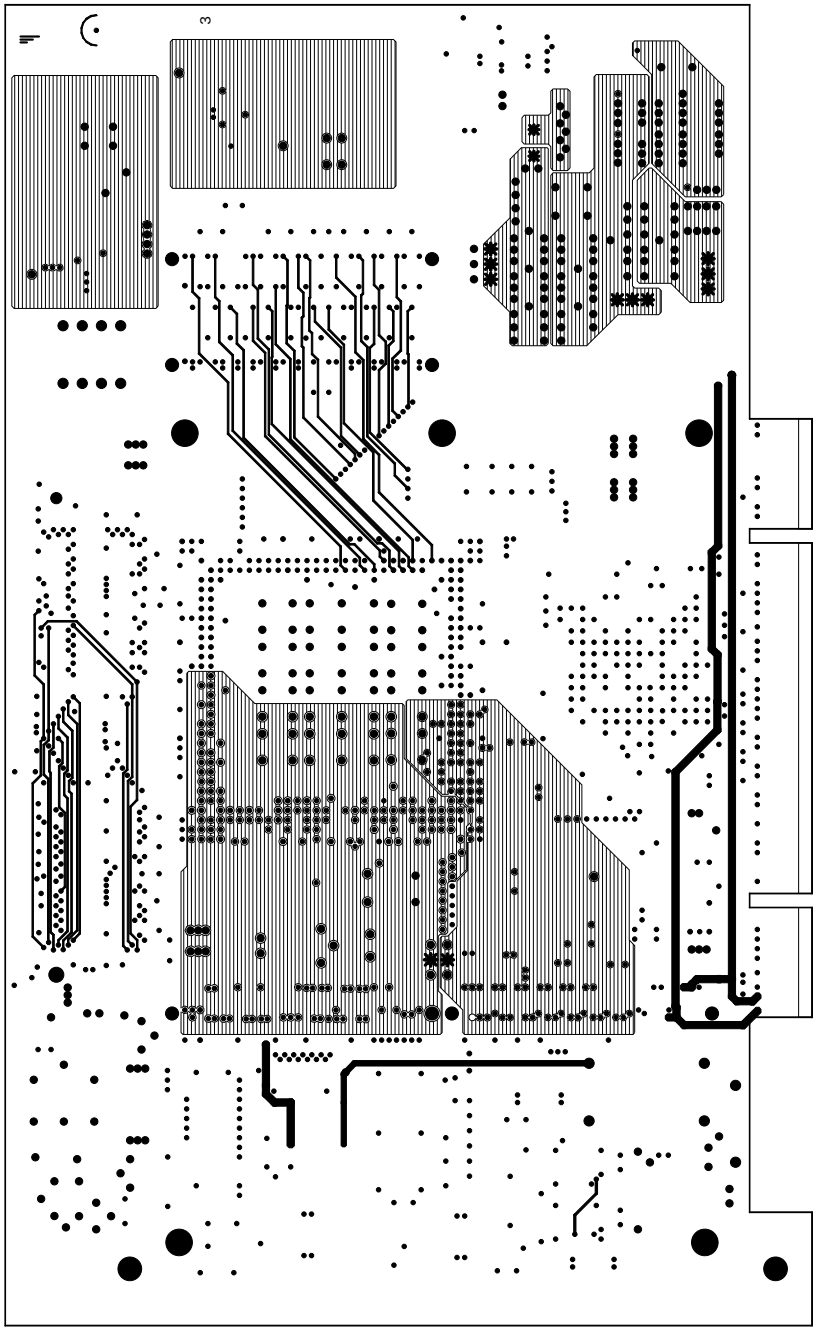


Abbildung E.4: Lage 3

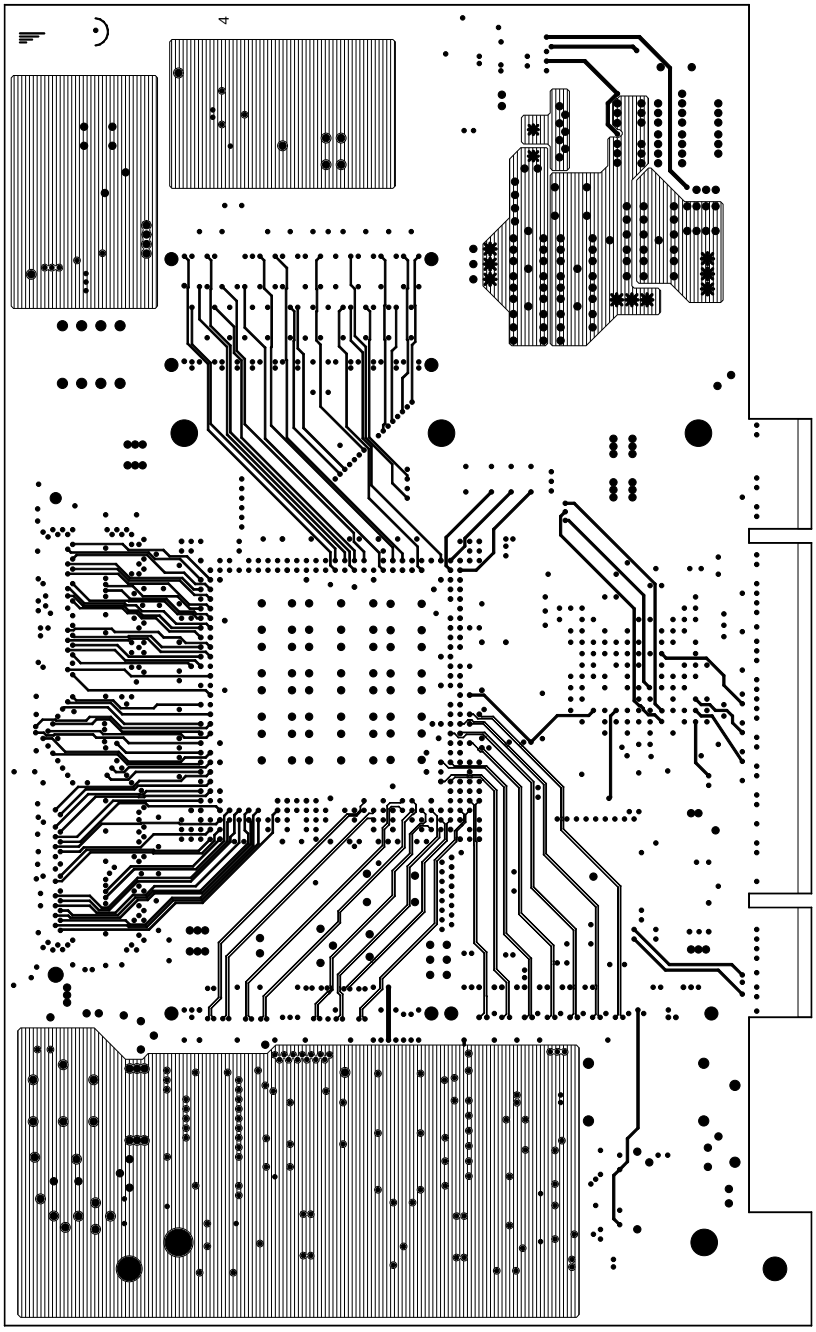


Abbildung E.5: Lage 4

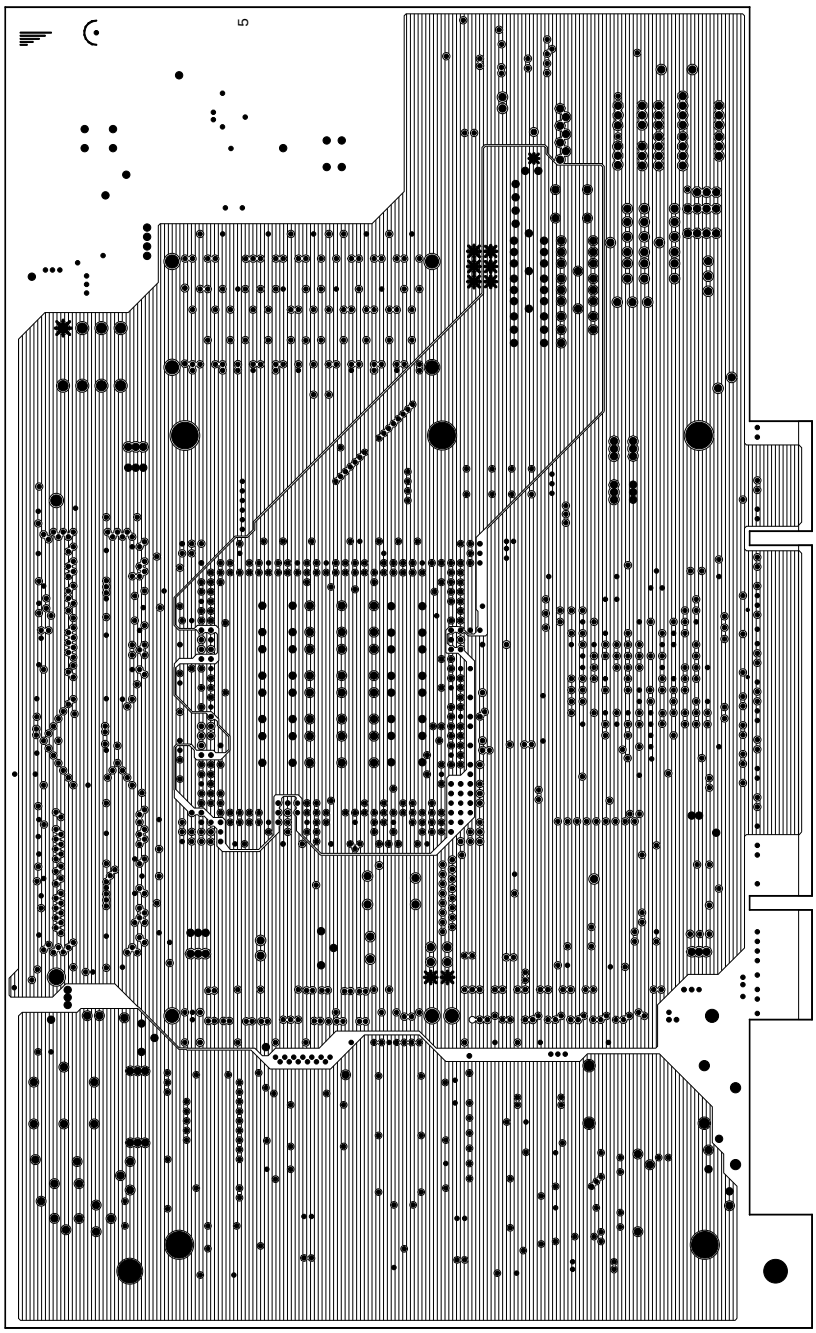


Abbildung E.6: Lage 5 / Power

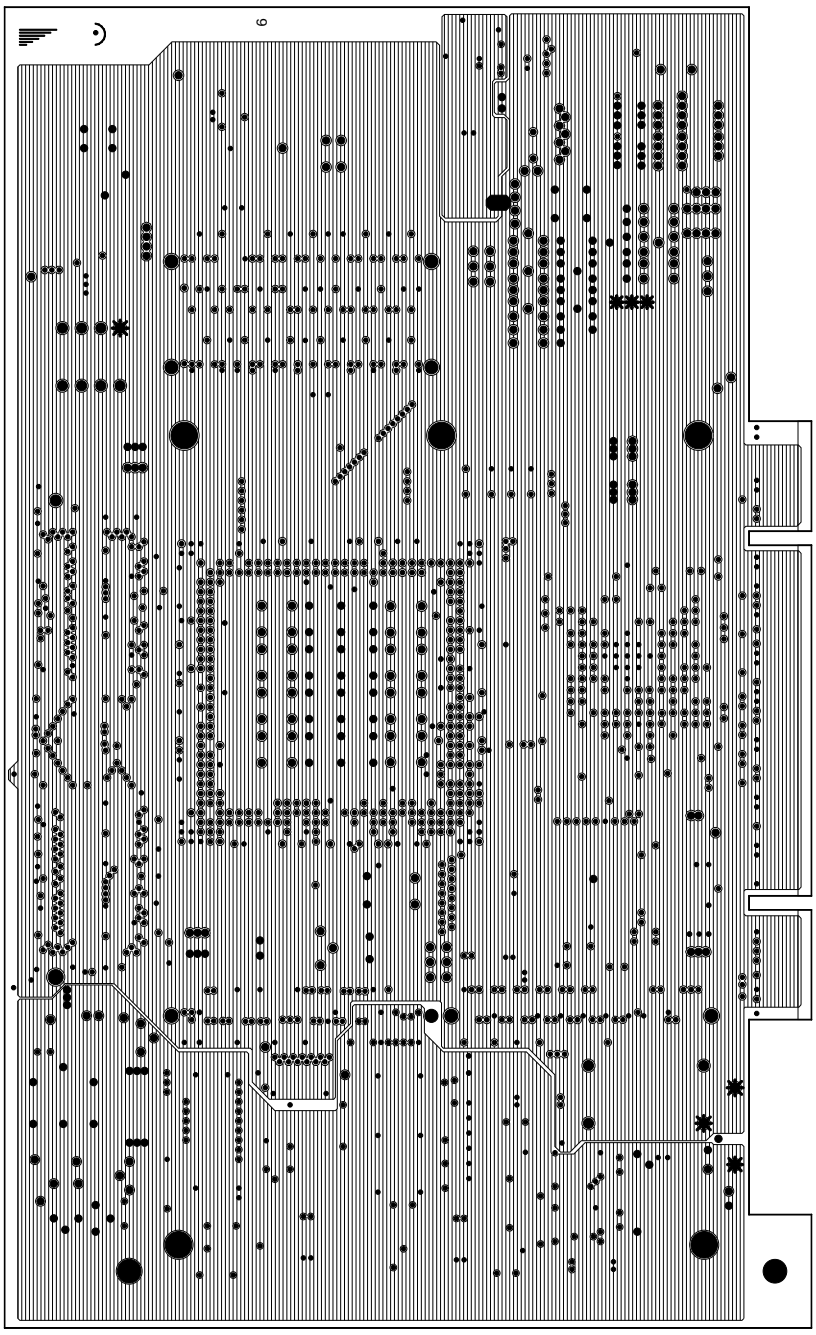


Abbildung E.7: Lage 6 / Ground

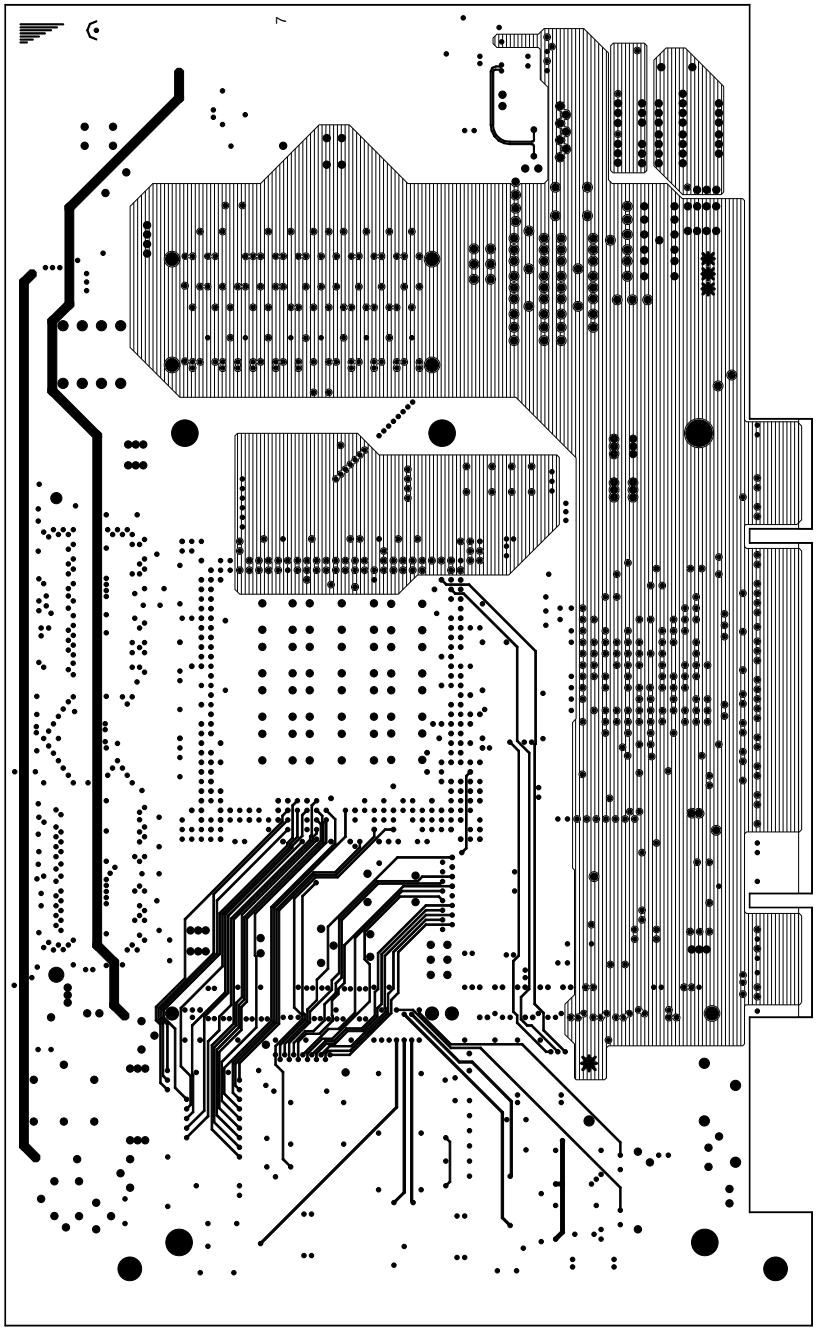


Abbildung E.8: Lage 7

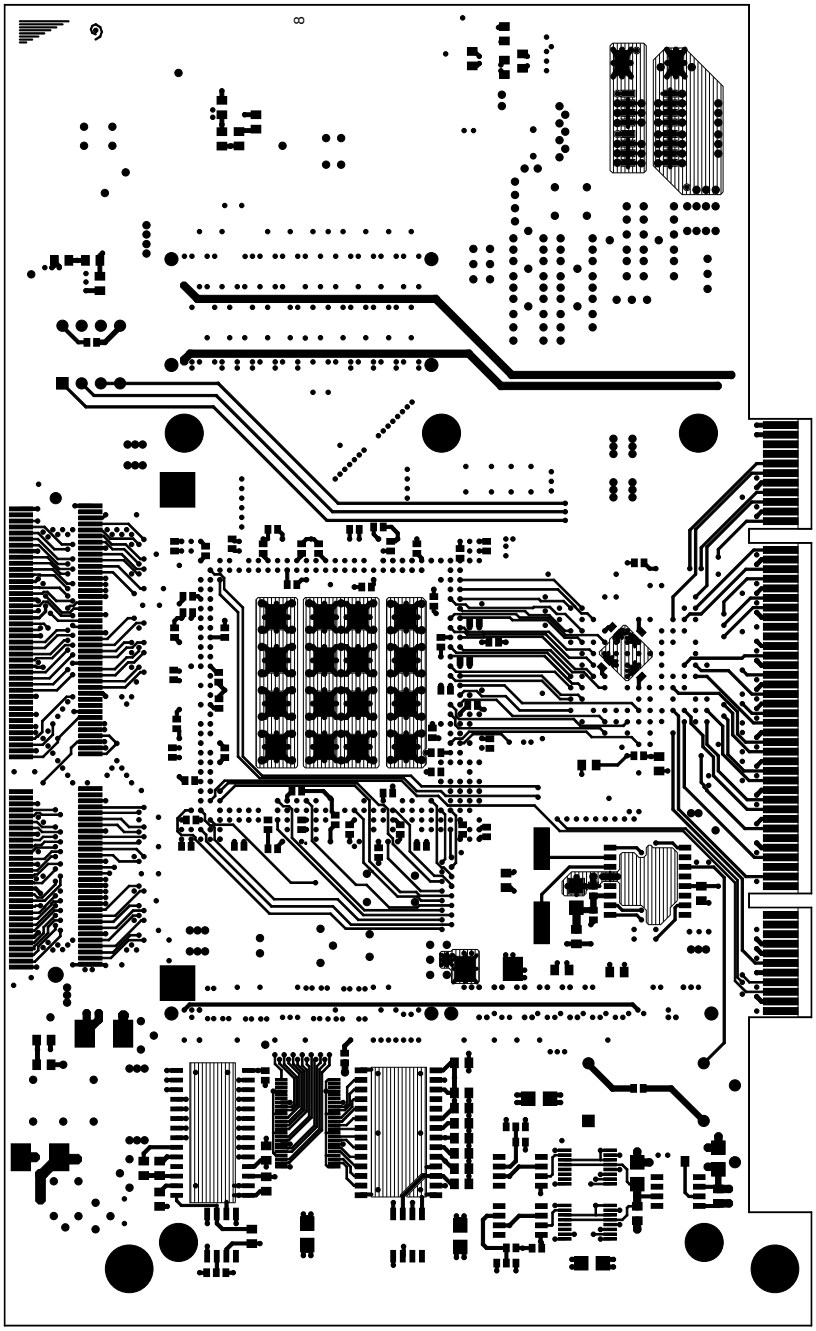


Abbildung E.9: Unterseite

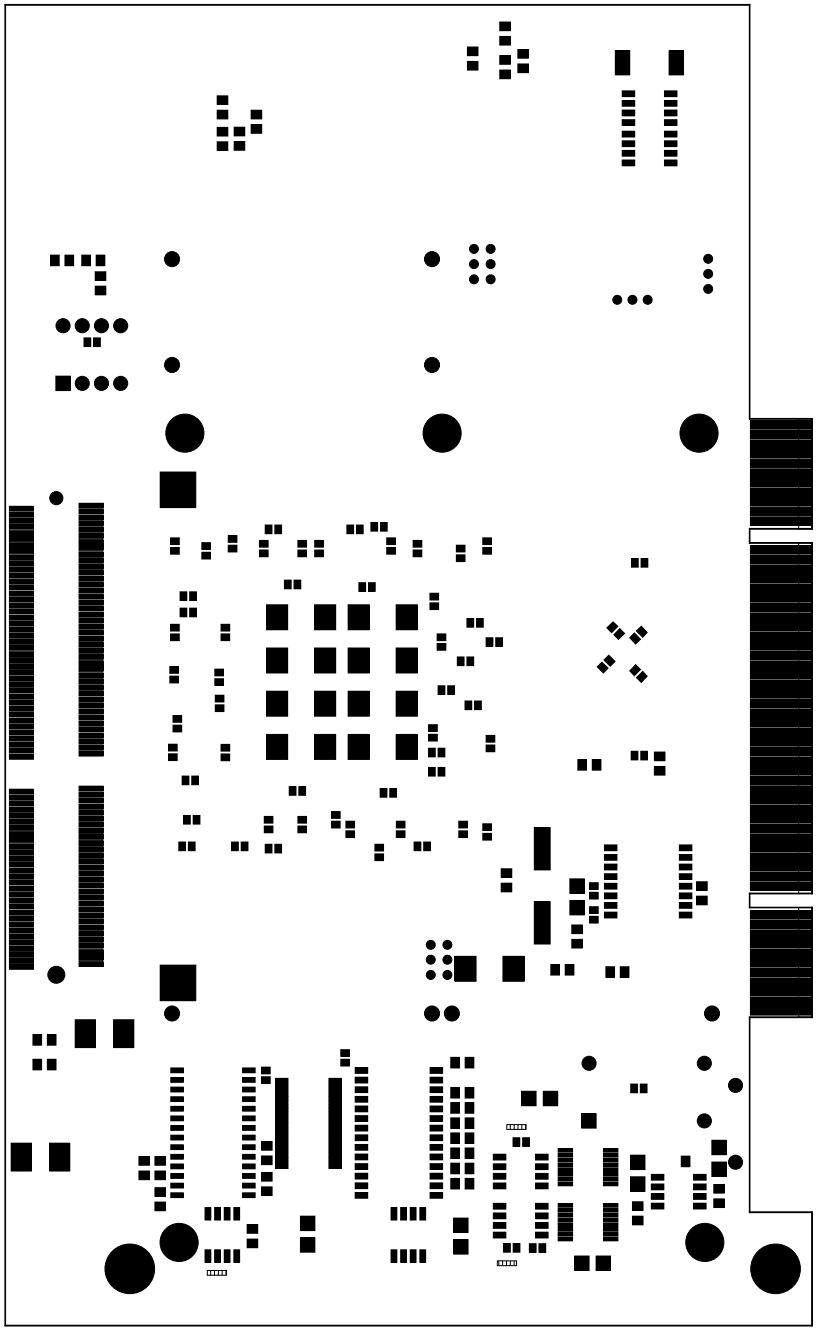


Abbildung E.10: Lötstopplack Unterseite

Anhang F

Megavolt

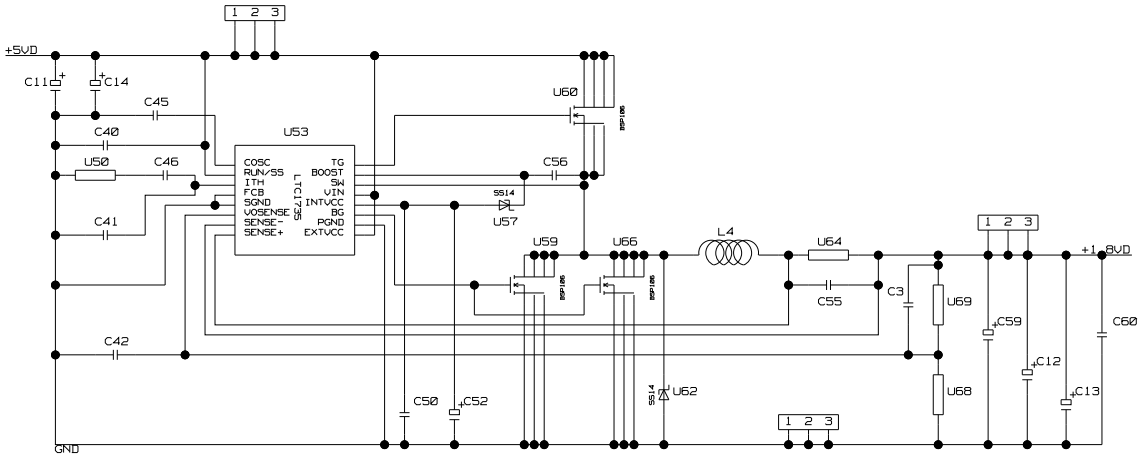
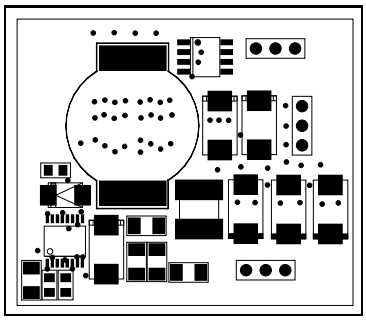
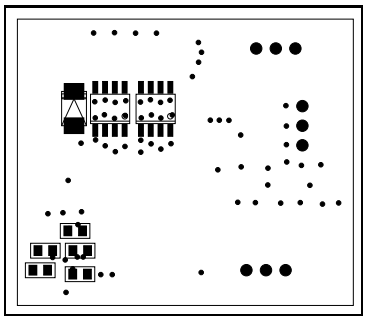


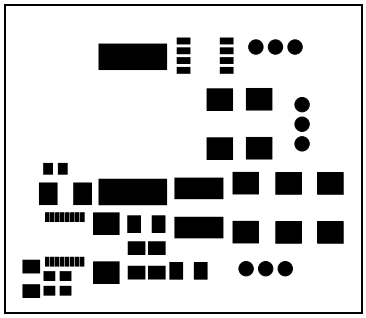
Abbildung F.1: Schaltplan Megavolt



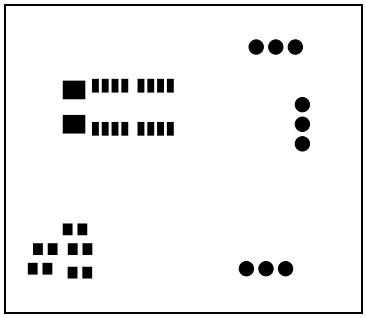
Bestückung Oberseite



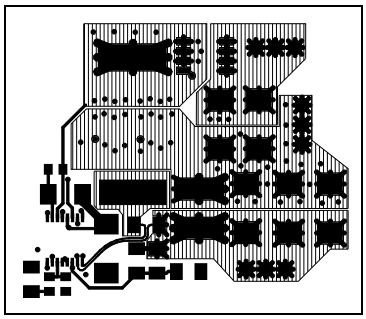
Bestückung Unterseite



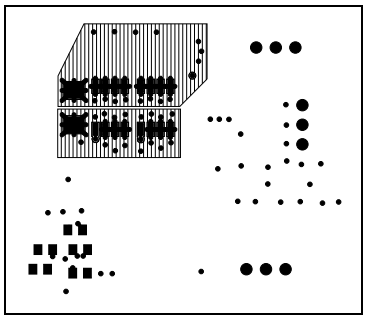
Lötstopplack Oberseite



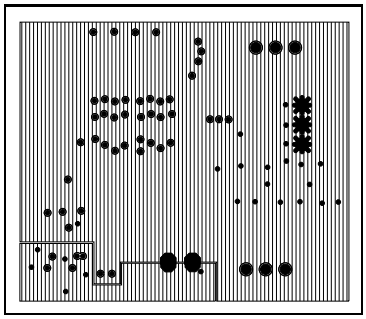
Lötstopplack Unterseite



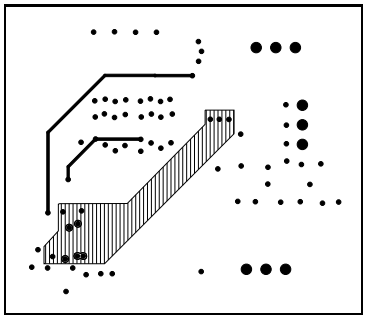
Leiterbahnen Oberseite



Leiterbahnen Unterseite



Lage 2



Lage 3

Abbildung F.2: Lagenpläne Megavolt

Anhang G

Tochterplatine Gosalyn

2	_RESET	A15
17	CMC4_IO<13>	A12
12	CMC4_IO<1>	B13
13	CMC4_IO<10>	A13
30	CMC4_IO<11>	A8
28	CMC4_IO<12>	B9
19	CMC4_IO<14>	C12
21	CMC4_IO<15>	A11
23	CMC4_IO<16>	C11
36	CMC4_IO<17>	B7
34	CMC4_IO<18>	C8
35	CMC4_IO<19>	B8
14	CMC4_IO<2>	C13
37	CMC4_IO<20>	A7
40	CMC4_IO<21>	B6
38	CMC4_IO<22>	C7
41	CMC4_IO<23>	A6
43	CMC4_IO<24>	C6
46	CMC4_IO<25>	C5
44	CMC4_IO<26>	A5
47	CMC4_IO<27>	B5
49	CMC4_IO<28>	A4
48	CMC4_IO<29>	B4
18	CMC4_IO<3>	B12
50	CMC4_IO<30>	C4
27	CMC4_IO<31>	B10
29	CMC4_IO<32>	A9
31	CMC4_IO<33>	C9
8	CMC4_IO<34>	B14
20	CMC4_IO<4>	B11
9	CMC4_IO<5>	A14

7	CMC4_IO<6>	C15
26	CMC4_IO<7>	C10
24	CMC4_IO<8>	A10
11	CMC4_IO<9>	C14
1,5,6,15,25, 22,32,42,52, 33,39,45,16	DGND	A16, C16
53,54,63	AGND	A1, C1
3,4	+3.3VD	B15
55,56	+5VA	B16
64	DAC12_OUTA	B3
62	DAC12_OUTB	A2
60	DAC12_OUTC	A3
58	DAC12_OUTD	C3
57	DAC16_OUT	C2
59	BUCHSE_IN2	B2
61	BUCHSE_IN1	B1

Tabelle G.1: Gosalyn: Abbildung von CMC 4 auf Buchse 3x16

Abbildungsverzeichnis

Abkürzungsverzeichnis

ADC	analog / digital converter
ANNA	analog neural network architecture
ASIC	application specific integrated circuits
BGA	ball grid array
BIOS	basic input output system
CLB	configurable logic block
CMC	common mezzanine card
CMOS	complementary metal oxide semiconductor
CPLD	complex programmable logic device
CPU	central processing unit
DAC	digital / analog converter
DMA	direct memory access
DIVICHI	digital vision chip
EDDA	edge detection array
EEPROM	electrically erasable programmable read only memory
ESR	equivalent/effective series resistance
FPGA	field programmable gate array
FPTA	field programmable transistor array
HDL	hardware description language
LVDS	low voltage differential signal
LVTTL	low voltage transistor-transistor logic
MOSFET	Metalloxydschicht Feldeffekt-Transistor
PC	personal computer
PCI	peripheral component interconnect
PCMCIA	people can't memorise computer industry's abbreviations
PLL	phase locked loop
S-RAM	static random access memory
SD-RAM	synchronous dynamic random access memory
TTL	transistor-transistor logic
VHDL	very high speed integrated circuit hardware description language
high / *low*	logische Namen für aktive / passive Signalzustände

Literaturverzeichnis

[AD768] *16-Bit, 30 MSPS D/A Converter.* Analog Devices, INC., 1996.

[ADC] *12-Bit, 40 MHz Sampling Analog-to-Digital Converter.* Burr-Brown Corporation, 1996.

[Bau01] Christoph Baumann: *Betrieb eines ASICs zur Kantenerkennung in einem Embedded PC unter Linux.* Universität Heidelberg, 2001. Diplomarbeit HD-KIP-01-09.

[Bli00] Holger Blinzinger: *Aufbau eines kompakten Bildverarbeitungsrechners für ein taktiles Blindenhilfssystem.* Universität Heidelberg, 2000. Diplomarbeit HD-KIP-00-02.

[DIVICHI] Markus Loose, Karlheinz Meier, Johannes Schemmel: *Self-calibrating logarithmic CMOS image sensor with single chip camera functionality.* IEEE Workshop on CCDs and Advanced Image Sensors, Karuizawa, R27, 1999. HD-KIP-99-15.

[EDDA] Johannes Schemmel, Markus Loose, Karlheinz Meier: *A 66 x 66 pixels analog edge detection array with digital readout.* Edition Frontinières, ISBN 2-86332-246-X, p. 298, 1999. Poceedings of the 25th European Solid-State Circuits Conference / HD-KIP-99-16.

[EEPROM] *2K 2.5 V Microwire Serial EEPROM.* Microchip Technology, Inc., 1998. 93LC56A/B.

[EVOOPT] Johannes Schemmel, Karlheinz Meier und Felix Schürmann: *A VLSI Implementation of an Analog Neutal Network suited for Genetic Algorithms.* Eingereicht zu „fourth international Conference on Evolvable Systems“, 2001. ICES2001.

[FCT164245] *Fast CMOS 16-Bit Bidirectionl 3.3V to 5V Translator.* Integrated Device Technology, Inc., 1999.

[FPTA] Jörg Langeheine, Joachim Becker, Simon Fölling, Karlheinz Meier und Johannes Schemmel: *Initial Studies of a new VLSI Field Programmable Transistor Array.* Eingereicht zu „fourth international Conference on Evolvable Systems“, 2001. ICES2001.

[Hin96] HERMANN HINSCH: *Elektronik: ein Werkzeug für Naturwissenschaftler*. Springer-Verlag, 1996.

[ICS9161A] *Dual Programmable Graphics Frequency Generator*. Integrated Circuit Systems, Inc., 2000.

[IEEE:CMC] DAVID MOORE: *Draft Standard for a Common Mezzanine Card Family: CMC*. IEEE Standarts Department, 1995.

[Jungo] *WINDRIVER for PLX I/O devices - Developer's Guide*. Jungo Ltd., 2001.

[LT1172] *100kHz, 5A, 2.5A and 1.25A High Efficiency Switching Regulators*. Linear Technology Corporation, 1991.

[LT1175] *500mA Low Dropout Micropower Regulator*. Linear Technology Corporation, 1995.

[LT1963] *1.5A, Low Noise, Fast Transient Response LDO Regulators*. Linear Technology Corporation, 2000.

[LTC1735] *High Efficiency Synchronous Step-Down Switching Regulator*. Linear Technology Corporation, 1998.

[LVDS] *LVDS Owner's Manual*. National Semiconductor Corporation, 2000.

[MAX4544] *Low-Voltage, Single-Supply, Dual SPST/SPDT Analog Switches*. Maxim Integrated Products, 2000.

[MAX5104] *Low-Power, Dual, Voltage-Output, 12-Bit DAC with Serial Interface*. Maxim Integrated Products, 1999.

[MAX604] *5V/3.3V or Adjustable, Low-Dropout, Low I_Q, 500mA Linear Regulators*. Maxim Integrated Products, 1994.

[MAX764] *-5V/-12V/-15V or Adjustable, High-Efficiency, Low I_Q DC/DC Inverters*. Maxim Integrated Products, 1994.

[MAX873] *Low-Power, Low-Drift, +2.5 V Precision Voltage Reference*. Maxim Integrated Products, 1992.

[OP] *Fast VIP Op Amps Offer High Speed at Low Power Consumtion*. National Semiconductor Corporation, 1995. LM6361.

[PCI] *PCI Local Bus Specification*. PCI Special Interest Group, 1998.

[PLX9054] *PCI 9054 Data Book*. PLX Technology, Inc., 2000.

[Polar] *Si6000b Controlled Impendance Field Solver: User Guide*. Polar Instruments Ltd, 2001.

[SDModul] *Small Outline SDRAM Module*. Micron Semiconductor Products, INC., 2000. 16Meg*64 SDRAM SODIMM.

[SDRAM] *Synchronous DRAM.* Micron Semiconductor Products, INC., 1999. 16Mb:x16, IT SDRAM.

[Si4886] *N-Channel Reduced Q_g, Fast Switching MOSFET.* Vishay Siliconix, 2000.

[SRAM] *256K*36 & 512K*18-Bit Flow Through NtRAM.* Samsung Electronics LTD, 1999.

[VHDL] GUNTHER LEHMANN, BERNHARD WUNDER, MANFRED SELZ: *Schaltungsdesign mit VHDL.* Franzis Verlag, 1994.

[Xil:Data] *Virtex^{TM}-E 1.8V FPGAs.* Xilinx, Inc., 2001. Preliminary Product Specification.

[Xil:DLL] *Synthesizable High Performance SD-RAM Controller.* Xilinx, Inc., 2000. XAPP234.

[Xil:Est] *Powerestimator for Virtex-E FPGAs.* Xilinx, Inc., 2000. XAPP158.

[Xil:Power] AUSTIN LESEA und MARK ALEXANDER: *Powering Xilinx FPGAs.* Xilinx, Inc., 2001. XAPP158.

Danksagungen

Mein herzlicher Dank gilt den folgenden Personen:

Herrn Prof. Dr. Karlheinz Meier für die freundliche Betreuung dieser Arbeit,
Herrn Prof. Dr. Reinhard Männer für die Zweitkorrektur,
Dr. Johannes Schemmel für immer neue Herausforderungen,
Jörg Langeheine für die Möglichkeit zur Mitarbeit am FPTA Projekt,
Thorsten Maucher für herzliche Gespräche aller Art,
Andreas Breidenassel für Diskussionen über VHDL,
Felix Schürmann für das *Hoverboard* und LVDS-Messungen,
den Mitarbeitern der Elektronikwerkstatt für Diskussionen über jegliche Elektronik,
Ralf Achenbach für die Hilfen im Testlabor,
Dr. John Lupton, Arne Wiebalck und David Emschermann für ihre Unterstützung,
Anne Weber für so vieles.

Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den ....