Entwicklung einer USB-Schnittstelle für die Ausleseelektronik eines Cadmium-Zink-Tellurid Detektors

Diplomarbeit

von

Giuseppe Distratis

Eberhard Karls Universität Tübingen
Fakultät für Mathematik und Physik
Institut für Astronomie und Astrophysik
Abteilung Astronomie

Mai 2006

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

Die Röntgenastronomie ist noch ein relativ junges Kapitel in der Geschichte der Astronomie. Es ist kaum mehr als ein halbes Jahrhundert her, als Thomas Burnight mit einem Detektor auf einer umgebauten V2-Rakete erstmals die Röntgenstrahlen der Sonnenkorona beobachtete.

Da Röntgenstrahlen vollständig von der Erdatmosphäre absorbiert werden, ist eine Beobachtung nur aus sehr großer Höhe bzw. aus dem Weltall möglich. Dies stellte natürlich eine Hürde dar, was vielleicht erklären kann, warum erst vierzehn Jahre nach dem ersten Experiment von Burnight Riccardo Giacconi mit einer Höhenforschungsrakete, die mit einem Röntgendetektor ausgestattet war, zwar nicht die beabsichtigte Röntgenaufnahme des Mondes, jedoch die Entdeckung der ersten kosmischen Röntgenquelle Scorpius X-1 gelang. Danach ging es auf diesem Gebiet zwar nicht sehr schnell, aber stetig voran. Zuerst wurden für die Beobachtung noch Raketen und Ballone benutzt, später Satelliten. Durch die technischen Entwicklungen in diesem Bereich konnten immer mehr kosmische Röntgenquellen entdeckt werden. Mit den empfindlichen Instrumenten an Bord des ROSAT-Satelliten gelang es in den 90er Jahren, mehr als 100 000 Röntgenquellen zu entdecken und nebenbei auch Giacconis ursprüngliches Vorhaben zu verwirklichen – eine Röntgenaufnahme des Mondes.

Das Institut für Astronomie und Astrophysik der Universität Tübingen (IAAT) engagiert sich bereits seit über 40 Jahren auf diesem Gebiet. Schon in den 1960ern wurden Kameras zur Beobachtung der Röntgenstrahlung der Sonne entwickelt und Anfang der 70ern unter der Leitung von Herrn Professor Joachim Trümper, der die Röntgenastronomie in Tübingen vorantrieb, wurden eigene Ballonexperimente durchgeführt. Herr Professor Rüdiger Staubert führte die Forschung in diesem Bereich fort mit der Beteiligung an vielen Satellitenprojekten u.a. mit dem bereits erwähnten, erfolgreichen ROSAT-Projekt sowie XMM-Newton und INTEGRAL, die noch aktiv sind. Als Nachfolger betreibt heute Herr Professor Andrea Santangelo die Weiterführung dieser Forschung u.a. mit der Beteiligung an den aktuellen Projekten SIMBOL-X, eROSITA und MIRAX.

1.1 Motivation und Inhalt dieser Arbeit

Gegenstand dieser Arbeit war die Entwicklung einer Schnittstelle für das Auslesen der Daten eines Röntgenteleskops, das sich an Bord des MIRAX-Satelliten befinden wird. Aus diesem Grund werden der Satellit, das Teleskop und der dafür eingesetze Detektor im folgenden Abschnitt kurz vorgestellt. Dieses Teleskop (siehe Abschnitt 1.2.1) wird derzeit am *Center for Astrophysics & Space Sciences* der Universität von Kalifornien in San Diego, USA (CASS/UCSD) entwickelt.

Es soll hier jedoch nicht der Eindruck erweckt werden, es würde sich bei dieser Schnittstelle um einen Teil der Bordelektronik handeln. Es ist vielmehr der erste Schritt in diese Richtung, mit der Hoffnung, das IAAT bei der Aufgabe, die es für das MIRAX-Projekt übernommen hat, ein kleines Stück voranzubringen.

Kapitel 2 befasst sich eingehend mit der Elektronik für das Auslesen des Detektors. Sie stellt den Ausgangspunkt dar für die Entwicklung, die in den Kapiteln 4 und 5 ausführlich beschrieben wird. Die Anhänge A und B sind kurze Einführungen in die für diese Entwicklung eingesetzten Technologien.

Abkürzungen und Akronyme, die im Laufe dieser Arbeit benutzt werden, sind bei ihrer ersten Verwendung auch im Klartext aufgeführt, kommen sie mehr als einmal im Text vor, ist es möglich, sie im Anhang C nachzuschlagen. [1]

1.2 MIRAX

Monitor et Imageador de RAios-X (MIRAX) ist der Name eines Kleinsatelliten der gemeinsam vom Brasilianischen *Instituto Nacional de Pesquisas Espaciais* (INPE), der *Space Research Organisation Netherlands* (SRON), dem *Massachussetts Institute of Technology* (MIT), der *University of Warwick* in Großbritannien sowie dem CASS/UCSD und dem IAAT entwickelt wird. Das IAAT steuert mit der *Central Electronics Unit* (CEU) die zentrale elektronische Steuereinheit bei und ist an der Entwicklung der Auswertungssoftware beteiligt.

Wissenschaftliches Ziel dieser Mission ist die Langzeitbeobachtung der galaktischen Ebene und insbesondere deren Zentralregion, im Energiebereich von $2 - 200\,\mathrm{keV}$ mit sehr hoher Zeitauflösung von $10\,\mu\mathrm{s}$ [SR+03].

Wegen der Nähe der Sonne zur galaktischen Zentralregion im Laufe eines Erdorbits, ist ihre Beobachtung nicht ununterbrochen möglich. Für drei Monate im Jahr sollen deshalb andere interessante Regionen wie Crab oder Vela beobachtet werden.

[1] Oft sind Begriffe der Elektronik besser unter deren Akronym bekannt als in Klarschrift. Um den Wiedererkennungswert dieser Kürzeln auszunutzen, werden sie auch dort eingesetzt, wo sie vielleicht nicht nötig gewesen wären.

Abbildung 1.1: Offizielles Logo der *MIRAX*-Mission

An Bord von MIRAX werden sich drei Coded-Mask-Röntgenteleskope befinden. Bei einem davon, dem *Soft X-ray Imager* (SXI), handelt es sich um das Ersatzexemplar der *Wide Field Camera* des sehr erfolgreichen italienisch/niederländischen Satelliten BeppoSAX. Es ist mit einem Vieldrahtproportionalzähler, empfindlich im Energiebereich von $2 - 28\,\mathrm{keV}$ ausgerüstet und besitzt ein Gesichtsfeld von $20° \times 20°$. Die zwei identischen *Hard X-ray Imager* Teleskope (HXI) überdecken zusammen einen Himmelsausschnitt von $45,8° \times 22,9°$ und sind im Energiebereich von $10 - 200\,\mathrm{keV}$ empfindlich.

1.2.1 Hard X-ray Imager

Der HXI ist der Beitrag vom CASS/UCSD an dem MIRAX-Projekt [R+03]. Es handelt sich um ein *Coded-Mask*-Teleskop mit einer Wolframmaske von $0,6\,\mathrm{mm}$ Dicke, gebildet aus 157×157 Zellen in *Modified Uniformly Redundant Array* (MURA) Anordnung. Die Abschattung der Maske beträgt 50%. Im Abstand von $76\,\mathrm{cm}$ hinter der Maske befindet sich eine Matrix aus 3×3 *Crossed-Strip*-Detektormodulen mit einer Gesamtfläche von ca. $360\,\mathrm{cm}^2$ (s. Abb. 1.2). Die Detektormodule bestehen ihrerseits aus einer Matrix aus 2×2 *CdZnTe*-Detektoren von $32 \times 32 \times 2\mathrm{mm}^2$. Die je 64 Anoden und Kathoden der 4 Einzeldetektoren sind so miteinander verbunden, dass das Modul als ein 128×128-Pixel großer Detektor angesteuert werden kann.

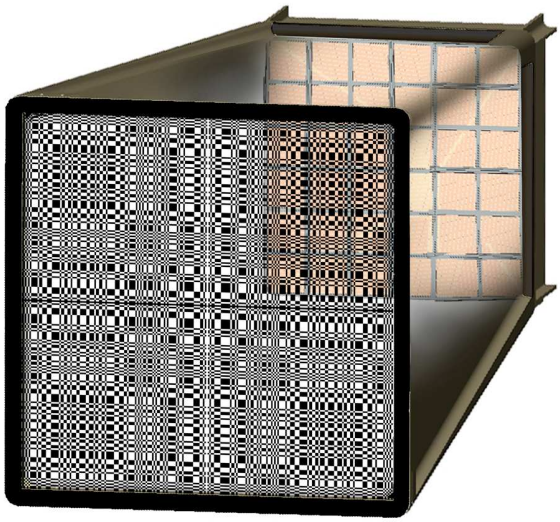

Abbildung 1.2: Aufbau des Hard X-ray Imagers [SR+03]

1.2.2 CdZnTe-Detektor

Cadmium-Zink-Tellurid-Detektoren zeichnen sich durch gute Effizienz und Energieauflösung sowie sehr niedriges thermisches Rauschen bei Raumtemperatur aus. Die ausgezeichnete Ansprechzeit dieser Art von Detektoren erlauben ihren Einsatz in Anwendungen wie MIRAX.

Die bei MIRAX einzusetzenden Detektoren weisen zwischen den Anodenstreifen zusätzliche s.g. *Steering Electrodes* auf. Sie werden mit einer Vorspannung (*Bias*) angesteuert, die bei etwa 90% der Biasspannung zwischen Anoden und Kathoden $(130 - 200\,\mathrm{V})$ liegt. Die Steering Electrodes tragen zur Verbesserung der Ladungssammeleigenschaft des Detektors bei [R+03; KM00; S+00].

1.2.3 Labor-Prototyp

Der dem IAAT zur Verfügung stehende *CdZnTe*-Detektor (siehe Abbildung 1.5) ist ein sehr früher Prototyp, welcher auf einem *Splayed-Out-Board* montiert ist. Dieses Board erlaubt durch die weit aufgefächerten Anschlüsse zwar einen schnellen Zugriff auf die einzelnen Elektroden, hat aber den Nachteil einer sehr großen Empfindlichkeit für elektromagnetische und sogar akustische Störungen. Zur Zeit wird am IAAT ein sehr kompakter Träger für den Detektor konstruiert, um diese Probleme wesentlich zu verringern.

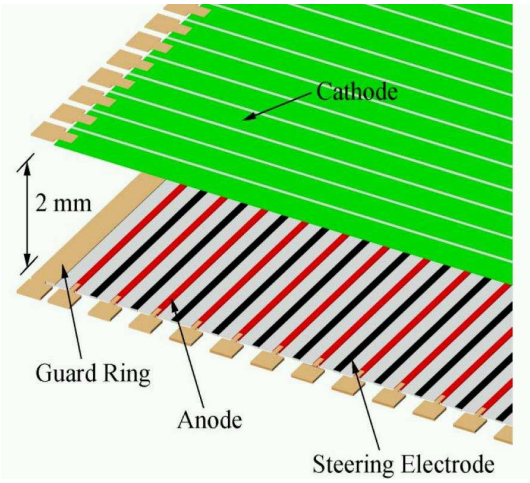

Abbildung 1.3: Elektrodenanordnung des *CdZnTe*-Detektors[S+00]

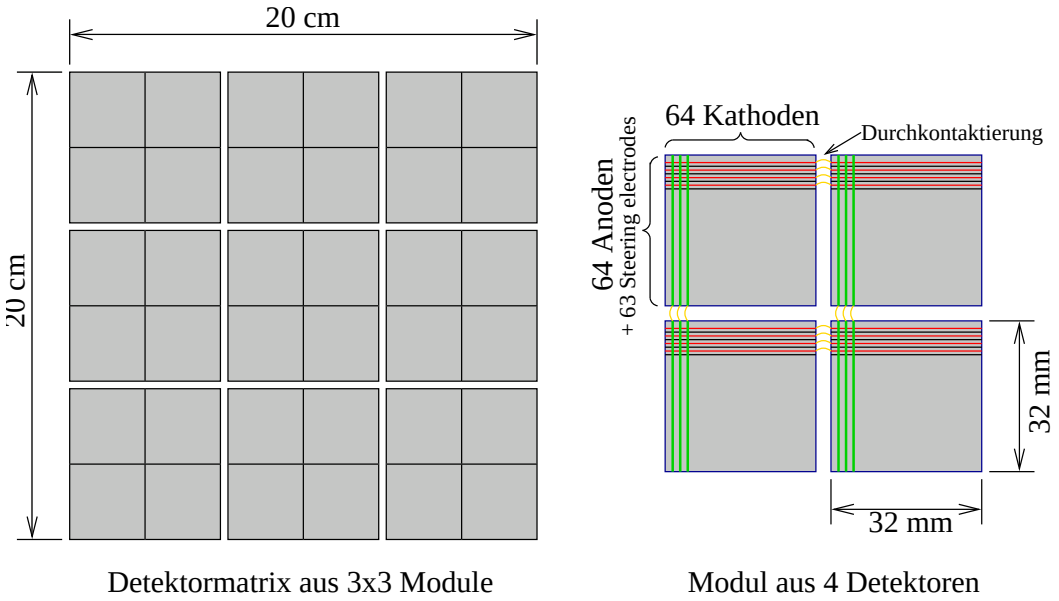

Abbildung 1.4: Aufbau des HXI-Detektor
Der Detektor ist aus 9 einzelnen Modulen aufgebaut. Jedes Modul besteht aus 4 *CdZnTe*-Crossed-Strip-Detektoren mit 64 Kathoden und 64 Anoden, die zwischen benachbarten Detektoren durchkontaktiert werden. Es entsteht auf dieser Weise einen Gitter aus 128×128 Elektroden, deren Kreuzungspunkte eine ebenso große Matrix von Bildpunkten (*Pixels*) bildet. Die *Steering electrodes* werden auf die gleiche Weise durchkontaktiert. Dies wird im Bild aus Gründen der Übersichtlichkeit nicht gezeigt.

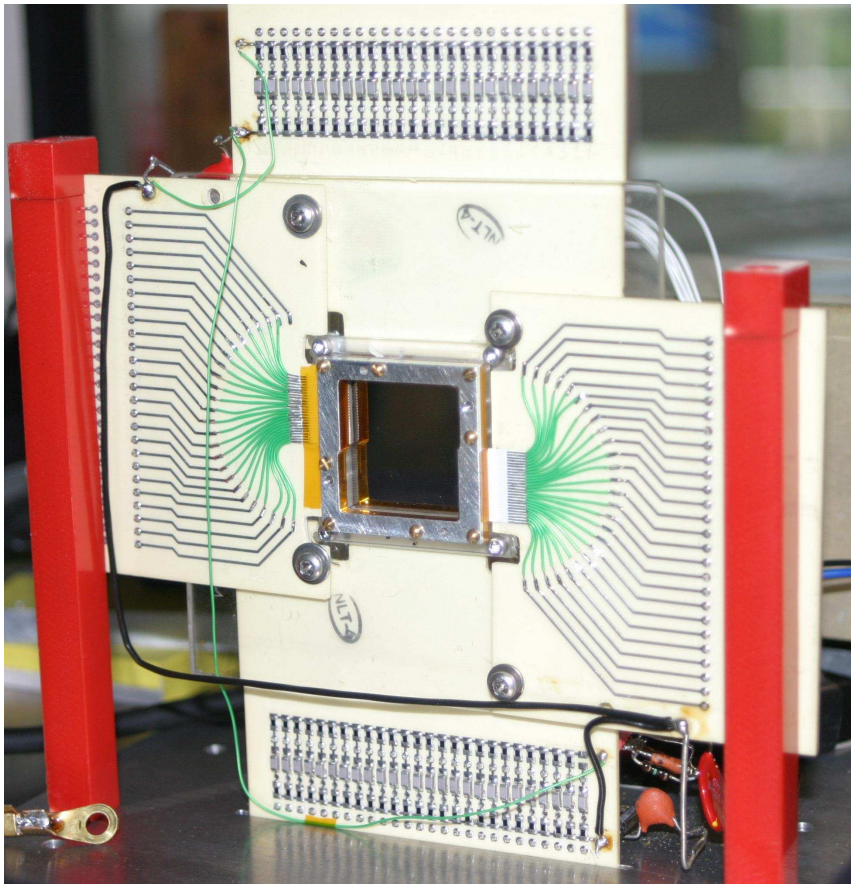

Abbildung 1.5: Prototyp des *CdZnTe*-Detektors am IAAT
Das Bild zeigt die Kathodenseite. Die Anschlüsse der Elektroden sind zu je einer Hälfte rechts, bzw. links des Detektors durch kleine Drähte mit den Leiterbahnen des *Splayed-Out-Boards* verbunden. Jede Kathode ist über einen Widerstand von 1 GΩ mit der Masse verbunden. Das Signal wird direkt an dem Widerstand abgegriffen. Auf der Rückseite befinden sich die Anoden. Sie sind senkrecht zu den Kathoden angeordnet und ebenfalls über 1 GΩ Widerstände auf der Bias-Spannung gehalten. Um die Ausleseelektronik nicht mit dieser hohe Spannung zu verbinden, werden die Signale durch Entkopplungskondensatoren geleitet.

Kapitel 2

Ausleseelektronik

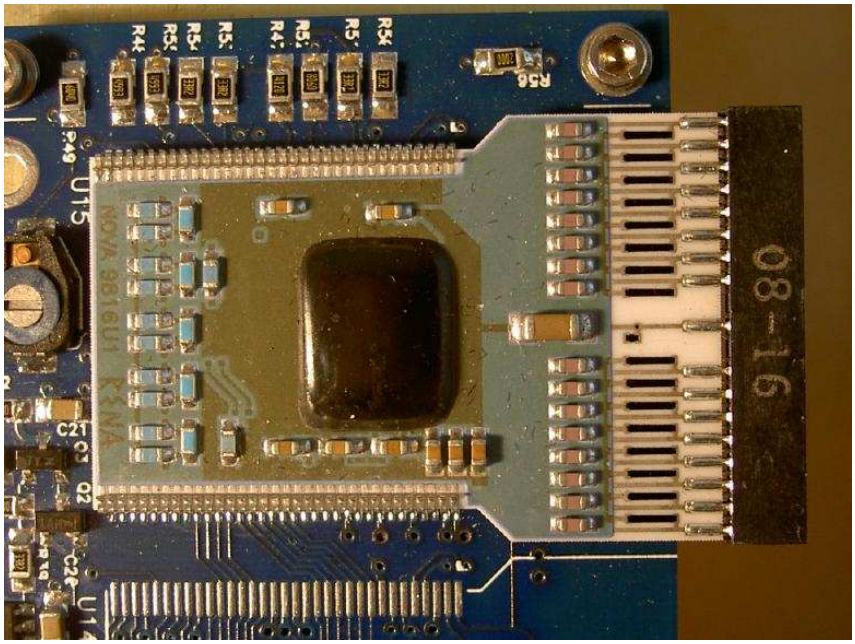

Abbildung 2.1: *RENA*-Chip auf der Hybrid-Platine

Das *Readout Electronics for Nuclear Applications* (*RENA*) Auslesesystem der Firma *NOVA R&D Inc.* in Riverside, Kalifornien ist speziell für den Betrieb mit hochkapazitiven Halbleiterdetektoren im Bereich von 6 bis 50 pF konzipiert. Es zeichnet sich durch eine hohe Flexibilität und Empfindlichkeit aus und erlaubt den Einsatz von Detektoren aus Silizium, Cadmium-Tellurid, Cadmium-Zink-Tellurid, Quecksilber-Jodid und Gallium-Arsenid. Die hohe Anzahl unabhängiger und einzeln triggerfähiger Eingangskanäle macht es für die Auslese von Crossed-Strip-Detektoren wie dem CdZnTe-Detektor der HXIs besonders geeignet [MTK+98].

2.1 Allgemeines Design

Die Hauptkomponente der *RENA* ist der s.g. *RENA*-Chip, eine *kundenspezifische integrierte Schaltung* (ASIC[1]) die auf einer Hybrid-Platine[2] montiert ist.

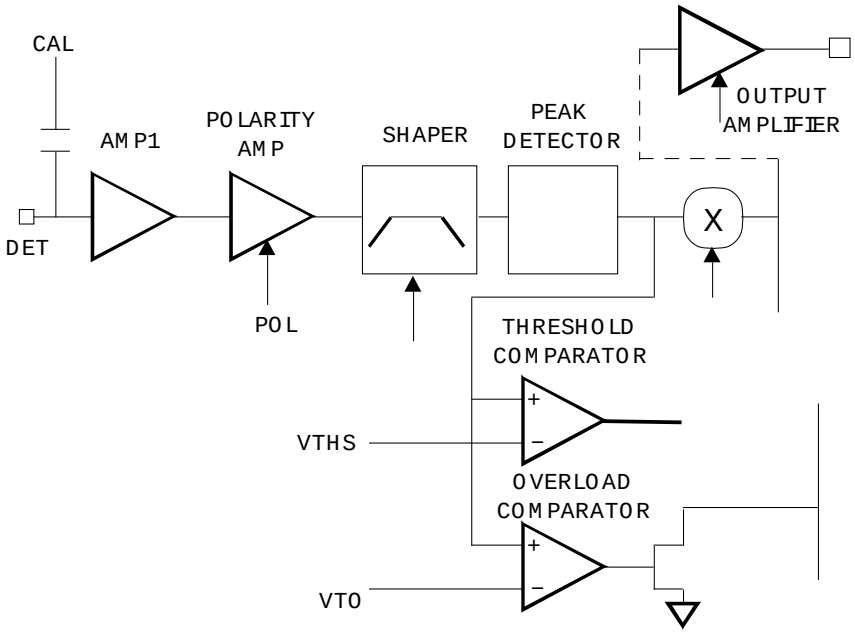

Abbildung 2.2: Analogpfad eines Eingangskanals [MTK+98]

In jedem seiner 32 Eingangskanäle erzeugt ein integrierender Ladungsverstärker einen Spannungspuls, dessen Energie proportional zur Ladung ist, welche von der angeschlossenen Detektorelektrode gesammelt wird (s. Abb. 2.2). Die von einem Photon im Detektor erzeugte Ladung kann von bis zu je drei benachbarten Kathoden und Anoden gesammelt werden. Ist der Kanal an einer Anode angeschlossen, muss die Polarität des Signals durch einen Polaritätsverstärker invertiert werden, bevor es durch den digitalgesteuerten Pulsfilter und Peak-Detektor weiter verarbeitet und über einen Multiplexer (der mit „X“ gekennzeichnete Kreis in Abb. 2.2) zum Ausgangsverstärker geführt wird. Zwei Komparatoren vergleichen die Spannung des Signals mit den digital eingestellten Trigger- bzw. Überladungspegeln.

Das Triggersignal am Ausgang des Schwellenkomparators steuert zusätzlich die digitale Adress-Logik.

[1] Application Specific Integrated Circuit.

[2] Engl., vom Lat. *hybris*: Zwitterwesen. Als *Hybrids* werden in der Elektronik Tochterplatinen bezeichnet, die wie Bauteile auf einer Mutterplatine verwendet werden (s. Abb. 2.1).

Die Konfigurationsdaten der digitalgesteuerten analogen Komponenten werden in zwei Schieberegister des *RENA*-Chip geschrieben (s. Abb. 2.3). Die in einem weiteren Register gespeicherte Information bestimmt welches Analogsignal zu Überwachungszwecken an einen Ausgangsverstärker geleitet wird. Die Ein- und Ausgangsleitungen der Schieberegister werden über alle Hybrids eines Moduls kaskadiert, so dass von aussen nur je ein Register sichtbar ist, dessen Breite von der Anzahl der auf dem Board installierten Hybrids abhängt.

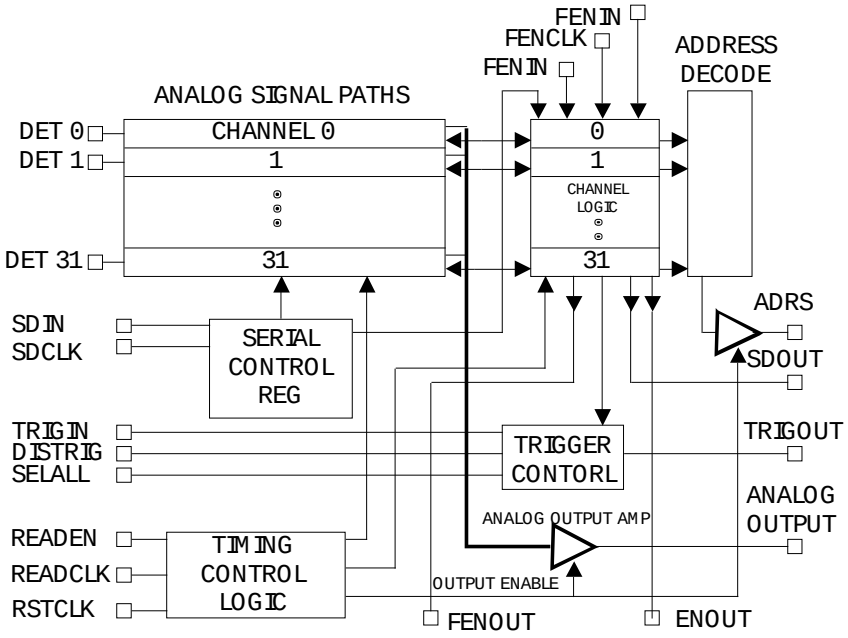

Abbildung 2.3: Blockschaltbild des *RENA*-Chips
Das *Serial Control Register* enthält die Konfigurationsdaten der digitalgesteuerten analogen Komponenten. Das *Force-Enable (FEN)* Register ist in die Adress-Logik integriert und bestimmt welcher Kanal dauerhaft am Analogausgang zur Überwachung bereitsteht [MTK+98].

Der *RENA*-Chip ist so konstruiert, dass bis zu 4 Hybrids auf einem *RENA Module Board* installiert werden können (s. Abb. 2.4) und sich sowohl einen digitalen Adress-Bus als auch einen analogen Ausgangsbus teilen, während die Steuerleitungen in einer *Daisy-Chain*[3] kaskadiert werden. Dadurch ist nur genau ein Befehl notwendig um die Register aller Hybrids zu konfigurieren.

Ein Digital-Analog-Wandler (DAC[4]) mit vier Kanälen vom Typ *MAXIM MAX525* erzeugt die Referenzspannungen VTHS und VTO. Ein 14-Bit Analog-Digital-Wandler (ADC[5]) vom Typ *Analog Devices AD9243* sorgt für die Digitalisierung der ge-

[3] Eine Kette von gleichen elektronischen Komponenten, bei der die Ausgänge des ersten Glieds direkt mit den Eingängen des zweiten Glieds verbunden sind usw.
[4] Digital to Analog Converter
[5] Analog to Digital Converter

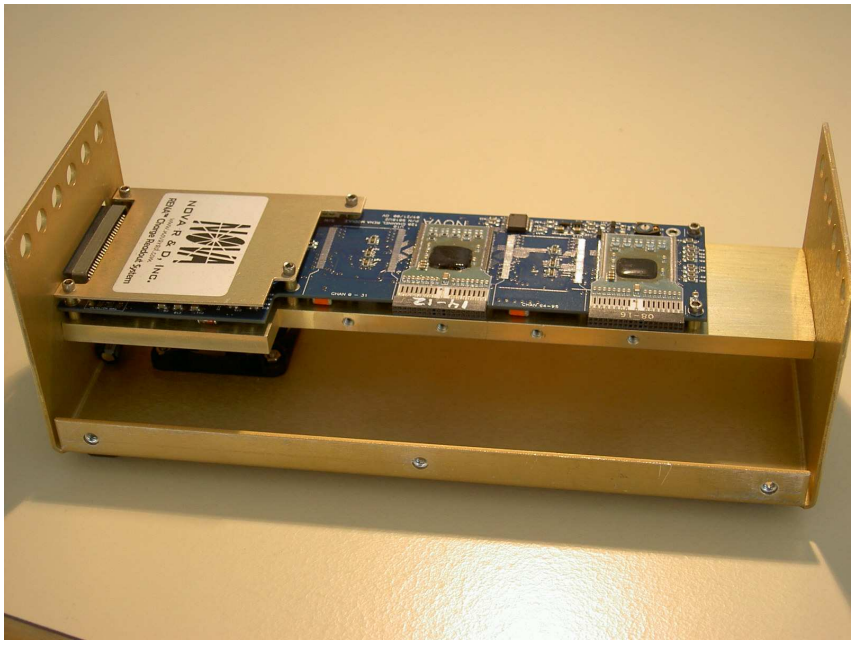

Abbildung 2.4: Das *RENA Module Board* des IAAT
Von vier möglichen *RENA*-Hybrids sind nur zwei installiert, auf den Positionen 2 und 0 (von links nach rechts). Da bei dieser Version des Boards die Einstellung der Polaritätsverstärker nicht für jeden einzelnen Kanal getrennt erfolgen kann, sondern nur paarweise für die Chips 0 und 1 bzw. 2 und 3, ist es nur mit dieser Minimalkonfiguration möglich, sowohl die Anoden als auch die Kathoden des Detektors auszulesen.

messenen Ladungsmenge.

Auf dem Board befindet sich noch ein *Field-Programmable Gate Array* (FPGA[6]) vom Typ *Spartan XCS20* der Firma *XILINX*. Das FPGA stellt zwei weitere Schieberegister zur Verfügung: das DAC-Register für die Kontrolle der Referenzspannungen und das *Control and Status Register* (CSR), das alle für den Betrieb des *RENA Module Boards* notwendigen Parameter enthält. Dieses FPGA steuert die Hybrids, die Erzeugung der Kalibrationspulse und die Kommunikation mit der Steuer- und Kommunikationsschnittstelle, dem *RENA System Interface*.

Die *RENA Module Boards* werden über ein 50-poliges Hochdichte-Flachbandkabel an das *RENA System Interface* angeschlossen. Obwohl das Interface bis zu 16 Boards adressieren kann, sind seine Stromversorgungskomponenten so dimensioniert, dass nur 8 Boards mit den nötigen Betriebsspannungen versorgt werden können.

Für die Kommunikation mit dem Datenerfassungscomputer wird das Interface über

[6] Auf FPGAs, ihre Eigenschaften und Arbeitsweise wird im Anhang A näher eingegangen.

ein weiteres 50-poliges Flachbandkabel an einer PCI-Schittstelle[7] vom Typ *National Instruments DIO-96* angeschlossen.

Die einzelnen Boards werden durch ihre Identifikationsnummer (ID), die an einer *Jumper*-Leiste als binär codiert wird, unterschieden. Die Nummerierung muss bei Null beginnen und lückenlos bis zur Anzahl der angeschlossenen Boards minus eins gehen. Der Datenbus wird auf einer Seite durch das *RENA System Interface* terminiert. Die Terminierung der anderen Seite wird durch entsprechende Widerstände auf dem letzten Board sichergestellt.

Das *RENA System Interface* übernimmt die Konfiguration[8] und die Kontrolle der einzelnen Boards, das zyklische Auslesen der Daten und deren Weitergabe an den Auswertecomputer. Auch in dem *RENA System Interface* werden alle Aufgaben von einem FPGA vom Typ *Spartan XCS20* übernommen. Da weder das *RENA Module Board* noch das *RENA System Interface* über einen permanenten Konfigurationspeicher verfügen, muss zuerst das FPGA auf dem Interface vom Datenerfassungsprogramm über die PCI-Schnittstelle konfiguriert werden. Erst danach kann der binäre Datenstrom für die Konfiguration der FPGAs auf den installierten *RENA Module Boards* über das dann betriebsbereite Interface gleichzeitig an alle Boards übertragen werden.

Daten und Befehle werden über einen 4 Bit breiten bidirektionalen, synchronen Differentialbus (LVDS[9]) mit unidirektionalen Takt- und Steuersignalen übertragen. Die zusätzliche Leitung GLOBX wird bei der aktuellen Version des Boards nicht verwendet (Tabelle 2.1, Block *Communication group*).

Die differentielle Leitung IAOUT+/- überträgt das analoge Kontrollsignal aus einem ausgewählten Eingangskanal, das im Interface weiter verstärkt wird und zur Kontrolle und Abstimmung genutzt werden kann. Die Konfiguration des FPGAs geschieht über die vier Leitungen im Block *Initialization group*.

Über die Leitungen des Blocks *Power Group* versorgt das Interface die angeschlossenen *RENA Module Boards* mit den drei benötigten Versorgungsspannungen, Masse sowie den drei optionalen Hochspannungen für die Polarisation des Detektors.

[7] Peripheral Component Interconnect, ein Bus-Standard für den Anschluß von Peripheriegeräten in Computersystemen.

[8] Dieser Begriff wird in zwei verschiedenen Kontexten benutzt. Wird das *RENA Module Board* als eine eigenständige funktionale Einheit betrachtet, bedeutet *Konfiguration* das Einstellen der Betriebsparameter. Im Englischen könnte stattdessen das Wort *Setup* benutzt werden. Wird dagegen über das FPGA auf den Boards gesprochen, heißt *Konfiguration* das Übertragen des Bitstroms, welcher die Funktionalität des FPGAs bestimmt. Um Missverständisse zu vermeiden, wird in diesem Fall das FPGA ausdrücklich genannt.

[9] *Low Voltage Differential Signaling*. Signalübertragungstechnik bei der das Signal als Spannungsdifferenz zwischen zwei parallelen Leitungen erfolgt. Dies verbessert wesentlich das Verhältnis zwischen Signal und Rauschen, da ein Störsignal die absolute Spannung beider Leitungen in gleichem Maße beeinflußt, nicht aber deren Differenz. Die Spannung kann auf diese Weise niedrig gehalten werden, was sich positiv auf die Schaltgeschwindigkeit auswirkt.

Tabelle 2.1: Belegung des Buskabels [NOV01]

Signal name	Type und direction	Function
(Communications group)		
CLK	LVDS to mod. boards	24.0000 MHz clock for all actions
BUSDAT[3:0]	LVDS bidirectional	data or command response from, and command to, the RENA boards
CMDVALID	LVDS to mod. boards	command valid on BUSDAT, going to the RENA boards
GLOBX	LVDS to mod. boards	global "X" line to the RENA boards - reserved (it could be used for pulser or SELALL trigger)
(Analog test output group)		
IAOUT+/-	differential analog	analog test output from RENA board
(Initialization group)		
$\overline{\text{RESET}}$	5V CMOS w/ pullups	reset all logic, but leave FPGA configured
$\overline{\text{PROGRAM}}$	5V CMOS w/ pullups	for FPGA configuration, see XCS20 data sheet
$\overline{\text{INIT}}$	5V CMOS w/ pullups	for FPGA configuration, see XCS20 data sheet
DONE	5V CMOS	for FPGA configuration, see XCS20 data sheet
(Power group)		
GND		power and signal reference
VSUPPLYA	+6 to +10 V power	source for VAA and VDD supply
VSUPPLYB	+10 to +15 V power	power supply for voltage regulator circuit
VSUPPLYC	+3.5 to +10 V power	source for VRI supply
VBIAS1	detector bias	configure by jumper as needed
VBIAS2	detector bias	configure by jumper as needed
VBIAS3	detector bias	configure by jumper as needed

Ein von NOVA erstelltes Windows-Programm, namens *RENA Data Acquisition (DAQ)*, wird zur Konfiguration des Systems und zur Datenerfassung eingesetzt. Der Betrieb dieses Datenerfassungsprogramms sowie die Betriebsparameter für den Einsatz mit dem *CdZnTe*-Detektor sind ausführlich in [Suc04] beschrieben.

2.1.1 Zeitstempel

Das System arbeitet mit einem Takt von 24.0 MHz, welcher von einem Quarz-Oszillator im *RENA System Interface* erzeugt wird und über das Buskabel an alle Boards verteilt wird. Aus diesem Takt wird im FPGA des Boards durch einen 1:8

Abbildung 2.5: Das *RENA System Interface*
Deutlich zu erkennen sind links hinten die LVDS-Treiber, in der Mitte das FPGA und rechts die Komponenten der Spannungsversorgung. Auf der Frontseite befinden sich, neben einer Reihe Kontroll-LEDs und der Reset-Taste, die Koaxialsteckplätze für die Bias-Spannung, der Analogausgang und die „X"-Leitung, die bei der hier eingesetzten Version der *RENA Module Board* nicht genutzt wird.

Frequenzteiler ein Sekundärtakt von 3.0 MHz erzeugt, der als Zeitbasis für einen 16 Bit breiten Zähler dient. Sollte der Detektor ein Ereignis auslösen, wird dieser Zählerstand als *Zeitstempel* dem Ereignisdatensatz, im folgenden *Hit* genannt, hinzugefügt. Der Zeitstempel hat somit eine Auflösung von $1/3 \cdot 10^{-6}$ s und kann eine maximale Zeitspanne von $2^{16} \cdot 1/3 \cdot 10^{-6} = 2.18$ ms erfassen, bevor der Zähler überläuft und wieder mit Null anfängt.[10]

Bei Messungen mit niedrigen Zählraten von ca. 500 Ereignissen pro Sekunde kann es vorkommen, dass der Zeitabstand zwischen zwei aufeinanderfolgenden Ereignissen so groß ist, dass zwei oder mehr Rollovers stattfinden und somit der reale Zeitabstand nicht rekonstruiert werden kann.

Bei langen Zeitabständen zwischen Ereignissen sammelt sich zudem Ladung aus Kriechströmen des Chips im Eingangsverstärker. Nach einem regulären Trigger verursacht das beim Auslesen des Kanals einen zu hohen Wert der gemessenen Spannung. Um dies zu vermeiden werden alle Kanäle regelmäßig zurückgesetzt.

[10] Für dieses Verhalten wird in dieser Arbeit der gebräuchlichere englische Terminus *Rollover* benutzt.

Wenn nach einer voreingestellten Zeitspanne nach einem Ereignis kein weiterer Trigger ausgelöst wird, sendet ein s.g. *Watchdog Timer* den Chips ein *Forced Reset* Signal und erzwingt dadurch die Leerung der Ladungsverstärker. Es wird dabei ein besonders markierter Hit erzeugt, dessen einzige signifikante Information der Zeitstempel ist (s. Abschn. 2.2.1).

Das Zeitintervall zwischen dem letzten regulären Trigger und dem *Forced Reset*, kann im Bereich von $10 - 2560\,\mu$s eingestellt werden. Ein kleiner Wert verbessert die Energiemessung, hat aber einen nachteiligen Einfluß auf die Totzeit. *NOVA R&D* empfiehlt einen Wert von $560\,\mu$s.

Wie im Abschnitt 5.2.1 gezeigt wird, ist es möglich, die *Forced-Reset*-Hits zu nutzen, um den Ereigniszeitpunkt auch bei sehr niedrigen Raten eindeutig zu bestimmen.

2.2 Datenverkehr auf dem Bus

Das *RENA System Interface* sendet auf dem vier Bit breiten Datenbus Kontroll- und Konfigurationsbefehle an die *RENA Module Boards* in Form von drei aufeinanderfolgenden Paketen (Nibble[11]).

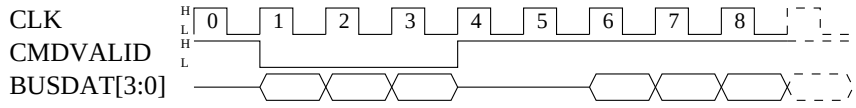

Abbildung 2.6: Struktur des Datenpakets
Der Wechsel von *High* nach *Low* von CMDVALID signalisiert den *RENA Module Boards* den Beginn einer Befehlssequenz (Takt-Zyklen 1, 2 und 3). Unmittelbar nach der Rückkehr von CMDVALID zu logisch eins (H) beginnen die Boards den Befehl zu interpretieren. Auf Befehle des Typs *Broadcast* geben die Boards keine Antwort, Konfigurationsbefehle werden mit der *Acknowledge* Sequenz $x5A$ in den Zyklen 6 und 7 quittiert, eventuell gefolgt von einem weiteren Nibble. Der Auslesebefehl kann nach dem *Acknowledge* mit bis zu 160 weiteren Datenpaketen beantwortet werden.

Die Kommunikation wird grundsätzlich vom Interface initiiert, indem die Leitung CMDVALID auf den Zustand *Low*[12] gesetzt wird (s. Abb. 2.6) und die hexadezimale Adresse des angesprochenen Boards an den Datenbus für die Länge eines

[11] Eng. *kleines Stück*, bezeichnet in der EDV die Hälfte eines Bytes, also eine Informationseinheit aus vier Bits. In den folgenden Abschnitten wird im Zusammenhang mit dem Datenverkehr auf dem *RENA*-Bus diese Bezeichnung benutzt. Angaben über ihren numerischen Inhalt werden meistens in der hexadezimalen Form mit vorangestellten x dargestellt, für die Binärdarstellung wird das Präfix b benutzt, z.B. $x5A$ oder $b0101\,1010$ für zwei Nibbles mit den dezimalen Werten 5 und 10. Um eine bessere Lesbarkeit zu erreichen, werden oft Gruppen von vier Ziffern durch ein Leerzeichen getrennt.

[12] In der Digitaltechnik werden mit *Low* und *High* Spannungniveaus der Signalleitungen bezeichnet, die den logichen Zuständen 0 und 1 bzw. den Wahrheitswerten *Falsch* (*False*) und *Wahr* (*True*) entsprechen.

Takt-Zyklus angelegt wird. Es folgen zwei weitere Zyklen mit dem aus zwei Nibbles bestehenden eigentlichen Befehl. Nach dem Senden der drei Befehlsteile geht CMD-VALID wieder auf *High* und das Interface geht in den Empfangsmodus zurück und wartet auf die Antwort des angesprochenen Boards.

NOVA R&D hat einen Satz von Befehlen für die Boards definiert, die in zwei Klassen unterteilt werden. Einige dieser Befehle dienen der Konfiguration der Boards sowie ihrer Komponenten und der Einstellung der Messparameter, die anderen dienen der Kontrolle des Messbetriebs (siehe Tabelle 2.2).

In der ersten Spalte der Tabelle 2.2 sind die mnemonischen Symbole der Befehle aufgelistet. Die zweite Spalte enthält die hexadezimalen Werte der Befehlsfolge. Das erste Nibble, mit dem Stellvertreterzeichen <ID>, ist die Identifikationsnummer des adressierten Boards.

Bei den meisten Befehlen kann das letzte Nibble einen von zwei Werten annehmen (die Auswahlmöglichkeiten sind in eckige Klammern gesetzt, getrennt durch ein *Pipe*-Zeichen „|“).

Mit Ausnahme des Auslesebefehls MB_FIFO_READ sind die Befehle für den Messbetrieb sogenannte *Broadcast*-Befehle, das heißt die Befehle sind nicht an ein spezielles Board adressiert. Sie sprechen alle Boards gleichzeitig an und erwarten daher keine Antwort. Das Stellvertreterzeichen ist in diesen Fällen <X>, da der tatsächlich übertragene Wert keine Bedeutung hat.

Die Antworten des Boards auf die Konfigurationsbefehle, in der dritte Spalte der Tabelle, beginnen immer mit der *Acknowledge*-Sequenz <ACK0> <ACK1> $=x5A$. Im Falle eines Schiebebefehls wird das niedrigwertigste Bit (LSB[13]) der Sequenz dem jeweiligen Register als LSB hineingeschoben. Das höchstwertigste Bit (MSB[14]) im Register wird herausgeschoben und als letztes Nibble der Antwort angehängt, so dass der alte Inhalt des Registers ausgelesen werden kann.

2.2.1 Datenformat

Die komplexeste Antwort wird auf dem Auslesebefehl MB_FIFO_READ gegeben. Der *Acknowledge*-Sequenz folgt ein Nibble mit der Anzahl (zwischen 0 und 15) der mitgesendeten Hits.

Bis zu 32 Hits können von einem im FPGA des Boards realisierten FIFO[15]-Puffer zwischengespeichert und zum Auslesen bereitgestellt werden. Jeder Hit ist 40 Bit breit und wird als eine Sequenz von 10 Nibbles übertragen, angefangen mit dem

[13] Least Significative Bit
[14] Most Significative Bit
[15] First In First Out

Symbol	Hex. Wert	Antwort	Funktion
Betrieb			
MB_FIFO_READ	<ID> 00	<ACK0> <ACK1> <CNT> <HITS>	Ruft Daten von<CNT> (0 bis 15) Hits auf
MB_FIFO_CLEAR	<X> 10	keine	Leert den Hit-Puffer
MB_TIMER_CLEAR	<X> 11	keine	Zeitstempel auf Null setzten
MB_WRITE_RUNFF	<X> 1[2|3]	keine	Schaltet den Messbetrieb aus (2) oder ein (3)
Konfiguration			
MB_RENA_SHIFT	<ID> C[0|1]	<ACK0> <ACK1> <OUT>	Löscht (0) oder setzt (1) das MSB im Schieberegister des RENA-Chips, der alter Wert wird in <OUT> zurückgegeben
MB_FEN_SHIFT	<ID> C[2|3]	<ACK0> <ACK1> <OUT>	Löscht (2) oder setzt (3) das MSB im *Force-Enable* Schieberegister
MB_DAC_SHIFT	<ID> A[0|1]	<ACK0> <ACK1> <OUT>	Löscht (0) oder setzt (1) das MSB im DAC-Register und deaktiviert den DAC
MB_DAC_SHIFTEX	<ID> A[2|3]	<ACK0> <ACK1> <OUT>	Löscht (2) oder setzt (3) das MSB im Register, schließt die Konfiguration des DAC ab und reaktiviert ihn
MB_SHAD_SHIFT	<ID> D[0|1]	<ACK0> <ACK1> <OUT>	Löscht (0) oder setzt (1) das MSB im *shadow* des CSR
MB_SHAD_TO_CSR	<ID> $D2$	<ACK0> <ACK1>	Kopiert den Inhalt des *shadows* in den CSR
MB_CSR_TO_SHAD	<ID> $D3$	<ACK0> <ACK1>	Kopiert den Inhalt des CSR in den *shadow*

Tabelle 2.2: *RENA*-Befehle [NOV01]

MSB. Die dem *RENA*-Handbuch [NOV01] entnommene Tabelle 2.3 zeigt den Inhalt und das Format eines Hit-Datensatzes.

Je nach Auslesemodus werden entweder nur die Hits derjenigen Kanäle übertragen, welche einen Trigger ausgelöst haben, oder auch zusätzlich die Hits der dem auslösenden Kanal unmittelbar benachbarter Kanäle. Es ist alternativ auch möglich die Hits aller durch eine entsprechende Bitmaske des Chip-Registers eingeschalteten Kanäle zu übertragen.

Tabelle 2.3: Format des HIT-Datensatzes [NOV01]

Bit Nr.	**Inhalt**
HIT[39]	Forced-Reset Hit Wenn 1, wurde der Datensatz durch ein Forced-Reset verursacht. Nur der Zeitstempel ist signifikant.
HIT[38]	FIFO Überlauf Wenn 1, war der FIFO vor der Annahme dieses Hits voll. Einige Hits könnten verloren gegangen sein.
HIT[37]	Abgeschnittenes Ereignis Wenn 1, wurde die maximale Anzahl der Hits für dieses Ereignis überschritten.
HIT[36:23]	ADC-Daten, 14 Bits
HIT[22:16]	Kanalnummer, 7 Bits Bezieht sich nur auf den Kanal auf dem ausgelesenen Board. Die eindeutige Identifizierung des Detektorkanals setzt sich aus dieser Zahl und der Board-ID zusammen.
HIT[15:0]	Zeitstempel, 16 Bits

Die maximale Anzahl der zu übertragenen Hits pro Ereignis kann auch auf einen Wert zwischen 1 und 15 Hits begrenzt werden. Sollte ein Ereignis mehr Kanäle ansprechen als in dieser Grenze angegeben sind, wird der letzte als *truncated* markiert (Bit 37 im Hit-Datensatz).

Alle von einem Ereignis generierten Hits werden am gleichen Wert des Zeitstempels erkannt.

2.3 Leistungsgrenzen

Wie bereits erwähnt, können mit einem Auslesebefehl bis zu 15 Hits von einem *RENA Module Board* zum Interface übertragen werden. Diese Operation kann bis

zu 168 Zyklen des 24.0 MHz Takts, also 7 μs dauern. Wenn eine kleine Wartezeit von 2 Taktzyklen (0,08 μs) zwischen zwei aufeinanderfolgenden Lesevorgängen eingerechnet wird, ergibt sich eine maximale Datenrate von $2,1 \cdot 10^6$ Hit/s.

Da jeder Hit eine Länge von 10 Nibbles bzw. 5 Bytes besitzt, sollte das System aus *RENA System Interface* und *DIO-96*-Schnittstelle bei Auslastung der angesteuerten *RENA Module Boards* respektive angeschlossenen Detektoren eine Übertragungsrate von 10,1 MByte/s bewältigen.

Messungen sowohl am IAAT als auch am UCSD/CASS haben einen maximalen Durchsatz dieses Systems von etwa 50.000 Hit/s gezeigt, der somit weit unter den Möglichkeiten der *RENA Module Boards* liegt.

Eine sorgfältige Analyse des Systems hat folgende Wege aufgezeigt, um eine Steigerung des Datendurchsatzes zu erreichen.

Als offensichtliches Nadelöhr wurde die eingesetzte Datenerfassungskarte für den PCI-Bus identifiziert, die laut Handbuch [Nat96, App. A] nur eine Transferrate von maximal 780 KByte/s leisten kann.

Aus der Analyse der VHDL-Quellen[16] des *RENA System Interface* ergibt sich, das für Daten nur ein Puffer von 13 Byte verwendet wurde, eventuell wegen der sehr begrenzten Ressourcen des *Spartan XCS20* FPGAs. Dies schränkt die Transferrate zusätzlich ein, z.B. für den Fall dass der Datenerfassungscomputer mit anderen Aufgaben beschäftigt ist und der Datenstrom kurzfristig unterbrochen werden muss.

Um sowohl die Leistungfähigkeit zu erweitern, als auch aus dem Wunsch nach höherer Flexibilität und der Möglichkeit Treiber und Software auch für das Linux Betriebssystem realisieren zu können, ist das Projekt entstanden, das *RENA System Interface* neu zu entwickeln.

[16] VHSIC Hardware Description Language. VHSIC steht für *Very High Speed Integrated Circuit*. Ein hervorragendes Werk über VHDL findet sich in [Ash02], ein einfaches und leicht verständliches Einstieg ist [ST04]. Siehe auch Anhang A.1

Kapitel 3

RENA Control Interface

Das *RENA Control Interface* (RCI) ist eine Weiterentwicklung des *RENA System Interface* von *NOVA R&D*. Es besteht aus einer speziell entwickelten Versorgungsplatine und der FPGA-Hardware die in Kapitel 4 besprochen wird.

Das Ziel dieser Arbeit war nicht, ein komplett neues Gerät zu konstruieren, sondern vor allem eine leistungsfähigere Schnittstelle für den Datenaustausch mit dem PC zu entwickeln. Die Anforderungen, welche die Kompatibilität zum *RENA System Interface* stellt, können wie folgt zusammengefasst werden:

- Bereitstellung der drei Versorgungsspannungen von $3,3\,\mathrm{V}$ und $5,8\,\mathrm{V}$ bei $1,5\,\mathrm{A}$ und $11,0\,\mathrm{V}$ bei $0,5\,\mathrm{A}$, ausreichend für die Versorgung von 8 *RENA Module Boards*
- Versorgung der Boards mit dem Taktsignal von 24 MHz
- Differentielle Buskommunikation (LVDS)
- Verstärker für das analoge Testsignal mit Bandbreite von 160 MHz und Verstärkung von 95 dB
- Konfiguration der FPGAs auf den *RENA*-Boards
- Direkte Weiterleitung der *RENA*-Befehle vom PC an die Boards
- Empfang der Antworten der Boards auf *RENA*-Befehle und Bereitstellung zum Auslesen durch den PC
- Möglichkeit zur Adressierung von bis zu 16 Boards
- Kontinuierliches, zyklisches Auslesen aller am Bus angeschlossenen Boards

Die neue Schnittstelle soll die Aufgaben des *RENA System Interface* ohne Abstriche erfüllen können aber vor allem dazu fähig sein, seine maximale Datenrate von ca. 50.000 Hit/s wesentlich zu übertreffen. Idealerweise sollte die rechnerische maximale Rate der *RENA Module Boards* von $2,1 \cdot 10^6$ Hit/s erreicht werden.

Darüberhinaus soll, speziell für Zeitreihenmessungen, auch eine Mögligkeit gegeben werden, die Uneindeutigkeit der Zeitinformation bei niedrigen Zählraten, über die im Abschnitt 2.1.1 berichtet wurde, zu beheben ohne die Kommunikation mit der

Übertragung der ansonsten unnötigen *Forced-Reset*-Hits zu belasten. Ausserdem sollen die Zeitstempelzähler mit einem genaue Zeitgeber, z.B. einem GPS- oder Funkuhrempfänger, jede Sekunde synchronisiert werden können.

Weiter sollen Verbesserungen und zusätziche Leistungsmerkmale realisiert werden, um eine höhere Flexibilität bei dem Einsatz mit moderne PC-Systeme im Labor zu erlauben. Unter Anderem soll ein größerer Abstand zwischen PC und Messelektronik als der mit dem *RENA System Interface* derzeit mögliche Abstand von einem Meter erlaubt sein. Dies ist sowohl wegen der besseren Möglichkeit zur Abschirmung der Detektoren vor dem störenden Einfluß eines PCs und seines Bildschirms erwünscht, als auch um einen besseren Schutz des Bedienpersonals vor der zu messende Röntgenstrahlung zu gewährleisten.

Als Kommunikationsschnittstelle hat sich der *Universal Serial Bus* (USB[1]) als eine gute Alternative zur *DIO-96* Schnittstelle erwiesen, da er alle gestellten Anforderungen in Bezug auf Leistung, Verfügbarkeit und Zuverlässigkeit erfüllt. Durch seine hohe Verbreitung findet man geeignete und preiswerte Komponenten auf dem Markt. Dank der verfügbaren Dokumentation ist es möglich, Software für den Einsatz unter Windows oder Linux mit vertretbarem Aufwand zu entwickeln.

Ein leistungsfähiges FPGA sollte alle Verwaltungsaufgaben und die Kommunikation sowohl mit den *RENA Module Boards* als auch mit dem USB-Chip übernehmen. Gleichzeitig sollte das RCI, aus der Sicht der *RENA Module Boards*, soweit möglich zum alten Interface kompatibel gestaltet werden. Vor allem die elektrischen Eigenschaften des Datenbuses und die Qualität und Leistung der Stromversorgung sollten dem Original möglichst ähnlich sein. Es war daher pragmatisch, aus den von *NOVA R&D* mit dem System gelieferten Schaltplänen die Teile, die nicht an der Kommunikation mit dem Auswertecomputer beteiligt sind, zu übernehmen.

Der Schwerpunkt dieser Arbeit besteht in der Entwicklung eines VHDL-Designs für die Konfiguration des FPGAs, auf das in den Kapiteln 4 und 5 eingegangen wird. Die folgenden Abschnitte beschreiben zunächst nur die Hardware, also die Versorgungsplatine des RCIs und die daran angeschlossenen Zusatzmodule.

3.1 „Swatter“

Das für diese Entwicklung gewählte FPGA ist ein *XILINX Virtex-II XC2V1000*, es bietet für diesen Zweck ausreichend große Ressourcen und befindet sich, zusammen mit einem USB-Chip vom Typ *Cypress AN2131* auf dem Entwicklungsmodul *Swatter* der Firma *DD&T*.

[1] Anhang B stellt eine kurze Einführung in die USB Technologie dar, darin werden hauptsächlich die Struktur und die Eigenschaften der Hardware und das *Low-Level*-Kommunikationsprotokoll beschrieben.

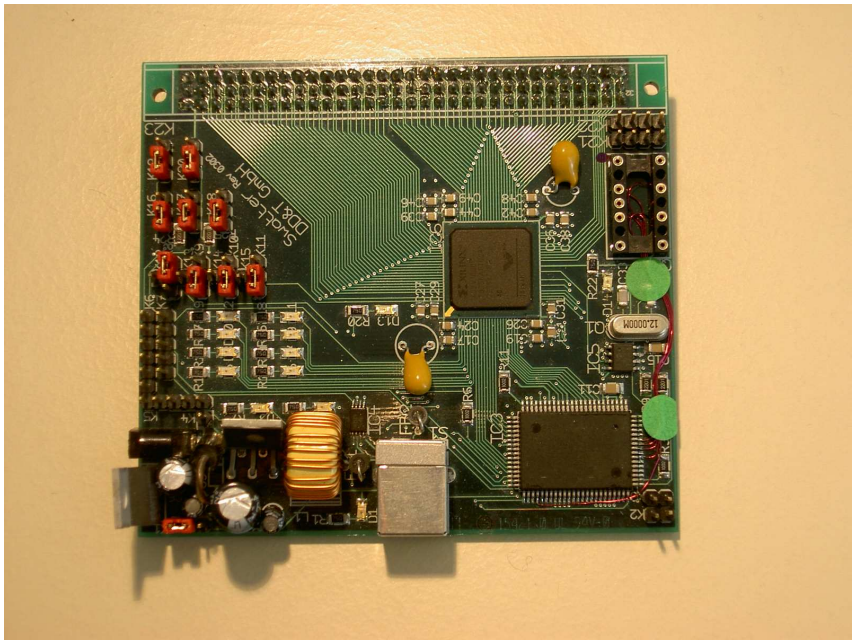

Abbildung 3.1: Der Swatter von DD&T
Das FPGA, leicht rechts von der Mitte, ist ein *Virtex-II XC2V1000*. In der unteren rechten Ecke befindet sich das USB-Chip *Cypress AN2131* und unten links die Stromversorgung. An der obere Kante erkennt man die Kontakte des an der Unterseite angebrachten *VG96*-Steckers für den Anschluß an den Prototyp.

Ein solches Modul eignet sich speziell für die Entwicklung von Prototypen, da es nur eine oder wenige Kernkomponenten beinhaltet, deren Einbau in einem Prototyp zu teuer und zu aufwendig wären.

Das FPGA auf dem Swatter hat ein Gehäuse in *FG256*-Ausführung. Seine 256 Kontakte sind auf der Unterseite in einer Matrix von $16x16$ kleinen Lötkugeln im Abstand von 1 mm angeordnet (Ball Grid Array, BGA). Da Komponenten dieses Typs nur in spezialisierten Werkstätten auf die Platinen gelötet werden können, erhöht dies zusätzlich die Kosten und den Zeitaufwand für die Realisierung eines Prototyps. Daher werden solche Komponenten auf einem separaten Modul zur Verfügung gestellt, das einfach auf die verschiedenen Entwicklungsstufen der neuen Platine eingesteckt werden kann.

Die Konfiguration des FPGAs kann beim Swatter nur über die JTAG-Schnittstelle[2]

[2] *Joint Test Action Group* ist der Name eine Arbeitsgruppe, die ein Verfahren zur Fehlersuche (Debugging) bei Hardware im Jahre 1985 entwickelt hat. Das Kürzel JTAG ist die Bezeichnung für das Verfahren selbst und für die damit in IEEE 1149.1 (Norm des amerikanisches *Institute of Electrical and Electronics Engineers*) standardisierte serielle Schnittstelle und wird inzwischen vermehrt zur Konfiguration von FPGAs bzw. CPLDs (Complex Programmable Logic Device, s. Anh. A) und zur Programmierung von Mikroprozessoren verwendet.

erfolgen, da er über keinen nichtflüchtigen Speicher für die Konfigurationsdaten verfügt. Um einen *Stand-Alone* Betrieb zu ermöglichen, wurde am IAAT ein zusätzliches Speichermodul (s. Abb. 6.1) entwickelt, das nach geringfügigen Veränderungen am Swatter (einige Leitungen mußten zusätzlich verbunden werden) huckepack auf das Board gesteckt werden kann. Die JTAG-Schnittstelle kann dann dazu benutzt werden, die Konfiguration wahlweise direkt in das FPGA oder in das EEPROM zu laden.

Der USB-Chip des Swatters entspricht der USB-Spezifikation 1.1, unterstützt also den *Full-Speed*-Betrieb (siehe Anhang B) mit einer nominalen Tranferleistung von 12 MBit/s.

Der Swatter wurde eingesetzt, obwohl es sich dabei um einen Prototyp handelt, und die maximale Bandbreite des USB 1.1 Chip von 9,7 MBit/s die Leistung des *RENA System Interface* nur um den Faktor 1,5 übersteigt.

Die in Aussicht gestellte nächste Version des Swatters soll ein FPGA gleichen Typs besitzen sowie die gleiche Pinbelegung aufweisen und zusätzlich über ein serielles EEPROM und einen USB 2.0 Chip verfügen.

Um nicht auf die Fertigstellung des neuen Swatters warten zu müssen und trotzdem schon mit USB 2.0 arbeiten zu können, wurde provisorisch ein Stück Lochrasterplatine am RCI angebracht und an freie Pins der VG96-Buchse gelötet (s. Abb. 3.4). Dies erlaubte den Einsatz eines separaten USB-2-Moduls.

3.2 Eingesetztes USB-Modul

Das *USB High Speed Interface Modul 2.5* der Firma *BrainTechnology* in Frankfurt/Main basiert auf dem USB-Chip *Cypress CY7C68013*.

Dieses Modul stellt drei 8 Bit breite I/O-Ports sowie einige Kontrollleitungen zur Verfügung, die je nach Betriebsmodus verschiedene Funktionen erfüllen.

Die Firmware kann in einem 8 KByte großen seriellen EEPROM gespeichert werden, und wird in diesem Fall von einer getrennt zu erwerbenden DLL[3] auf den Chip übertragen.

Die DLL stellt zugleich eine Programmierschnittstelle (Application Programming Interface, API) mit einer großen Anzahl von Funktionen bereit, mit der, beim Programmieren eigener Windows-Anwendungen, auf sehr einfache Weise auf die Funktionalität des Moduls zugegriffen werden kann.

[3] Dynamic-Link Library, eine Bibliothek für das Betriebssystem Microsoft Windows. Eine DLL ist ein Programm das keine eigene Benutzeroberfläche bereitstellt. Die in ihr implementierten Funktionen werden dynamisch, d.h. nach Bedarf vom Betriebssystem in den Speicher geladen und dem aufrufenden Programm zur Verfügung gestellt. Wenn diese Funktionen nicht mehr benötigt werden, entfernt sie das Betriebssystem wieder aus dem Speicher.

Der Entwickler der DLL und der Firmware hat auch über GPIF 10 verschiedene Betriebsmodi für parallele Datenübetragungen auf 8 und 16 Bit implementiert, sowie die Möglichkeit alle 24 Leitungen der 3 I/O-Ports voneinander unabhängig zu lesen oder zu setzen.

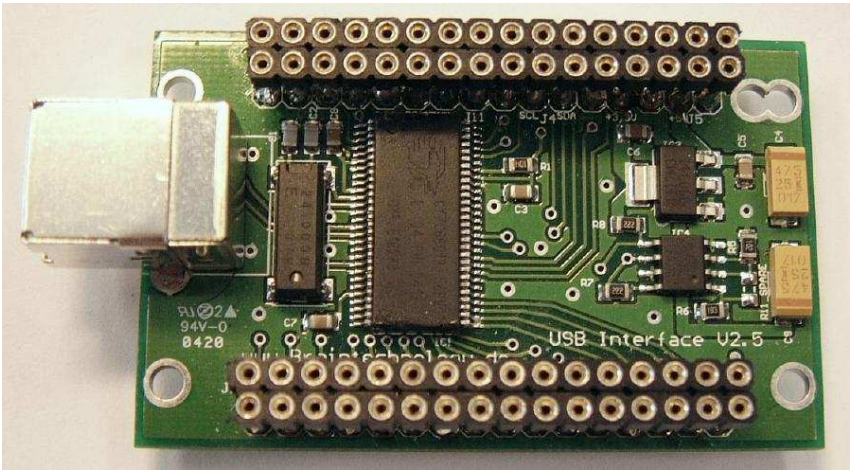

Abbildung 3.2: USB-Modul von BrainTechnology

3.3 RCI-Versorgungsplatine

Der Haupteil des RCI wird als Hardware innerhalb des FPGAs erzeugt. Da der Swatter alle für den Betrieb des FPGAs notwendigen Komponenten beinhaltet und genügend I/O-Ports über den VG96-Stecker zur Verfügung stellt, war es ausreichend eine Platine zu entwerfen, die über einen VG96-Anschluß den Swatter an den LVDS-Treiber anschließt und die Stromversorgunskomponenten unterbringt.

Mit Hilfe des Softwarepakets *Altium Protel2004* wurden sowohl der Schaltplan (siehe Abbildung 3.3) entworfen als auch die doppelseitige Platine im Format $10 \times 16\,\mathrm{cm}^2$ (Euro-Standand) entflochten. Es wurden dabei Bauteile in klassischer und in SMD[4]-Technologie eingesetzt.

3.3.1 Stromversorgung

Eine der drei für den Betrieb der *RENA Module Boards* notwendigen Versorgungsspannungen (VSUPPLYB $= +11,0\,\mathrm{V}$) und die Hauptversorgung (Vcc $= +5,0\,\mathrm{V}$)

[4]Surface Mounted Device

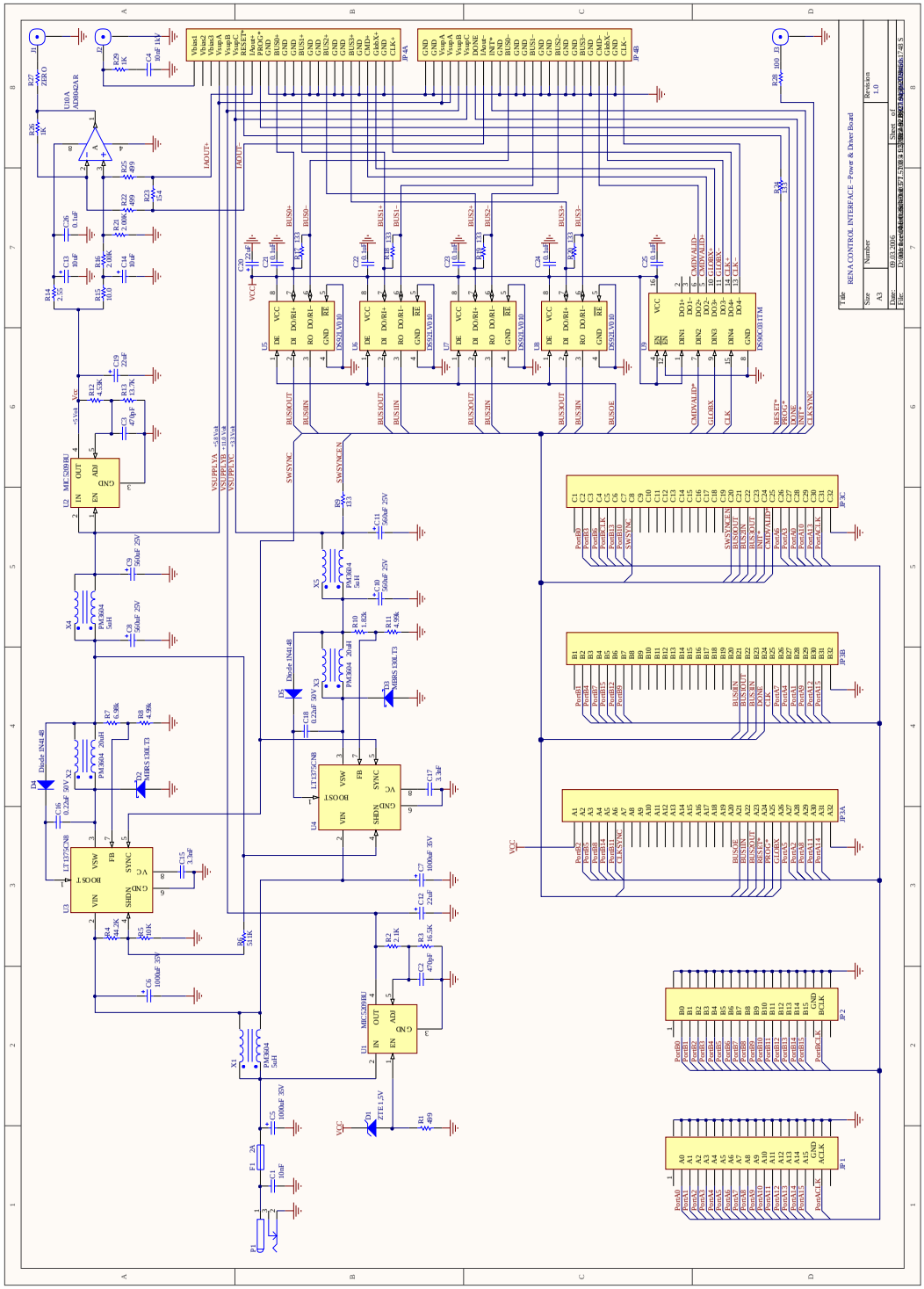

Abbildung 3.3: *RENA Control Interface*, Schaltplan

aller Komponenten des RCIs und des Swatters, werden von dem rauscharmen Spannungsreglern U1 und U2, beide vom Typ *MICREL MIC5209*, erzeugt.

Da es nicht möglich war, die Originalteile *LT1763* von *Linear Technology* auf dem europäischen Markt zu finden, war es in diesem Fall notwendig, sie durch den Typ *MIC5209* zu ersetzen.

Beide Typen haben sehr ähnliche Leistungsmerkmale, benötigen aber leicht unterschiedliche Anordnungen der Steuerkomponenten, so dass es hier Abweichungen von der Originalschaltung gibt.

Die restlichen zwei Versorgungsspannungen der Boards (VSUPPLYA = +5, 8 V und VSUPPLYC = +3, 3 V) werden von den integrierten Schaltnetzteilen U3 und U4 vom Typ *Linear Technology LT1375* erzeugt. Zur Minimierung des Rauschens werden ihre internen Oszillatoren, durch ein Referenzsignal von 600 KHz mit dem Takt des FPGAs synchronisiert (SWSYNC). Dieses Signal wird aus dem Systemtakt von 48 MHz durch einen 1:80 Frequenzteiler im FPGA erzeugt und, entsprechend der Herstellerspezifikationen, erst dann an die Schaltnetzteile geliefert, nachdem eine positive Spannung an der Leitung SWSYNCEN an die Schaltnetzteile gelegt wurde.

Die Schaltnetzteile sind für eine maximale Last von 1, 5 A und die Spannungsregler für einen Strom von 0, 5 A ausgelegt, was laut der Dokumentation von *NOVA R&D* [NOV01] ausreicht, bis zu 8 Boards am Bus zu betreiben.

Beide Typen von Stromversorgungskomponenten können abgeschaltet werden, wenn die Spannung an ihren Freigabepins die jeweiligen Referenzwerte unterschreitet. Durch entsprechende Spannungsteiler-Netzwerke an diesen Pins kann die Abschaltung des gesamten Systems veranlasst werden, falls die Spannung an der Versorgungsbuchse P1 unterhalb den für den Betrieb erforderlichen Wert von 13 V liegen sollte.

Die maximal erlaubte Spannung an P1 liegt bei 19 V. Es wurde keine Schutzvorrichtung gegen die Überschreitung dieses Pegels eingebaut.

3.3.2 LVDS-Bustreiber

Die bidirektionale Konvertierung der CMOS-Signale des FPGAs, die an der VG96-Buchse JP3 anliegen, in das differentielle Format des *RENA*-Datenbus (siehe Abschnitt 2.1 bzw. Tabelle 2.1) wird von den vier LVDS-Einzeltransceivern U5 bis U8 vom Typ *National Semiconductor DS92LV010A* übernommen.

Die Umschaltung von Empfang- zum Sendebetrieb erfolgt durch Anlegen einer positiven Spannung am *Driver Enable* (DE) Eingang über die Signalleitung BUSOE des FPGAs. Zwischen dem negativen und dem positiven Anschluß jedes Transceivers ist ein Widerstand von 133 Ω für die Terminierung des Busses angebracht.

Die IC U9 ist ein vierfacher LVDS-Treiber vom Typ *DS90C031TM* von *National Semiconductor* für die Konvertierung der drei unidirektionalen Steuersignale.

3.3.3 Weitere Anschlüsse

Das differentielle Analogsignal aus der Testleitung der Boards wird durch den Operationsverstärker U10A an der *LEMO*-Koaxialbuchse J1 zur Verfügung gestellt. Bei U10A handelt es sich um einen Hochleistungsverstärker mit einer Bandbreite von 160 MHz und einer maximalen Verstärkung von 95 dB.

Da der in dem *RENA System Interface* eingebaute Verstärker *AD8041* von *Analog Devices* nicht lieferbar war, wurde hier eine Hälfte des doppelt ausgeführten Verstärkers *AD8042* eingesetzt, der die gleichen Eigenschaften wie das Original aufweist.

Über die Koaxialbuchse J2 kann eine Spannung von bis zu 500 V für den Detektor-Bias angelegt werden. Nach einem Entstörfilter wird sie über den Bus-Stecker JP4 direkt an die *RENA Module Boards* geleitet.

Die Koaxialbuchse J3 dient der Einspeisung eines 1 Hz Signals für die Synchronisation der Zähler, welche die Zeitstempel erzeugen.

Neben der VG96-Buchse für den Swatter befinden sich auf der Platine zwei 40-polige Messerleisten für Flachbandkabel. Diese sind für den Anschluß eines *Logic Analyzers* vorgesehen und sind direkt mit der VG96-Buchse verbunden.

Das *RENA*-Buskabel wird an der SMD-Messerleiste JP2 angeschlossen.

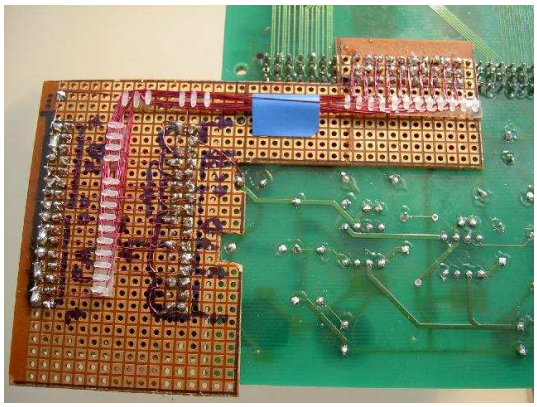

Abbildung 3.4: Zwischenlösung für USB
Das Bild zeigt die Unterseite der Lochrasterplatine, die den Swatter mit dem separaten USB-2-Modul verbindet. Die Pins der vier Pinleisten, auf das Modul aufgesteckt wird, wurden durch Einzeldrahtverbindungen an freien Pin des VG96 angeschlossen.

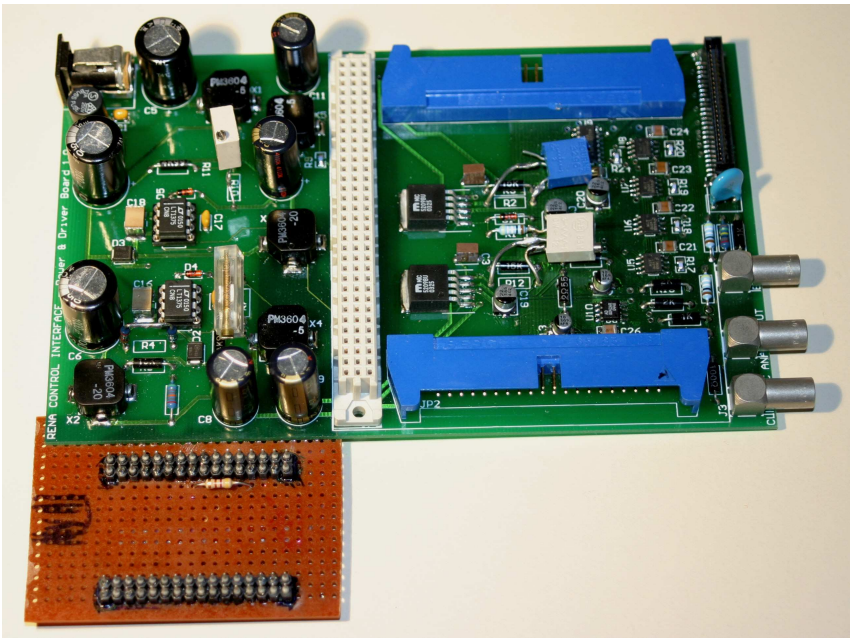

Abbildung 3.5: Vollständig bestückte Hauptplatine
Links von der VG96-Buchse sind die zwei Schaltnetzteile U3 und U4 untergebracht, da sie Elektrolytkondensatoren benötigen, die unter dem Swatter keinen Platz haben. Alle weiteren Komponenten sind sehr kompakt zwischen den blauen Messerleisten angeordnet. Die Widerstände R2, R7, R10 und R12 wurden durch Prezisionstrimmer ersetzt, um die erzeugten Spannungen fein einstellen zu können. Die braune Platine unten links wurde nachträglich eingesetzt um das USB-Modul anzuschließen (s. Abschn. 3.2).

Kapitel 4

VHDL-Design - Aufbau

Das Modell oder *Design* für die Konfiguration des FPGAs wurde mit Hilfe der *Hardware-Beschreibungssprache VHDL*[1] entworfen.

Das Blockdiagramm des RCIs ist in Abbildung 4.1 dargestellt. Das RCI selbst wird durch das blaue Rechteck, die Hardwarekomponenten durch gelbe Blöcke symbolisiert. Das FPGA entspricht der weißen Fläche, darin befinden sich die sechs logischen Komponenten, aus denen das RCI-Design besteht. Sie sind durch Pfeile, die den Datenfluss zwischen ihnen verdeutlichen, verbunden.

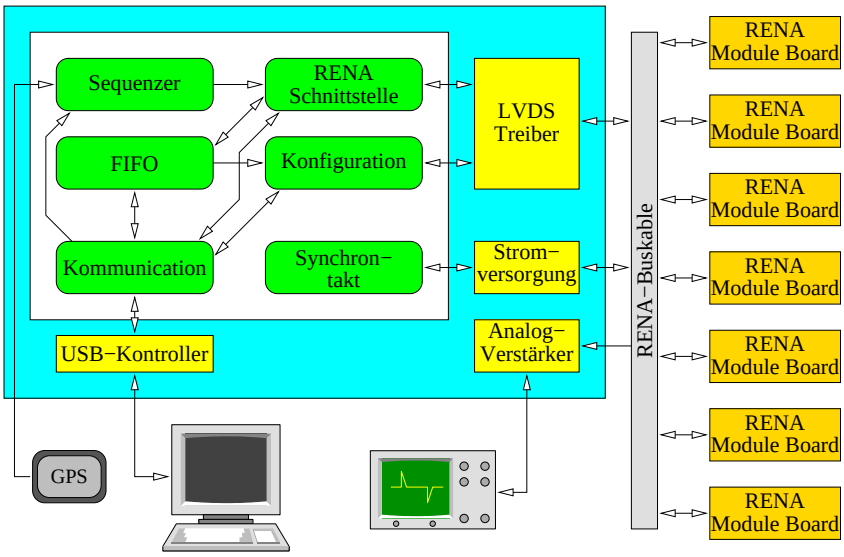

Abbildung 4.1: Blockdiagramm des RCIs

Auf der rechten Seite ist das RCI an das Bus-Kabel angeschlossen zusammen mit einigen *RENA Module Boards*, unten ist ein Oszilloskop mit dem analogen Überwachungsausgang und der Datenerfassungs-PC mit dem USB-Kontroller verbunden.

[1] Siehe Fußnote 16 auf Seite 18 sowie Anhang A.1

Das graue Kästchen unten links symbolisiert einen GPS-Empfänger, der den Sekundentakt für die Synchronisation der Zeitstempelzähler liefert.

Die Aufgaben des RCI werden von vier Modulen, die im Kapitel 5 näher erläutert werden, übernommen:

- **Die *RENA*-Schnittstelle** bereitet die *RENA*-Befehle, die entweder vom Sequenzer oder direkt von der Auswertungssoftware auf den PC über die Kommunikationsschnittstelle kommen, für die *RENA Module Boards* vor und speichert ihre Antwortdaten in dem FIFO-Puffer.
- **Der Auslesesequenzer** erzeugt die Befehlsfolge für das zyklische Auslesen aller installierten *RENA Module Boards* und die Synchronisation der Zeitstempelzähler.
- **Das Konfigurationsmodul** übermittelt den Konfigurationsbitstrom an die FPGAs der Boards und überwacht ihren Status.
- **Die Kommunikationsschnittstelle** sorgt für die Übertragung von Daten und Befehlen zwischen dem USB-Kontroller und den RCI-Modulen.

Beim FIFO-Puffer handelt es sich um eine von *XILINX* vordefinierte Standardkomponente (*Core*), die vom Benutzer auf die gewünschte Größe konfiguriert wird. Der Speicherpuffer hat eine Wortbreite entsprechend der Breite des USB-Datenbus von 16 Bits und, in diesem Stadium der Entwicklung, eine Tiefe von 256 Worten.

Das Symbol mit der Beschriftung *Synchrontakt* im Blockdiagramm (Abb. 4.1) stellt kein eigenständiges Modul dar. Es handelt sich um einen einfachen Frequenzteiler mit Freigabeport, wie in Abschnitt 3.3.1 erläutert, der als eigenständiger Prozess direkt innerhalb des Top-Level-Moduls (s. Abschn. 5.1) implementiert ist.

4.1 Funktionsprinzip

Das Funktionsprinzip des RCIs wird anhand eines vereinfachten Flussdiagramms, das in Abbildung 4.2 dargestellt ist, deutlich gemacht.

Bei der Inbetriebnahme lädt das FPGA des RCIs seine Konfiguration aus dem EEPROM (s. Abschn. 3.1 und Abb. 6.1) und ist nach wenigen Sekunden betriebsbereit. Da die *RENA Module Boards* kein EEPROM besitzen, werden ihre FPGAs mit Hilfe des RCIs konfiguriert. Die FPGAs der Boards signalisieren diesen Zustand durch ein *Low* auf einer spezielle Leitung des Buskabels. Wird dieser Zustand erkannt, steht dem Benutzer bzw. der Auslesesoftware nur ein begrenzter Funktionsumfang zur Verfügung.

Bei noch nicht betriebsbereiten *RENA Module Boards* kann nur der Inhalt des CSRs[2] ausgelesen werden, Schreibzugriffe dürfen in dieser Phase nur der Konfigu-

[2] Das *Control and Status Register* enthält einige Betriebsparameter und Statusinformationen

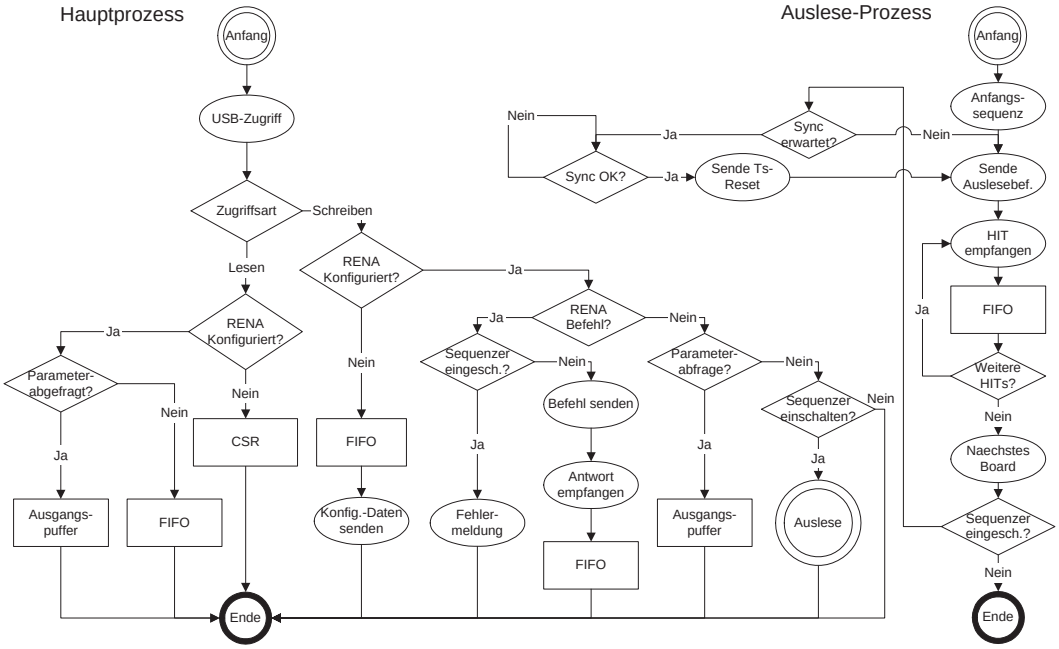

Abbildung 4.2: Flussdiagramm der RCI-Befehle
Der Hauptprozess wird bei der Inbetriebnahme gestartet und reagiert auf Zugriffe des USB-Kontrollers. Der Auslese-Prozess läuft unabhängig vom Hauptprozess und wird von einem entsprechenden Befehl gestartet.

ration der FPGAs der Boards dienen. Alle ankommenden Daten werden in den FIFO-Puffer geschoben und sofort an die Boards weitergeleitet.

Sind die FPGAs der Boards erfolgreich konfiguriert, werden Schreibzugriffe des USB-Kontrollers als Befehle interpretiert. Alle bisher definierten 16 Bit breiten Befehle sind in Tabelle 4.1 aufgelistet.

Bei Lesezugriffen wird in der Regel der Inhalt des FIFO-Puffers oder, bei leerem FIFO, ein *Null-Paket* ($x0000$) übertragen. Zwei der RCI-Befehle (die „RCI_GET“-Befehle) fordern Information über den momentanen Zustand des RCIs an. Da diese Information nur sehr kurzfristig von Bedeutung ist, werden die angeforderten Daten in einem Ausgangspuffer zwischengespeichert und bei dem diesen Befehlen unmittelbar folgenden Lesezugriff ausgegeben. Der Ausgangspuffer wird danach gelöscht und weitere Lesezugriffe liefern wieder den FIFO-Inhalt. Werden mehrere RCI_GET-Befehle hintereinander gesendet, bezieht sich der Inhalt des Ausgangspuffers nur auf den letzten Befehl. Wird anstatt des Lesezugriffs nach einem RCI_GET-Befehl ein anderer Befehl gesendet, dann wird der Ausgangspuffer sofort gelöscht.

Die Befehle RCI_SEQ_START, RCI_SEQ_START_FR und RCI_SEQ_STOP steuern den Betrieb des Auslesesequenzers. Dieser Prozess (rechts im Flussdiagramm,

Tabelle 4.1: RCI-Befehle

Symbol	Wert	Funktion
RCI_NULL	$x0000$	*Nullbebefehl*, hat keine Funktion
RCI_SEQ_START	$x001$<ID>	Startet die Auslesesequenz, <ID> ist die höchste ID-Nummer der installierten Boards
RCI_SEQ_START_FR	$x002$<ID>	Wie oben, überträgt auch *Forced-Reset-Hits*
RCI_SEQ_STOP	$x003$<X>	Stoppt die Auslesesequenz
RCI_GET_CSR	$x004$<X>	Setzt den Inhalt des CSRs in den Ausgangspuffer
RCI_GET_DCOUNT	$x005$<X>	Setzt die Anzahl der FIFO-Einträge in den Ausgangspuffer
RCI_FIFO_CLEAR	$x006$<X>	Leert den FIFO-Puffer
RCI_RENA_CONFIG	$xE000$	Reinitialisiert die FPGAs der Boards
RCI_RENA_RESET	$xE001$	*Reset*-Signal für die Boards
RCI_RENA_CMD	xF<CMD>	Sendet den 12 Bit breiten *RENA*-Befehl <CMD>

Abb. 4.2) läuft parallel zum Hauptprozess und davon völlig unabhängig. In einer Dauerschleife werden nacheinander alle installierten *RENA Module Boards* ausgelesen, die empfangene Hits-Datensätze ergänzt und formatiert und in den FIFO-Puffer gespeichert. Die Schleife wird $7,2\,\mu$s bevor ein neuer Puls für die Synchronisation der Zeitstempelzähler erwartet wird angehalten. Diese Zeitspanne reicht aus, den längstmöglichen Auslesevorgang zu beenden, bevor bei Ankunft des Pulses der *RENA*-Befehl für die Löschung der Zeitstempelzähler abgeschickt und die Schleife wieder gestartet wird.

Durch den Befehl RCI_RENA_CONFIG wird die Löschung der FPGA-Konfiguration aller *RENA Module Boards* erzwungen und eine erneute Konfiguration notwendig gemacht. Dagegen setzt RCI_RENA_RESET die Boards nur in den Ausgangszustand zurück, ohne die Konfiguration der FPGAs zu löschen.

Beim Befehl RCI_RENA_CMD handelt es sich nur um einen 4 Bit breiten Header, der einem *RENA*-Befehl vorgestellt wird. Die *RENA*-Befehle können so erkannt und sofort unverändert an die Boards weitergeleitet werden. Die Antworten der Boards auf diese Befehle werden auch formatiert und im FIFO-Puffer gespeichert. Solange der Sequenzer in Betrieb ist, werden diese Befehle vom RCI zurückgewiesen.

Der Befehl RCI_FIFO_CLEAR leert nur den FIFO-Puffer des RCIs. RCI_NULL hat keine besondere Funktion, das RCI quittiert den Befehl an den USB-Kontroller und ignoriert ihn. Er kann zum Testen der USB-Verbindung genutzt werden.

Der Adressbereich von $x0070$ bis $xDFFF$ ist frei und kann für zukünftige Zwecke genutzt werden.

4.2 Datenformate

Bedingt durch die Breite des Datenbus von 16 Bits, werden Daten in ebenso breiten Paketen vom RCI geliefert.

Entsprechend ihrem Inhalt, werden die Pakete mit einem 3 oder 4 Bit breiten Datenkopf (*Header*) an ihrem Anfang (der MSB) versehen, um sie unabhängig vom Zeitpunkt ihrer Abholung durch den Auswertungscomputer unterscheiden und einordnen zu können.

Tabelle 4.2 zeigt die bis zu diesem Zeitpunkt definierten Header und den Inhalt der Pakete.

Tabelle 4.2: Header und Inhalt der Datenpakete

Symbol	**Wert**	**Bedeutung**
HDR_NULL	$x0000$	*Null-Paket*
Adressbereich	$x0001 - x2FFF$	Nicht definiert, frei für künftige Anwendungen
HDR_CSR	$b0011$<CSR>	Inhalt des CSRs (12 Bits)
HDR_FIFO_COUNT	$b0100$<CNT>	Anzahl der FIFO-Einträge (Ganzzahl, 12 Bit)
HDR_RENA_FAIL	$b0101$<CMD>	Unbeantworteter *RENA*-Befehl
HDR_RENA_CMD	$b011$[0|1]<CMD>	Antwort und *RENA*-Befehl
HDR_HIT0	$b100$<HIT0>	Erster Teil eines Hits (13 Bits)
HDR_HIT1	$b101$<HIT1>	Zweiter Teil eines Hits (13 Bits)
HDR_HIT2	$b110$<HIT2>	Dritter Teil eines Hits (13 Bits)
HDR_HIT3	$b111$<HIT3>	Vierter Teil eines Hits (13 Bits)

Bei HDR_NULL, das wie der freie Adressbereich in der Tabelle in hexadezimaler Form dargestellt ist, handelt sich genaugenommen nicht um einen Header, sondern um das gesamte *Null-Paket*. Der Inhalt des 12 Bit breiten Control und Status Registers und des 4×13 Bit breiten Hit-Datensatzes ist in Tabelle 4.3 angegeben.

Tabelle 4.3: Formate der CSR- und Hit-Datensätze

Bit	Bedeutung
	Control und Status Register
11-9	Unbenutzt, frei für künftige Anwendungen
8	*RENA*-FPGAs sind konfiguriert
7	RCI-FIFO ist voll
6	RCI-FIFO ist leer
5	Speichern der *Forced-Reset*-Hits ist aktiv
4	Sequenzer in Betrieb
3-0	Höchste ID der installierten Boards
	Hit
51	RCI-FIFO war voll vor diesem Hit
50	Zeitstempel-Warnflagge
49	*Forced-Reset*-Hit
48	*RENA*-FIFO war voll vor diesem Hit
47	*Truncated-Event-Error*
46-41	Zeitstempel-Erweiterung
40-25	*RENA*-Zeitstempel
24-21	Board-ID
20-14	Kanalnummer
13-0	ADC-Daten

Kapitel 5

VHDL-Design - Detaillierte Darstellung

5.1 Top-Level-Modul

Das Top-Level-Modul eines VHDL-Designs bildet den Rahmen, in dem die einzelnen Module instanziert und miteinander verbunden werden. Abbildung 5.1 zeigt detailliert seine Struktur. Das Top-Level wird durch die schwarze Umrandung dargestellt, rechts und links außerhalb dieser Umrandung befinden sich die Ports seiner Schnittstelle, die zugleich die Schnittstelle des RCIs bilden.

Alle Module sind durch grüne Rechtecke dargestellt. Ihre Schnittstellen sind genau aufgeführt, mit Angabe über die Breite und die Richtung der Ports. Die Pfeile zeigen die Richtung und die Art der Verbindung der Signale.

Die gelben Symbole stehen für logische Bausteine, die durch *bedingte Signalzuweisungen* oder als Prozess, wie den oben erwähnten Frequenzteiler (ClkDiv), in dem *Architecture*-Teil des Moduls definiert werden. Die Eingangspuffer für die Leitungen *UTCSync*, *Clock* und *usbStrobe* sind feste Bestandteile der FPGA-Architektur und sollen in dem Design für Taktsignale oder andere periodische Signalen instanziert werden. Sie sorgen für die korrekte Benutzung der Takt-Verteilerbäume und für die Entzerrung des Signals. Professionelle Synthesewerkzeuge können allerdings in der Optimierungsphase solche Signale erkennen und ihnen die passenden Eingangspuffer automatisch zuweisen.

Bei der *Global-Reset*-Komponente handelt es sich um ein Signal, welches der Initialisierung des gesamten Modells bei der Inbetriebnahme dient und vom FPGA zur Verfügung gestellt wird.

Bis auf einen Prozess der Befehlschnittstelle (siehe Abschnitt 5.2), arbeiten alle Module des RCIs mit einem Systemtakt von 48 MHz.

5.2 RENA-Befehlsschnittstelle

Die Befehlschnittstelle *CmdInterface* ist das Bindeglied zwischen dem *RENA Control Interface* und den *RENA Module Boards*. Aus dem Systemtakt wird in diesem Modul der für den Betrieb der Boards notwendige Takt von 24 MHz erzeugt. Er wird sowohl über den Ausgangsport *busClk* auf den *RENA*-Bus übertragen als auch intern über das Signal *clk24* genutzt.

In dem *CmdInterface* laufen zwei voneinander unabhängige Prozesse. Der erste, der *comm*-Prozess, ist ein mit dem *clk24* getakteter Zustandsautomat (FSM[1]) der die Übermittlung der RENA-Befehle (s. Tab. 2.2) an die Boards und den Empfang ihrer Antworten übernimmt. Im Wartezustand *idle* überwacht die FSM den Eingang *Exec*. Wird auf diesem Eingang der Zustand *High* erkannt, wechselt sie in den Zustand *tx* und übermittelt den 12 Bit breiten Befehl, der am Eingang *Cmd* vorliegt, in drei Taktzyklen an den 4 Bit breiten *RENA*-Bus. Sollte es sich um einen *Broadcast*-Befehl handeln, kehrt die FSM sofort in den Wartezustand zurück, da die Boards diese Art von Befehlen nicht beantworten, sonst wechselt sie in den Empfangszustand.

Im Zustand *rx* wird bei jedem Takt der Inhalt des Busses in ein Schieberegister gespeichert. Wird nicht innerhalb von neun Taktzyklen das Bestätigungsmuster <ACK> erkannt, erstellt die FSM einen Datensatz in einem dafür vorgesehenen 51 Bit breiten Puffer, mit der Meldung des Zeitüberschreitungsfehlers. Wird der Befehl positiv quittiert, enthält der Datensatz sowohl den Befehl als auch die Antwort des Boards und die FSM geht wieder auf *idle* zurück. Handelt es sich um einen Auslesebefehl, geht die FSM stattdessen in den *hitrx*-Zustand. Der Puffer wird dabei nacheinander mit den Daten jedes empfangenen gültigen Hits beschrieben. Sollte die Eingangsleitung *StoreFR* auf *High* gesetzt sein, werden auch *Forced-Reset*-Hits wie normale Hits behandelt.

Sobald vom *comm*-Prozess ein Datensatz in den Puffer geschrieben wird, wird der zweite Prozess in diesem Modul aktiviert. Der *store*-Prozess läuft mit dem Systemtakt und sorgt für das korrekte Speichern der Datensätze in dem FIFO-Puffer.

Für Antworten auf *RENA*-Befehle wird nur ein Datensatz in den 16 Bit breiten FIFO geschrieben, Hits werden in vier Datensätze geteilt. Anhand des Eingangsports *fifoAFull* wird dem Modul mitgeteilt, dass im FIFO noch Platz für weniger als vier Datensätze zur Verfügung steht. In diesem Fall wartet der *store*-Prozess dass FIFO-Platz frei wird, solange bis ein neuer Hit verfügbar wird. Sollte es nicht möglich gewesen sein, den alten Hit zu speichern, wird er verworfen und eine *Flagge* in dem neuen Hit gesetzt, die auf diese Situation hinweist.

Dieses Modul kann wahlweise von der Kommunikationsschnittstelle oder vom Auslesesequenzer angesteuert werden, sein Verhalten bleibt in jedem Fall identisch. Durch

[1] Finite State Machine

das Setzen des Ausgangs *Busy* auf *High* signalisiert es dem ansteuernden Modul, dass der letzte Befehl noch ausgeführt wird und keine weiteren angenommen werden.

5.2.1 Erweiterung der Zeitstempel

In Abschnitt 2.1.1 wurde erwähnt, dass in dem 16 Bit breiten Zähler für den Zeitstempel in den *RENA Module Boards* alle 2,18 ms ein Rollover stattfindet und dies bei niedrigen Zählraten zu Fehlern in der Messung des Zeitabstands zwischen zwei Ereignissen führen kann.

Dieses Problem wird umgangen, indem die *RENA*-Rollovers gezählt werden, diese Zahl wird dann vor den Zeitstempel gehängt. Das RCI stellt für diesen Zweck 16 solche Zähler, einen für jedes der 16 möglichen Boards, mit einer Breite von 6 Bits zur Verfügung. Der RCI-Zeitstempel weist so eine Breite von 22 Bits auf und die Zeitspanne zwischen zwei RCI-Rollover beträgt 1,398 s.

Ein *RENA*-Rollover wird erkannt, indem für jedes ausgelesene Board der Zeitstempel des vorletzten empfangenen Hits getrennt gespeichert wird und mit dem des letzten Hits des selben Boards verglichen wird. Ist der letzte Zeitstempel kleiner als der vorletzte, d.h. ist ihre Differenz negativ, hat sich zwischen beiden Hits ein Rollover ereignet und der Zähler wird um eins erhöht.

Um zu gewährleisten, dass es sich um maximal einen Rollover handelt, ist es wichtig, die Periode der *Forced-Reset* klein genug gegenüber 2,18 ms zu halten (s. Abschn. 2.1.1).

Solange die Zeitstempelzähler nicht während der Messung zurückgesetzt werden, liefert dieses Verfahren genaue Ergebnisse.

Beim Betrieb der RCIs mit einer externen Synchronisationsquelle kann es aber vorkommen, dass zwischen zwei aufeinanderfolgenden Auslesevorgängen derselben Boards ein Rollover und kurz danach die Synchronisation stattfinden.

Die Hits, die eventuell zwischen dem Rollover und der Synchronisation in den FIFO des Boards geschrieben werden, könnten den Rollover-Zähler einmal zuviel inkrementieren und so die gesamte Messreihe beeinträchtigen.

Im RCI wird dies verhindert indem der Rollover-Zähler beim ersten Auftreten der Negativdifferenz während des ersten Lesevorgangs nach der Synchronisation auf Null gesetzt und jedes weitere Inkrement unterdrückt wird. Alle Hits ab dem auslösenden, bis zum letzten in dem Lesevorgang werden mit einer Warnflagge versehen. Dieser Sachverhalt ist in Abbildung 5.2 veranschaulicht.

Von dieser Unsicherheit bzw. fehlerhaften Angabe des Zeitstempels können in sehr ungünstigen Fällen und bei extrem hohen Zählraten bis zu fünfzehn Hits pro Board bei jedem Synchronisationsvorgang betroffen sein.

Dieser systematische Fehler wäre von einer Auswertungssoftware leicht korrigierbar mittels eines einfachen Algorhytmus[2], das für diese Entwicklungstufe nicht in der Hardware realisiert werden konnte.

5.3 Auslese-Sequenzer

Dieses Modul erlaubt das kontinuierliche, sequentielle Auslesen aller installierten *RENA Module Boards*. Auch dieses Modul ist durch eine FSM realisiert worden.

Bevor es durch Setzen des Eingangs *Run* auf *High* eingeschaltet wird, soll im Eingang *MaxBrd* die höchste ID-Nummer aller am Bus installierten Boards eingestellt sein.

Von der eingestellten ID aus, werden nacheinander Auslesebefehle für alle Boards an die Befehlschnittstelle gesendet. Der 12 Bit breite Befehl wird an den Ausgang *intCmd* gelegt und durch Setzen des Ausgangs *intExec* auf *High* freigegeben.

Vor dem Senden eines Befehls überprüft die FSM in einem Wartezustand, dass der Eingang *intBusy* auf *Low* steht, d.h. dass die Befehlschnittstelle bereit für die Ausführung von Befehlen ist.

Die Auslesesequenz läuft zyklisch rückwärts, von der eingestellten ID bis Null und beginnt erneut bei dieser ID. Der Grund dafür liegt in den Eigenschaften der Vergleichslogik: das Erkennen des Wertes Null in einer Zahlvariablen wird effizienter und mit weniger Hardwareressourcen realisiert, als der direkte Vergleich zweier Zahlvariablen.

Der Zyklus wird durch eine Reihe vorbereitender *RENA*-Befehle (siehe Tabelle 2.2) eingeleitet: zuerst werden die Auslese-Flip-Flops mit MB_WRITE_RUNFF gesetzt, dann werden die *RENA*-FIFOs mit MB_FIFO_CLEAR gelöscht und die Zeitstempelzähler mit MB_TIMER_CLEAR zurückgesetzt.

Bei Rückkehr des Eingangs *Run* auf *Low* wird der Auslesezyklus unterbrochen und die Auslese-Flip-Flops der Boards wieder mit dem Befehl MB_WRITE_RUNFF zurückgesetzt.

5.3.1 Synchronisation der Boards

Durch den *Handshake* mittels der *Busy*-Leitung entstehen praktisch keine Wartezeiten im Auslesebetrieb, die länger als zwei Taktzyklen des 24 MHz *RENA*-Taktes sind.

[2] Es ist ausreichend, beim ersten markierten Hit bis zum dem vor dem zweiten Rollover, den Wert des Rollover-Zähler durch den des letzten unmarkierten Hits plus eins zu ersetzen.

Sollen das RCI und die *RENA Module Boards* durch eine externe Taktquelle, wie z.B. durch das 1 Hz-Signal eines GPS-Empfängers synchronisiert werden, muss aus dem oben genannten Grund der Auslesezyklus rechtzeitig, wie in Abschnitt 4.1 schon erwähnt, angehalten werden.

Der Prozess *GenUTCHold* in diesem Modul überwacht für diesen Zweck den Eingang *UTCSync*.

Wird nach dem Einschalten des Sequenzers eine steigende Flanke (ein Wechsel des Zustands von *Low* auf *High*) an *UTCSync* erkannt, beginnt ein Zähler die Zyklen des Systemtakts zu zählen. Bei der nächsten steigenden Flanke wird der Zähler angehalten.

Der Inhalt dieses Zählers entspricht der Periode des Synchronisationstakts in Systemtaktszyklen. Von dieser Zahl wird die Zahl 345 abgezogen, d.h. die Periode wird um $7,2\,\mu$s verkürzt, also um etwas mehr als die maximale Länge des Auslesevorgangs eines Boards (s. Abschn. 2.2), und in ein Register gespeichert.

Bei jeder weiteren steigenden Flanke von *UTCSync* wird ein Abwertszähler mit dieser Zahl geladen und bei jedem Systemtaktzyklus getriggert. Erreicht dieser Zähler die Null, wird die Auslesesequenz angehalten. Wenn ein neuer Synchronisationspuls erkannt wird, sendet die FSM einen MB_TIMER_CLEAR-Befehl an die Befehlsschnittstelle und startet dann die Auslesesequenz erneut.

Sollte der Synchronisationspuls nicht innerhalb $100\,\mu$s registriert werden können, wird die Sequenz unsynchronisiert wieder gestartet.

Um Störungen oder Verzögerungen des Auslesebetrieb zu vermeiden, werden weitere Synchronisationspulse bis zum Ausschalten des Sequenzer unterdrückt.

5.4 RENA-Konfiguration

Dieses Modul wird nur dann aktiviert, wenn die *RENA*-FPGAs nicht konfiguriert sind oder ihre Konfiguration erneut übertragen werden soll.

Bei der Inbetriebnahme des RCIs und der *RENA Module Boards* befindet sich die Leitung *mbDone* im *Low*-Zustand. Die Konfiguration beginnt, sobald sich im FIFO-Puffer Daten befinden (der Eingang *fifoEmpty* ist auf *Low*).

In einer Schleife wird zuerst ein 16 Bit breiter Datensatz vom FIFO angefordert, der einem Taktzyklus später im Eingang *fifoDOut* verfügbar sein wird. Die *Waveform* mit dem zeitlichen Verhalten der betroffenen Leitungen ist in Abbildung 5.3 dargestellt.

Nach einem weiteren Taktzyklus wird das MSB (in diesem Fall Bit 15) an den Ausgang *mbDIn* gelegt und noch einen Taktzyklus später der Ausgang des Konfigurationstakts *mbCCLk* auf *High* gesetzt. *mbCCLk* bleibt auf *High* für drei Taktzyklen, dann wird er wieder zurückgesetzt.

Nach zwei weiteren Taktzyklen wird das nächste Bit (Bit 14) an *mbDIn* gelegt und die Schleife solange wiederholt, bis auch das LSB übertragen worden ist. Auf diese Weise werden die Daten an das FPGA mit der vom Hersteller empfohlenen Geschwindigkeit von 8 MHz geliefert.

Beim nächsten Durchgang wird wieder vom FIFO ein neuer Datensatz angefordert und der Prozess beginnt von Neuem und geht weiter, bis *mbDone* auf *High* geht und das erfolgreiche Ende der Konfiguration anzeigt.

Sollte der FIFO noch nicht genügend Daten für die Konfiguration empfangen haben und noch während des Prozesses leer geworden sein, geht die Leitung *fifoEmpty* auf *High*. In diesem Fall wird die Schleife angehalten bis neue Daten zur Verfügung stehen.

Ein Konfigurationsfehler wird von den FPGAs über die Leitung *mbInit_n* gemeldet. Erkennt der Prozess ein *Low* auf dieser Leitung, wird die Schleife sofort unterbrochen.

Wenn die Leitung *Enabled* während des normalen Betriebs von der Kommunikationsschnittstelle auf *High* gesetzt wird, löscht ein zusätzlicher Prozess die *RENA*-FPGAs indem die Ausgangsleitung *mbProgram_n* für etwa 500 ms auf L gehalten wird. Danach wird der Konfigurationsprozess erneut gestartet.

5.5 Kommunikationsschnittstelle

Die Kommunikation zwischen dem RCI und dem Auswertecomputer erfolgt über diese Schnittstelle mittels eines entsprechend konfigurierten USB-Chips.

Die Schnittstelle nach Aussen ist für eine asynchrone Kommunikation mit *Handshake* ausgelegt, wie sie üblicherweise bei Mikroprozessoren eingesetzt wird. Diese Art der Übertragung erlaubt zwar nicht den maximalen Datendurchsatz bei gegebener Taktrate, ist aber sehr zuverlässig und ermöglicht die Kommunikation zwischen Geräten mit unterschiedlichen Taktraten oder -phasen. Das eingesetzte USB-Modul von *BrainTechnology* unterstützt, dank der sehr flexibel gestalteten Firmware, unter anderem auch diese Übertragungsart[3].

[3] In der Betriebsanleitung zu der im Abschn. 3.2 erwähnten Windows-DLL wird diese Übertragungsart „Parallel Mode 9“ genannt

Wie aus Abbildung 5.4 erkennbar ist, braucht ein Schreib- bzw. Lesevorgang vier Taktzyklen. Die Daten werden über den 16 Bit breiten bidirektionalen Bus *usbData* transferiert. Das USB kontrolliert die Kommunikation über drei Leitungen: ein Schreib- bzw. Lesevorgang wird mit einem *Low*-Zustand auf der Leitung *usbStrobe* angekündigt, die Transferrichtung über die Leitung *usbDir* bestimmt. Für Schreibzugriffe auf dem RCI wird diese Leitung auf *High* gesetzt, für Lesezugriffe auf *Low*. Die erfolgreiche Übertragung wird vom RCI durch das Setzen der Leitung *usbAck* auf *Low* quittiert. Wird dieses Signal nicht innerhalb einer voreingestellten Zeit (von eines bis einigen Tausend Millisekunden) registriert, erzeugt der Treiber eine von der Software auffangbare Ausnahme.

Der USB-Kontroller schließt den Übertragungvorgang durch das Zurücksetzten von *usbStrobe* auf *High* ab. Ein Taktzyklus später setzt das RCI daraufhin *usbAck* auf *High* zurück.

Wie in Abschnitt 4.1 erwähnt, bestimmt die *RENA*-Leitung *mbDone* über das Ziel von Schreibzugriffen bzw. die Datenquelle bei Lesezugriffen.

Während es bei nicht betriebsbereiten *RENA Module Boards* nur ein Ziel (den FIFO-Puffer) und eine Quelle (das CSR) gibt, ist die Arbeitsweise dieser Schnittstelle beim normalen Betrieb etwas komplexer.

Ein Schreibzugriff wird im normalen Betrieb als Befehl angenommen und interpretiert. Wenn der Sequenzer nicht in Betrieb ist, werden *RENA*-Befehle sofort quittiert und an die Befehlsschnittstelle weitergeleitet. Bei laufendem Sequenzer werden sie abgewiesen (nicht quittiert), so dass die Software eine Fehlermeldung des Treibers empfangen kann.

Steuerungsbefehle (s. Tab. 4.1) für die einzelnen Module bewirken das Setzen der betreffenden Leitungen und werden sofort quittiert.

Die zwei z.Z. definierten RCI_GET-Befehle bewirken das Speichern der angeforderten Information in ein dafür vorgesehenes Zwischenregister und das Setzen eines Gültigkeits-Bit.

Ist dieses Bit gesetzt, wird bei einem Lesezugriff der Inhalt des Zwischenregisters übertragen und das Bit zurückgesetzt. Bei den darauffolgenden Lesezugriffen werden Datensätze aus dem FIFO geholt und übertragen.

Bei leerem FIFO wird, anstatt den Lesezugriff nicht zu quittieren und eine Ausnahme in der Software auszulösen, einfach ein Null-Paket übertragen.

5.6 Testumgebung

Alle Module wurden mit Hilfe der HDL-Simulationssoftware *ModelSim* von *Mentor Graphics* entwickelt und als Simulation auf ihre Funktionalität getestet. Je nach Modul wurde dabei unterschiedlich verfahren.

Für den Test, der als erstes Modul entstandenen *RENA*-Befehlschnittstelle, wurde eigens ein Simulator des *RENA Module Boards* entwickelt.

An einem an das *RENA System Interface* angeschlossenen Board konnte mit dem Logic Analyzer das Zeitverhalten der Busleitungen bei einigen Labormessungen untersucht werden. Die Ergebnisse dieser Untersuchung wurden bei der Entwicklung des Simulatormoduls genutzt, um das Verhalten des Boards möglichst realistisch für den Test der RCI-Module nachbilden zu können. Während der Messungen wurden vom *RENA*-System Dateien erstellt mit Datensätzen sowohl von echten Detektorereignissen als auch von Testpulsen und *Forced-Reset*-Hits.

Diese Dateien wurden in ein Format konvertiert, das in VHDL-Routinen gelesen werden kann und bilden die Ausgangsbasis für die Simulation.

Das Top-Level-Modul, das ursprünglich nur die Befehlschnittstelle enthielt, wurde in einem übergeordneten *Testbench*-Modul zusammen mit dem *RENA*-Simulator instanziert und damit verbunden.

Als die Simulation zufriedenstellende Ergebnisse brachte, wurden nach und nach alle anderen Module mit in den Top-Level integriert und getestet.

5.6.1 Leistung der Datenübertragung

Das Kommunikationsmodul und der FIFO-Puffer wurden bisher als einzige Komponenten auch synthetisiert und in einer realen Arbeitsumgebung getestet.

Dieser Test war schon zu einem relativ frühen Zeitpunkt notwendig, um durch die Untersuchung des Verhaltens des USB-Moduls die beste Modalität für die Kommunikation mit dem RCI zu bestimmen.

Anhand der Ergebnisse dieser Untersuchung wurde die endgültige Architektur des Kommunikationsmoduls festgelegt und eine erste Abschätzung der zu erwartenden Datendurchsatzrate ermöglicht.

Mit dem Logic Analyzer wurde beobachtet, dass das USB-Modul alle zehn Taktzyklen einen Lesezugriff startet. Nach 256 aufeinanderfolgenden Zugriffen, also nach fast $53,3\,\mu$s hält die Übertragung für $28\,\mu$s an, dann wird sie für weitere 256 Zugriffe fortgesetzt.

Da bei jedem Lesezugriff zwei Byte übertragen werden, bedeutet dies der Transfer von einem 512 Byte großen Paket alle $81,3\,\mu$s, was einem Durchsatz von ca. 6 MByte/s oder 750.000 Hit/s entspricht.

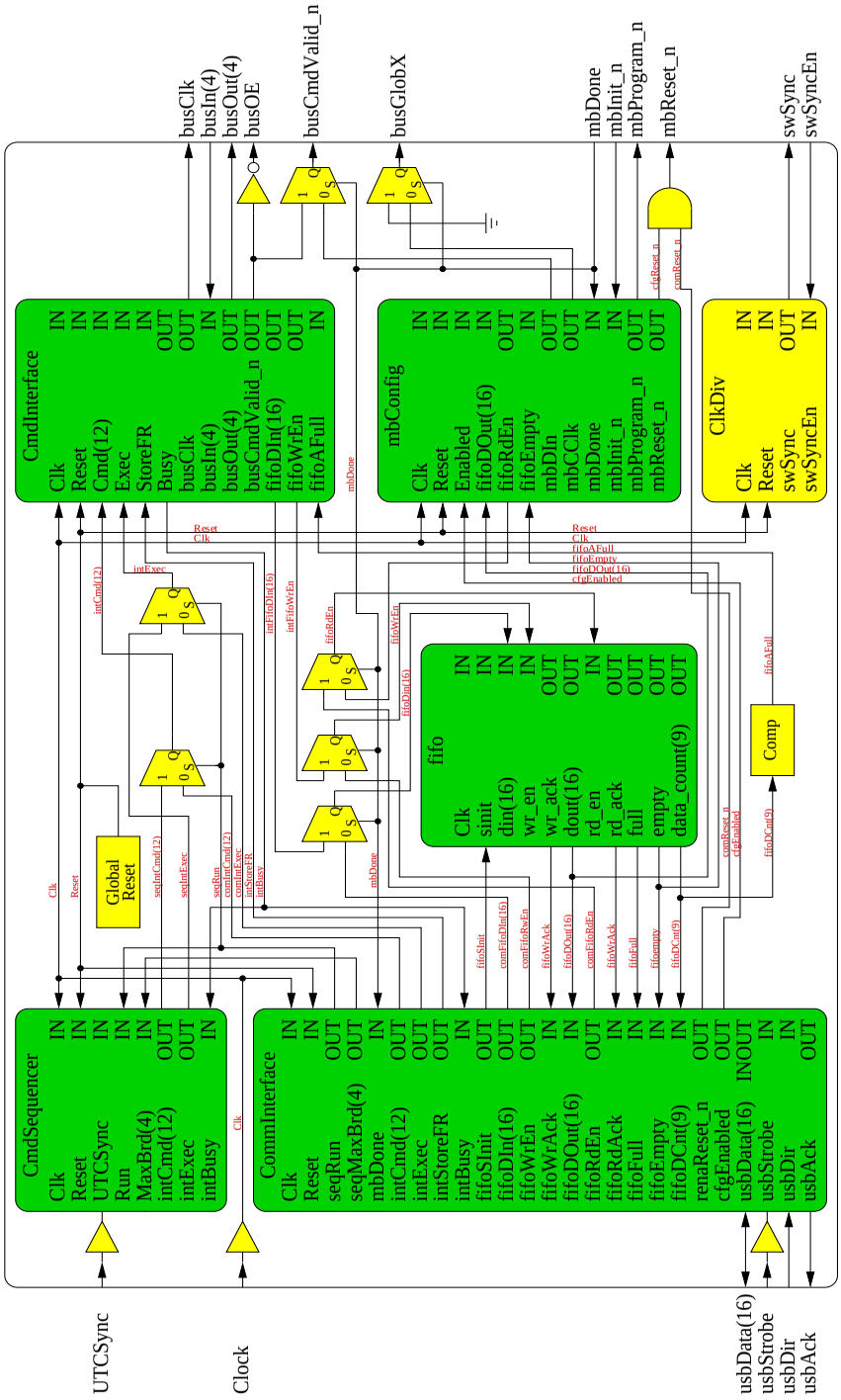

Abbildung 5.1: Top-Level des RCI-Designs

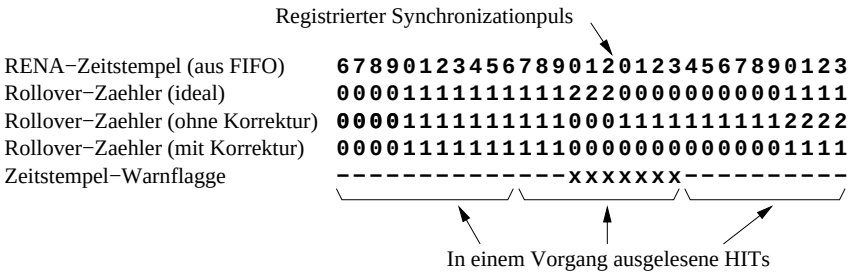

Abbildung 5.2: Entstehung und Markierung problematischer Zeitstempel

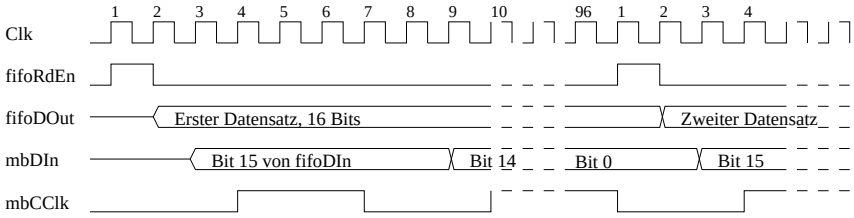

Abbildung 5.3: *Waveform* der Konfigurationsleitungen

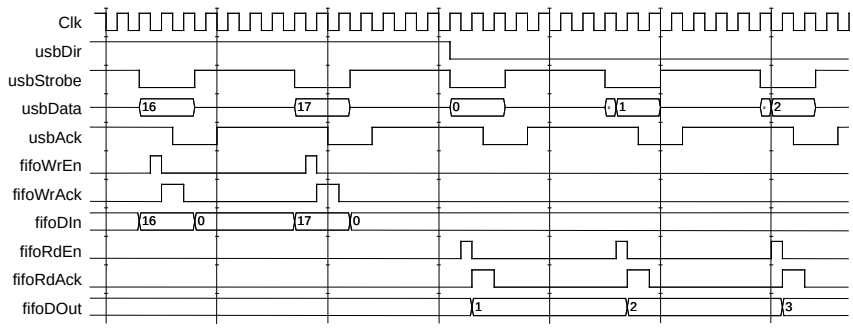

Abbildung 5.4: *Waveform* der Kommunikationsleitungen

Kapitel 6

Ausblick

Die Entwicklungsarbeit am *RENA Control Interface* ist, wie aus Abschnitt 5.6 ersichtlich wird, noch nicht vollständig abgeschlossen.

Die Simulationstests haben zwar zufrieden stellende Zwischenergebnisse gebracht, es hat sich jedoch auch gezeigt, dass dennoch Potential für Optimierungen vorhanden ist.

Vor allem soll noch die Stabilität und die Ausfallsicherheit des Designs in sehr unterschiedlichen Situationen, die während des Betriebs auftreten könnten, in einem noch vertretbaren Umfang untersucht werden.

Weitere, nicht zu unterschätzende Hürden wird die Synthese des Gesamtdesigns darstellen. Die Ergebnisse einiger erfolgreich abgeschlossener Tests geben allerdings Grund zur Zuversicht, in kurzer Zeit ein voll einsatzfähiges Gerät präsentieren zu können.

Die ersten Tests der Übertragungsleistung, die im Abschnitt 5.6.1 geschildert werden, zeigen zwar, dass die angestrebte Rate von $2,1 \cdot 10^6$ Hit/s nur zu etwa 36% erreicht wird, geben aber dennoch keinen Grund zur Sorge. Einerseits sind 750.000 Hit/s schon das 15-fache dessen, was das bisherige System leistet. Anderseits hat der Test auch gezeigt, dass dieser Wert nicht durch das Design selbst, sondern durch das derzeit eingesetzte USB-Modul bzw. dessen Software bedingt ist.

Die Kommunikationsschnittstelle schließt einen Lesevorgang des USBs - im Idealfall gleicher Taktfrequenz und Phase beider Komponenten - innerhalb vier Taktzyklen ab. Sollte ein USB-Modul in der Lage sein, nach nur einem Taktzyklus Latenz einen weiteren Lesevorgang zu starten und kontinuierlich arbeiten zu können, d.h. ohne die 28 μs Pause zwischen zwei 512 Byte großen Paketen, würde das RCI eine Übertragungsleistung von 18,31 MByte/s erreichen. Das entspricht, für die 8 Byte großen RCI-Hits, $2,4 \cdot 10^6$ Hit/s.

Die nächste Aufgabe liegt in der Auswahl eines noch leistungsfähigeren USB-Moduls und effizienterer Firmware und Treiber. Eventuell würde es auch reichen nur Firmware und Treiber für das bestehende Modul zu optimieren. Weitere Untersuchungen können das zeigen.

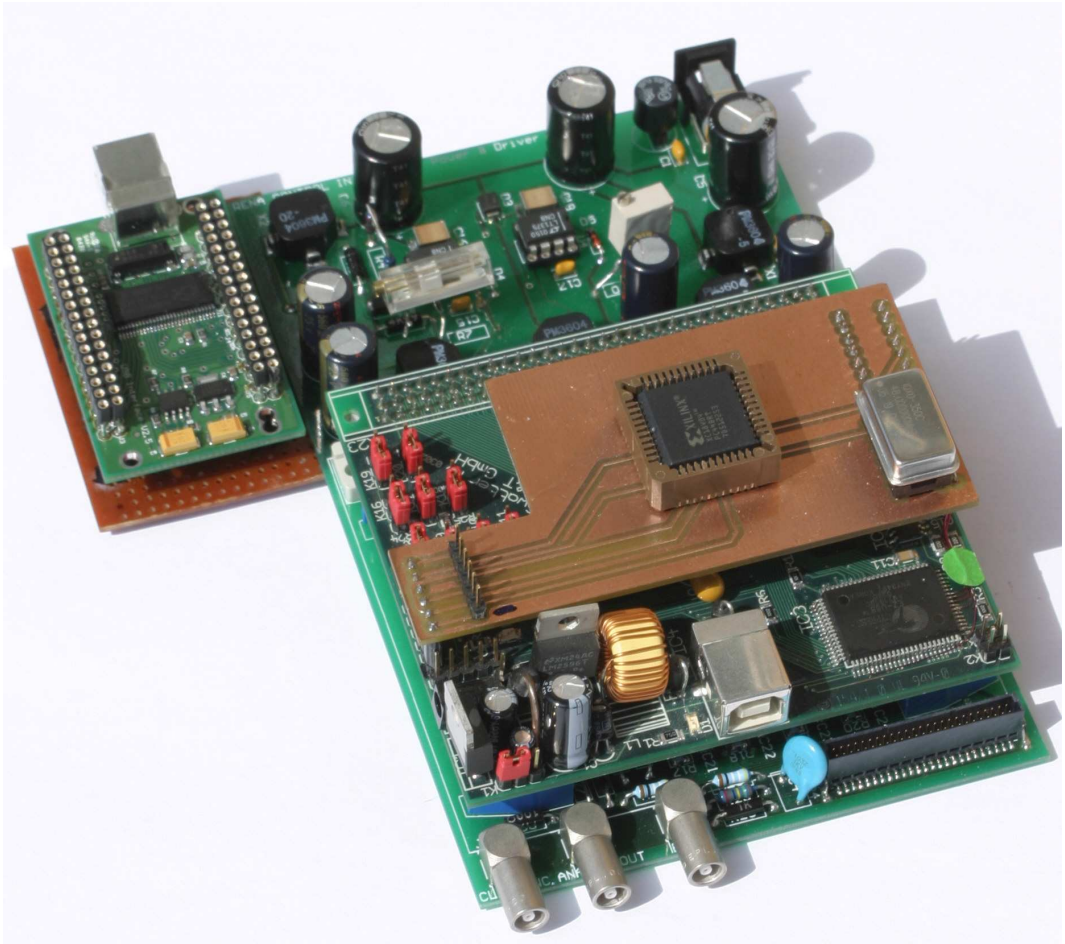

Abbildung 6.1: Das komplett aufgebauter Prototyp des *RENA Control Interface*

Auch muss noch überprüft werden, ob nicht am Layout der Platine oder der Dimensionierung und Effizienz der Spannungsversorgungskomponenten etwas verändert werden soll.

Nicht zuletzt ist die Programmierung eines kompletten Softwarepakets für den Betrieb des RCIs und die Datenauswertung notwendig.

Es gibt also noch viel zu tun...

Anhang A

Field-Programmable Gate Array

Ein FPGA ist ein elektronisches Bauteil, welches aus einer Matrix frei programmierbarer Gatter[1] besteht. Diese speziellen Gatter werden *Configurable Logic Block* (CLB) genannt und bestehen ihrerseits aus einer Nachschlagetabelle (LUT[2]) in der die kombinatorische Logik gespeichert wird, einem *D-Latch*-Flip-Flop, der das Ausgangssignal der LUT mit dem Taktsignal synchronisiert und einem Multiplexer (MUX) der entweder das synchrone oder asynchrone Ergebnis zum Ausgang des Gatters führt (s. Abb. A.1).

Ein CLB kann auch als Schieberegister oder als einzelne, unabhängige Speicherzelle (Distributed RAM) genutzt werden. Je nach Typ und Komplexität kann ein FPGA einige Hundert bis mehreren Millionen CLBs beinhalten, oft werden auch zusätzliche Spezialkomponenten wie zusammenhängende Speicherblöcke (Block RAM), Addierer, Multiplizierer, Digital Clock Managers (DCM[3]), PLLs, oder arithmetische Einheiten (ALU[4]) bis hin zu kompletten Prozessoren (wie der PowerPC-Prozessor im *Xilinx Virtex-II PRO* und *Virtex-4*) mit aufs FPGA integriert.

Die CLBs, alle anderen Komponenten und die Gehäuseanschlüsse (je nach Gerät, von wenigen Dutzend bis über tausend) sind untereinander durch ein sehr dichtes Gitter von programmierbaren Verbindungen vernetzt und durch mindestens einen dedizierten Taktverteilerbaum (Clock Fanout Network) versorgt, um störende Laufzeitunterschiede des Taktsignals zu vermeiden. Die Struktur dieses Verbindungsgitters kann bis zu 90% der Chipfläche belegen.

Dank dieser Architektur ist es möglich, durch Speicherung von Konfigurations-Bits in LUTs, Multiplexern, Verbindungsknoten und in den Registern der Zusatzkomponenten eine große Vielfalt an digitalen Funktionen zu realisieren.

[1] Digitale logische Basiselemente. Sie führen die elementaren Operationen UND, ODER und NICHT aus.

[2] Look-Up Table

[3] Eine dedizierte Komponente für die Aufarbeitung von Taktsignalen. Ein DCM ermöglicht den *Clock Deskew*, d.h. die Reduktion der Abweichung von der idealen Rechteckform des Signals, und kann Signale mit verschiedenen Phasen liefern. Die DCMs der *Virtex-II* stellen auch einen Frequenzverdoppler und einen einstellbaren Frequenzteiler zur Verfügung.

[4] Arithmetic Logic Unit.

Ein anderes Bauteil dieser Art, in mancher Hinsicht einem FPGA ähnlich, ist das *Complex Programmable Logic Device* (CPLD). Es wird mit einer anderen Technologie gefertigt und kann dadurch seine Konfiguration nach dem Ausschalten beibehalten, ist aber wesentlich weniger leistungsfähig und vielseitig als ein FPGA. Ein CPLD kann nur einige Hundertmal konfiguriert werden und nur wenige Hunderttausende CLBs und keine RAM Blöcke enthalten. Einige auf dem Markt befindliche CPLDs integrieren deswegen im selben Gehäuse einen zusätzlichen RAM-Chip.

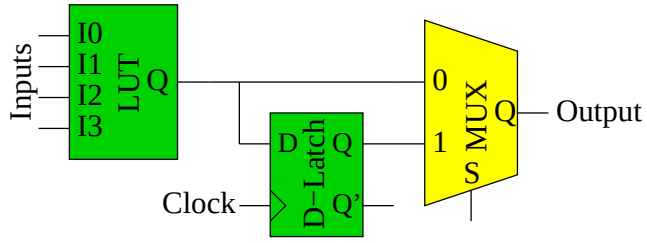

Abbildung A.1: Configurable Logic Block

A.1 Hardwaremodellierung

Um das benötigte Konfigurations-Bitmuster zu erstellen werden die gewünschten Funktionen und das Verhalten der Zielanwendung in einem *Modell* beschrieben, das in einer Hardware-Beschreibungssprache (HDL[5]) erstellt wird.

Eine HDL ähnelt einer Programmiersprache, aber ihre Struktur und Grundelemente sind nicht sequentielle Anweisungen für einen Prozessor, sondern spiegeln die Strukturen einer elektronischen logischen Schaltung[6] wieder.

In einem HDL-Modell entsprechen *Signale* den Variablen einer Programmiersprache und *Prozesse* nehmen die Rolle von Funktionen und Prozeduren ein, alle Prozesse laufen aber parallel, d.h. gleichzeitig ab und Signale ändern ihren Zustand entweder sofort, falls sie eine Verbindung beschreiben, oder nach Abschluß des Prozesses in dem ihnen ein neuer Zustand zugewiesen wird, und dienen in diesem Fall als Ein- bzw. Ausgangspuffer.

[5] Hardware Definition Language. HDL-Modelle werden in der Halbleiterindustrie auch für die Herstellung von ASICs eingesetzt.

[6] Diese Ähnlichkeit führt oft dazu, für das Entwickeln von Hardware mit einer HDL, etwas unpassend von „Programmieren“ zu sprechen. Die Beschreibung von Hardware wird zutreffender „Modell“ (Lat. *modulus* = Maß) oder mit dem Anglizismus „Design“ bezeichnet, da gerade das sequentielle, fortschreitende „Wesen“ eines Programms (Griech. *pro* = vor-, voranschreitend und *grámma* = Schrift, Gesetz) der massiv-parallelen Architektur eines FPGA nicht gerecht wird.

Während in den USA, unter einer Vielfalt verfügbarer HDLs, *Verilog* bevorzugt wird, hat in Europa *VHDL* die größere Verbreitung. Da VHDL auch am IAAT eingesetzt wird [Sch01], ist das Modell für dieser Arbeit mit dieser HDL realisiert.

VHDL erlaubt ein modulares Design, d.h. ein komplexes Modell kann in einzelne Komponenten oder Module zerlegt werden, die dann innerhalb eines übergeordneten Moduls instanziert und miteinander verknüpft werden.

Die oberste Ebene des Modells ist das so genannten Top-Level-Modul, in dessen Schnittstelle die Ports definiert werden, die direkt den Ein- und Ausgabepins des FPGA zugewiesen werden.

Obwohl die Aufteilung der Aufgaben in Module bis auf die Gatter-Ebene fortgesetzt werden kann, ist es sinnvoll, das Design in einzelne, überschaubare Module zu gliedern, die eine abgeschlossene Funktionalität erreichen und einzeln entwickelt und getestet werden können.

Jedes Modul oder das komplette Modell ist in zwei Hauptbestandteile unterteilt: in dem *Entity* genannten Teil wird die Schnittstelle der modellierten Komponente definiert. Im Teil *Architecture* wird die Funktionalität des Moduls sowohl *struktural*, d.h. als Verdrahtung elementarer oder komplexeren Bausteine, als auch *funktional* durch Operatoren und Sprachkonstrukte zur Ablaufsteuerung beschrieben.

Effiziente Arbeitsumgebungen, wie das am IAAT eingesetzte HDL-Software-Paket *FPGA Advantage* von *Mentor Graphics*, erlauben, neben der Entwicklung des Modells, auch die *Simulation* seines Verhaltens. Dies beinhaltet die Visualisierung von Signalzuständen und der Inhalte von Speichern und Registern. Die Simulation kann sowohl unter Idealbedingungen als auch unter Berücksichtigung von Signallaufzeiten mit dem so genannten *Back Annotation* Verfahren stattfinden. Für die Back Annotation werden Informationen in die Simulation gespeist, die aus der zweiten Stufe der Entwicklung, der *Synthese*, gewonnen werden (siehe Abbildung A.2).

Die Synthese ist ein von der eingesetzten Technologie abhängiger Prozess, d.h. Hersteller, Typ, Version, Gehäusetyp und Geschwindigkeitsklasse des Zielbausteins müssen der Synthesesoftware exakt bekannt sein. Es wird dabei eine *Verbindungs-* oder *Netzliste* erstellt, die das entworfene HDL-Modell in die physikalische Struktur des FPGAs bzw. CPLDs abbildet. Dieser Prozess durchläuft u.U. mehrere Optimierungsstufen, und wird durch einen weiteren Prozess namens *Place and Route* ergänzt, in dem Randbedingungen wie Pinzuweisung, minimale Taktrate und maximale Signallaufzeiten bestimmter Verbindungen berücksichtigt werden. An dieser Stelle werden die Daten für die Back Annotation berechnet.

Am Ende der Synthese wird eine Datei erzeugt, in deren Bitmuster die Konfiguration von CLBs, des Verbindungsgitters und aller anderen Komponenten des Zielbausteins enthalten ist. Dieses Bitmuster wird seriell entweder über eine dedizierte Schnittstelle oder mittels *JTAG* an den Baustein übertragen.

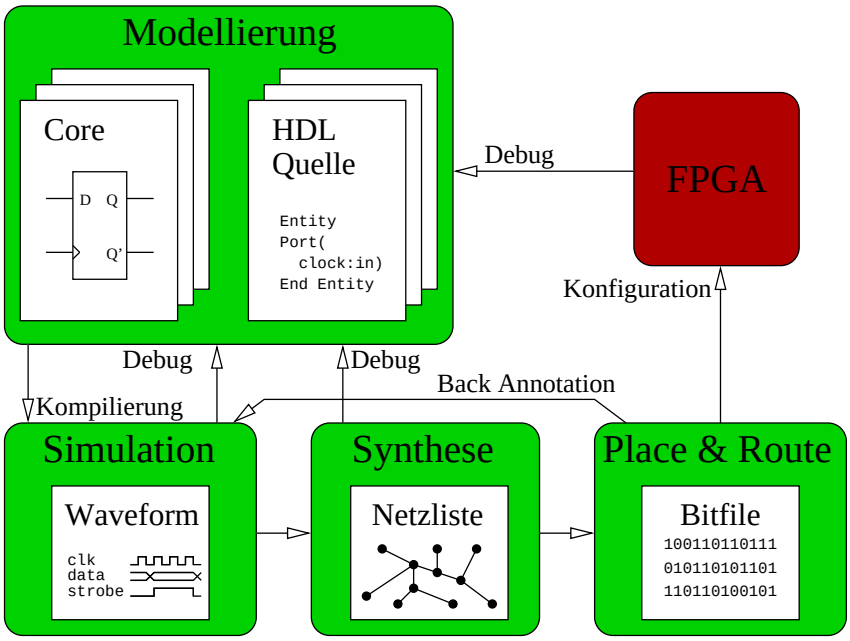

Abbildung A.2: HDL-Modellierung: Ablaufschema

Hersteller von FPGAs, wie *XILINX* mit dem Paket *ISE*, stellen oft dem Entwickler Bibliotheken mit einer Reihe von vorgefertigten Lösungen (*Cores*) zur Verfügung, durch die eine optimale Nutzung ihrer Bausteine erleichtert wird. Es handelt sich dabei sowohl um Beispiele zur Einbindung und Konfiguration von RAM-Blöcken, DCMs, Ein- und Ausgangspuffer oder Taktverteilungsbäume als auch komplette Implementierungen von, je nach Leistung des Zielbausteins, relativ komplexen Elementen wie Ring- bzw. FIFO-Speicherpuffern, Zählern, Addierern, Multiplizierern, seriellen und parallelen Kommunikationsschnittstellen bis hin zu Mikrocontrollern. Je nach Komplexität können Cores entweder kostenlos oder gebührenpflichtig in eigene Entwürfe eingebunden werden.

Anhang B

Universal Serial Bus

Der *Universal Serial Bus* hat sich seit seiner Einführung Mitte der 1990er Jahre als eine praktische, schnelle und sehr zuverlässige serielle Schnittstelle für PC und Workstation-Computer am Markt behauptet.

Die USB-Spezifikation [Com00] wurde von einem Konsortium führender Hard- und Softwarehersteller ausgearbeitet und liegt seit dem Jahr 2000 in der Version 2.0 vor. Auch Geräte die entsprechend den Richtlinien der Vorgängerversion 1.1 gebaut werden sind immer noch sehr verbreitet und werden wegen der niedrigeren Kosten für Anwendungen eingesetzt, welche keine sehr hohe Datenraten verlangen.

Die Version 1.1 des USB-Standards unterstützt zwei Übertragungsgeschwindigkeiten: 1,5 MBit/s und 12 MBit/s, entsprechend als *Low-Speed* und *Full-Speed* bezeichnet. Die Version 2.0 unterstützt zusätzlich die als *High-Speed* bezeichnete Geschwindigkeit von annähernd 480 MBit/s[1].

B.1 Kommunikationsmodell

Die Kommunikation über die serielle Verbindung zwischen dem *Host*, d.h. dem USB-Kontroller im Rechner, und den USB-Geräten (in der Literatur oft *Function* oder *Device* genannt) wird über spezielle, *Pipe* genannte, virtuelle Kanäle durchgeführt.

Pro Übertragungsrichtung[2] gibt es 15 unabhängige Pipes die, von 1 bis 15 nummeriert, entweder vom Typ „IN“ oder „OUT“ sind. Eine weitere Pipe, die einzige vom Typ „INOUT“ mit der Nummer 0 wird nur für die Initialisierung und die Kontrolle des Geräts benutzt. Jede Pipe endet in einem *Endpoint*, physikalisch handelt es sich

[1] Bei diesen Bitraten handelt sich um „Bruttodaten“, nach Abzug von Kontrollsequenzen, Prüfsummen und anderem „Overhead“ reduziert sich der maximale Nutzdatendurchsatz z.B. bei High-Speed auf 384 MBit/s

[2] Die USB-Terminologie definiert die Richtungen streng nach der Hierarchie des Systems: „oben“ ist die Anwendersoftware, dann das Betriebssystem mit den Treibern und dem USB-Host-Kontroller im Computer, ganz „unten“ sind die USB-Geräte. Man spricht auch von *Downstream* in Richtung der Geräte, oder *Upstream* von den Geräte zu dem Computer. „OUT“-Daten gehen von der Software zu den Geräten, „IN“-Daten werden von den Geräten angefordert.

hierbei um einen FIFO-Puffer von festgelegter Tiefe für die empfangenen bzw. zu sendenden Daten. Jedes Gerät unterstützt mindestens den Endpoint 0, kann aber durch die Implementierung mehrerer Endpoints verschiedene Funktionen oder Modi bereitstellen.

Daten und Befehle werden in Form von Paketen übertragen. Es entsteht zwischen dem Host und den adressierten Geräten eine Art Dialog, bei dem immer sichergestellt wird, dass die gesendeten Daten korrekt empfangen und interpretiert werden.

Abbildung B.1 zeigt drei Pakete, die zu einem Datentransfer gehören. Bei dem ersten Paket in diesem Beispiel handelt es sich um ein *Token* (ein spezielles Paket das Befehle oder Kontrollsequenzen transportiert, das nur durch den Host gesendet wird), in diesem Fall ein „OUT“-Token, das einen Transfer von Daten zu einem Gerät ankündigt.

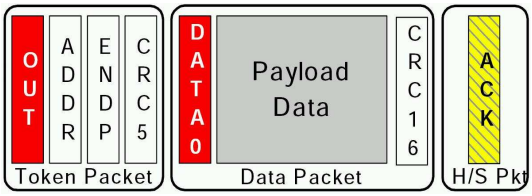

Abbildung B.1: Typische USB-Pakete, hier ein *Bulk-Transfer* vom Host zu einem Gerät [Cyp04].

Das erste Feld in diesem Token enthält den *Packet-Identifier* (*PID*). Das ist eine 8 Bit breite Zahl welche Zweck und Inhalt des Pakets angibt. Anstatt seines numerischen Wertes, wird in der Literatur nur sein Akronym (meistens *Mnemonic* genannt) angegeben, siehe Tabelle B.1, welche alle definierten PIDs auflistet.

Dem PID folgt die Adresse und die Pipe-Nummer des Ziels sowie die 5 Bit breite *CRC*-Prüfsumme[3] anhand derer die korrekte Übertragung vom Empfänger festgestellt werden kann.

Das zweite Paket transportiert die Nutzdaten und die 15 Bit breite Prüfsumme vom Host zum Gerät, das dann mit dem *Handshake*-Paket (das dritte im Bild) antwortet. In diesem Fall zeigt die Antwort „ACK“ (Acknowledge), dass die CRC-Prüfung der Daten positiv verlaufen ist.

Außer dem Handshake-PID „ACK“ und seiner Negation „NAK“ sei hier das Token „SOF“ besonders erwähnt. Dieses *Start of Frame* Paket wird am Anfang eines 1 ms langen Zeitfensters vom Host unadressiert gesendet, d.h. jedes Gerät empfängt es und kann es für die interne Zeitverwaltung nutzen. Alle Frames werden mit einer

[3] engl. Cyclic Redundancy Check (zyklische Redundanzprüfung). Ein Verfahren zur Erkennung von Fehlern bei der Übertragung von Daten, das auf Vergleich mit einer Quersumme basiert.

Tabelle B.1: USB PIDs [Cyp04]

PID Type	PID Name
Token	IN, OUT, SOF, SETUP
Data	DATA0, DATA1, **DATA2**, **MDATA**
Handshake	ACK, NAK, STALL, **NYET**
Special	PRE, **ERR**, **SPLIT**, **PING**

Bold type indicates PIDs introduced with USB 2.0

11 Bit breiten Zahl durchnummeriert, die zusammen mit dem PID und der CRC-Prüfsumme den SOF-Token bilden.

Im High-Speed-Modus ist jedes Frame in weitere 8, je 125 μs lange *Microframes* unterteilt. Auch Microframes werden von 0 bis 7 durchnummeriert, wobei die Breite der Frame-Nummer auf 8 Bit reduziert wird.

Da eine detaillierte Erläuterung aller anderen PIDs den Rahmen dieser Arbeit bei weitem sprengen würde, sei hier auf die Quelle dieser Tabelle, [Cyp04], sowie auf [Com00] und [Kel03] verwiesen.

B.1.1 Transferarten

Die USB-Spezifikation [Com00] definiert vier verschiedene Transferarten:

- **Control** wird nur zur Übertragung von Konfigurationsdaten und Kontrollbefehlen benutzt. Der Transfer kann nur auf dem Endpoint 0 stattfinden und erlaubt als einzige Transferart die bidirektionale Kommunikation.
- **Interrupt** dient zur Übertragung kleinerer Datenmengen in regelmäßigen Zeitabständen von 1 ms oder einem Vielfachen davon. Sie wird meistens zur Statusabfrage oder zum Einlesen von Sensoren oder Benutzerschnittstellen (HID, Human Interface Device), z.B. Maus oder Tastatur eingesetzt, meistens in „IN“-Richtung.

  Obwohl durch diesen Namen die Analogie zum Interrupt-Mechanismus von Prozessoren nahe liegend erscheint, handelt es sich in der Tat um ein „Polling“. Ein USB-Gerät darf nicht einen Transfer initiieren und damit den Host auf sich „aufmerksam machen“.
- **Bulk** eignet sich für die Übertragung größerer Datenmengen, bei denen es nicht auf den genauen Zeitpunkt der Übertragung ankommt. Dies ist die Übertragungsart, die den größten Datendurchsatz erlaubt. Bulk ist nicht für Low-

Speed-Geräte spezifiziert. High-Speed-Transfers dieser Art erfolgen in 512 Byte großen Paketen.

- **Isochronous** ist für zeitkritische Daten reserviert wie z.B. Audio- oder Video-Datenströme. Diese Übertragungsart ist die einzige, die ohne Fehlererkennung und ohne Handshake durchgeführt wird. Sie garantiert aber den Transfer von 1 bis 1023 Byte großeen Datenblöcken innerhalb eines Frames. Im High-Speed-Isochronous-Transfer werden bis zu drei Pakete, je 1024 Byte groß, innerhalb eines Microframes übertragen. Für Isochronous- und Interrupt-Transfers zusammen werden bis zu 90% der Bandbreite reserviert.

B.2 Topologie

Bis zu 127 Geräte können an einem Bus angeschlossen werden. Die physikalische Topologie vom Bus ist die eines Baums, wobei die Knoten von speziellen Geräten, genannt *Hub*, gebildet werden. Ursprung der Struktur ist der *Host* von dem aus alle Datentransfers gestartet und kontrolliert werden. Diese Betriebsart wird auch *Single Master* genannt.

Eine direkte Kommunikation zwischen einzelnen Geräten am Bus ist nicht möglich, genauso wenig wie Anfragen oder Transfers, die von den Geräten gestartet werden.

Logisch entspricht die USB Topologie der eines Sterns (Abb.: B.2, rechts), mit dem Host in der Mitte.

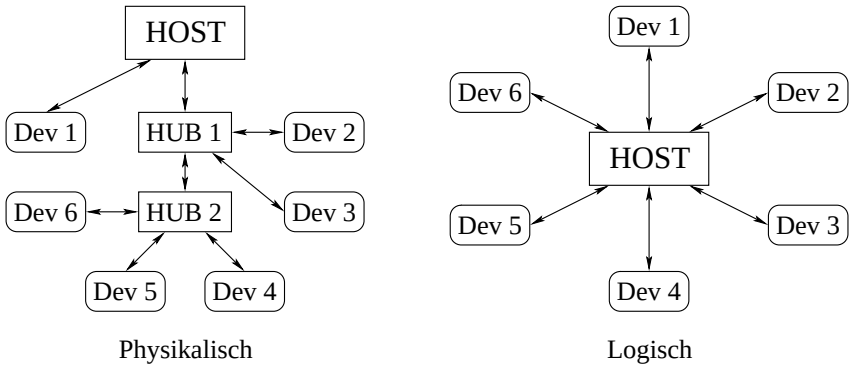

Abbildung B.2: Physikalische und logische Topologie [Kel03]

Die Verbindung zwischen dem Host bzw. Hub und den Geräten ist eine differentielle Punkt-zu-Punkt-Verbindung über ein verdrilltes und, bei USB 2.0 auch abgeschirmten Adernpaar (Adern $D+$ und $D-$ in Abb. B.3). Das Kabel beinhaltet auch eine Leitung mit $V_{BUS} = +5\,\mathrm{V}$ und eine Masseleitung, über die angeschlossene Geräte

mit Strom von bis zu 500 mA versorgt werden können. Das Kabel darf maximal 5 m lang sein.

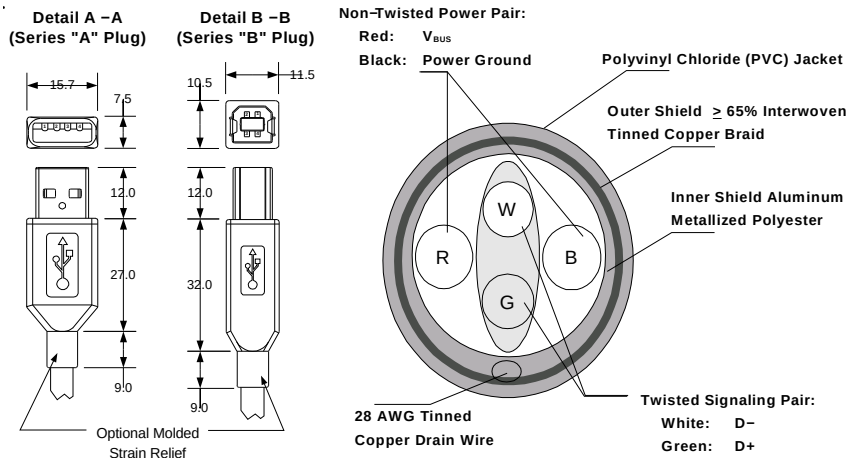

Abbildung B.3: Stecker der Serie „A“ und „B“, Kabelquerschnitt [Com00]

Einheitliche Steckertypen, unterschiedlich auf dem Host bzw. Hub in Richtung der Geräte (*Downstream*, Serie „A“) und auf den Geräten bzw. Hub in Richtung Host (*Upstream*, Serie „B“) verhindern Umpolung und Fehler beim Anschließen der verschiedenen Geräte. Je nach Gerätetyp und Hersteller kann es verschiedene Ausführungen von miniaturisierten Steckern der Serie „B“ geben.

B.3 Funktionsprinzip

USB-Geräte können jederzeit, beim laufenden Betrieb des Systems, dem Bus angeschlossen oder entfernt werden (*Hot Plug and Play* Fähigkeit). Der Host erkennt, dass ein neues Gerät auf dem Bus angesteckt oder eingeschaltet worden ist und kann es nach seinen spezifischen Eigenschaften und Merkmalen, so genannten *Deskriptoren* abfragen.

Jedes neu angeschlossene Gerät (inklusive eventueller Hubs) ist zuerst unter der Adresse „0“ für den Host erreichbar. Über einen als *Enumeration* bezeichneten Prozess vergibt der Host dem neuen Gerät eine Adresse zwischen 1 und 127 unter der das Gerät für den Rest der Betriebszeit eindeutig identifiziert ist.

Die abgefragte Information wird dann dem Betriebssystem mitgeteilt, welches die passenden Gerätetreiber lädt und das Gerät den Anwendungen zugänglich macht.

Wird das Gerät entfernt oder abgeschaltet, sorgt der Host dafür, dass das Betriebssystem den Gerätetreiber wieder entfernt oder deaktiviert.

B.3.1 Elektrische Buszustände

Um zu gewährleisten, dass der Host das neue Gerät auf dem Bus erkennt, werden spezielle Zustände der Datenleitungen benutzt.

Bei einem differentiellen Bus sind die Zustände *Logisch 1* und *Logisch 0* durch die Spannungsdifferenz zwischen beiden Leitungen definiert. Beim USB sind es: [Kel03]

- Differentielle 1, wenn $V_{D+} - V_{D-} > 200\,\mathrm{mV}$
- Differentielle 0, wenn $V_{D+} - V_{D-} < -200\,\mathrm{mV}$

Sollte am Host oder am Hub kein Gerät angeschlossen sein, halten zwei $15\,\mathrm{k\Omega}$ *Pull-Down* Widerstände beide Leitungen auf niedrigem Pegel. Dieser Zustand (*SE0*-Zustand, *Single Ended Zero*) ist zwar differentiell nicht definiert, wird jedoch erkannt und wird nicht nur dazu benutzt, einen leeren Port zu kennzeichnen. Wird ein Gerät angeschlossen, bringt ein $1,5\,\mathrm{k\Omega}$ *Pull-Up* an seinem Port eine der Leitungen auf hohen Pegel. Bei Low-Speed-Geräten ist es die Leitung $D-$ (entspricht Logisch 0), bei Full-Speed-Geräten die $D+$ (Logisch 1). Dieser ungetriebene[4] Zustand wird J-Zustand oder *Idle* genannt.

Registriert der Host einen Wechsel des Buszustands von *SE0* nach *Idle*, erkennt er dass ein Gerät angeschlossen worden ist (*Connect*-Ereignis) und am logischen Wert des *Idle*, ob es sich um Low-Speed- (*Idle* ist Logisch 0) oder Full-Speed-Gerät (Logisch 1) handelt. High-Speed-Geräte melden sich auch wie Full-Speed-Geräte am Bus an und beginnen die Übertragung ihrer Parameter mit dem Full-Speed-Protokoll. Erst nach der Enumeration ändern sie ihre Arbeitsweise, dabei koppeln sie den $1,5\,\mathrm{k\Omega}$ Pull-Up aus der $D+$-Leitung ab.

Die Umkehrung des J-Zustands heißt K-Zustand oder *Resume*. Treibt ein Sender (Host oder Gerät) den Bus in einen K-Zustand für 2 Taktzyklen, signalisiert er den Beginn der Übertragung eines Datenpakets (SOP, Start-Of-Paket). Am Ende des Pakets wird der *SE0* für 2 Taktzyklen getrieben. Wird der *SE0* vom Host länger als $2,5\,\mu\mathrm{s}$ getrieben, so bedeutet dies für das Gerät einen Reset-Befehl. Vom Host registriert, wird dies als *Disconnect* interpretiert.

Die *Tranceiver* der High-Speed-Geräte sind aufgrund der wesentlich höheren Bitrate anders gebaut und können durch die unterschiedliche Busterminierung und den fehlenden Pull-Up den SE0-Zustand beim *Disconnect* nicht auslösen. Da es sich

[4] In der Elektronik wird als *Treiber* eine Komponente bezeichnet, die eine Leitung aktiv mit Spannung oder Strom ansteuert. Elektrische Leitungen können durch *Pull-Up* bzw. *Pull-Down* Widerstände entsprechend auf einem bestimmten Potential oder auf Masse gehalten werden. Ein aktiver Treiber arbeitet gegen diese Widerstände und erzwingt auf der Leitung einen anderen logischen Zustand. Er „treibt“ den Zustand. Wird der Treiber deaktiviert, kommt die Leitung in den Ruhezustand zurück. Nicht zu verwechseln mit dem in der Fußnote 2 auf Seite 53 oder im Abschnitt B.3 genannten Gerätetreiber, bei dem sich um eine Softwarekomponente handelt, die dem Betriebssystem und der Software den Zugriff auf die Hardware ermöglichen.

beim Sender um eine Stromquelle handelt, erkennt der Host die fehlende Ohmsche Last der Terminierung am Bus-Ende indem er einen J- oder K-Zustand treibt und dabei eine höhere Spannung (800 mV statt 400 mV) an den Leitungen feststellt [Kel03; Com00].

B.3.2 Datenkodierung

Die eigentliche Übertragung der Daten auf dem seriellen USB erfolgt im so genannten *Non Return to Zero Inverted* (NRZI)Verfahren.

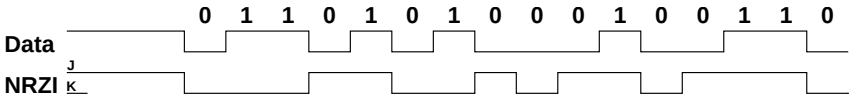

Abbildung B.4: NRZI-Kodierung serieller Daten [Com00]

Jede 0 im Datenstrom entspricht einem Zustandswechsel des Busses, eine 1 belässt den Bus in seinem letzten Zustand. Aus einer Folge von Nullen, also einem wiederholtem Wechsel kann der Empfänger mit Hilfe von PLL-Schaltungen[5] den Bustakt zurückgewinnen. Das geschieht z.B. am Anfang jedes Pakets: das so genannte *SYNC*-Feld ist eine Folge von 7 Null-Bits bei Low- und Full-Speed oder 31 Null-Bits bei High-Speed gefolgt von einem Eins-Bit.

Damit eine lange Folge von Einser nicht einen zu langen Stillstand im Bus verursacht, was den möglichen Verlust der Taktsynchronisation bedeuten würde, wird nach sechs aufeinander folgenden Einsern eine Null hinzugefügt (*Bit Stuffing*).

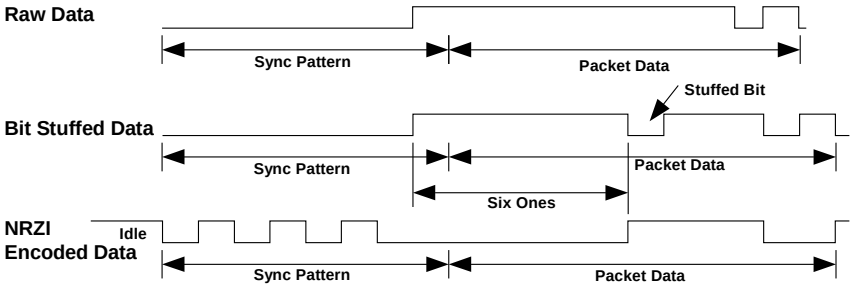

Abbildung B.5: Bit-Stuffing [Com00]

Der Empfänger dekodiert zuerst aus dem NRZI-Strom eine Bitfolge, anschließend sorgt er für das *Destuffing*, indem er die hinzugefügten Nullen im Bitstrom erkennt und eliminiert.

[5] *Phase-locked loop*. Ein phasengekoppelter Regelkreis, der u.a. für die Taktsynchronisation verwendet wird.

B.4 Implementierung von Cypress

Cypress ist einer der weltweit größten Hersteller von USB-Komponenten. Die Chipfamilie *EZ-USB* (sprich: „Easy-USB“) eignet sich durch die besondere Architektur (Abb. B.6) hervorragend für die Prototypenentwicklung.

Eine Kernkomponente des Chips ist die *Serial Interface Engine* (SIE), welche die Low-Level-Aufgaben der Buskommunikation übernimmt. Diese sind Serialisierung und Parallelisierung, Bit-Stuffing und Destuffing, NRZI-Kodierung und Dekodierung, CRC-Fehlerprüfung und Bussteuerung.

Ein im Chip integrierter Mikroprozessor vom Typ 8051 übernimmt alle weiteren Steuer- und Verwaltungsaufgaben. Da der Mikroprozessor mit einer Taktrate von wahlweise 12, 24 oder 48 MHz arbeiten kann, ist er zu langsam, um am High-Speed-Datentransfer aktiv teilzunehmen. Die SIE ist daher direkt mit den Endpoint-FIFOs verbunden, die wiederum entweder direkt oder über eine programmierbare Schnittstelle (GPIF[6]) mit den Benutzerports verbunden werden kann.

Beim GPIF handelt es sich um einen programmierbaren Zustandsautomaten, der dem Benutzer die Möglichkeit bietet, den Zugriff auf die Endpoint-FIFOs frei nach den Bedürfnissen seiner Anwendung zu modellieren. Durch GPIF kann das Verhalten z.B. von speziellen Leitungen zur Steuerung des Datenflusses (*Strobes* bzw. *Handshake*-Leitungen) detailliert beschrieben werden. Wartezustände und bedingte Schleifen können mit dem Zustand der Steuer und Datenleitungen gekoppelt und so optimal und ohne nennenswerte Einbußen an Transferleistung den Anforderungen der Anwendung angepasst werden.

Das Assembler-Programm für den Mikroprozessor, allgemein *Firmware* genannt, wird vom Gerätenentwickler bereitgestellt und in einem von Cypress patentierten Prozess namens *ReNumeration* nach der „normalen“ Enumeration bei der Inbetriebnahme von einer ersten Betriebssystem-Treiberstufe (*Bootstrap*-Treiber) geladen. In der Firmware enthalten sind auch, neben eventuell benötigten GPIF-Modellen, neue vom Entwickler definierte Deskriptoren, die dem Chip eine neue „Identität“ geben und die durch die Firmware definierten neuen Eigenschaften beschreiben. Nach dem Bootstrap führt der Chip ein Reset aus und meldet sich dem Host erneut unter seiner neuen Identität. Das Betriebssystem lädt daraufhin einen neuen, vom Entwickler für die Firmware passend erstellten Treiber.

Die Firmware kann auch in einem externen programmierbaren Festwertspeicher (PROM[7]) gelagert sein, das den Chip bei Inbetriebnahme programmiert. Auf diese Weise entfällt der Bootstrap und die ReNumeration findet sofort statt.

[6] *General Programmable Interface*. Meistens werden GPIF-Modelle die das Verhaltens der Schnittstelle und ihrer Steuerleitungen beschreibt *Waveforms* genannt

[7] *Programmable Read-Only Memory*. Häufig werden löschbare Speichertypen wie *Erasable PROM* (EPROM) oder *Electrically Erasable PROM* (EEPROM) eingesetzt.

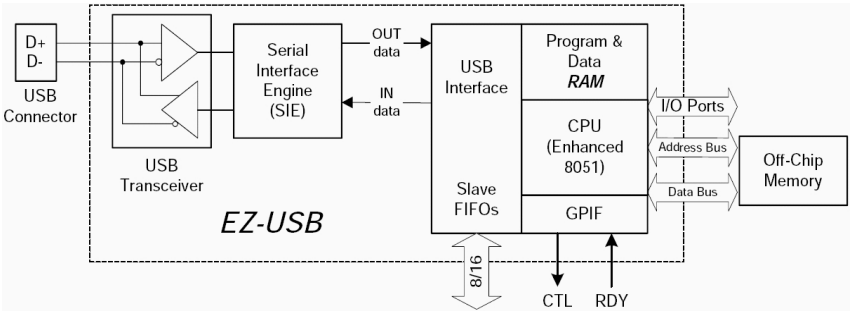

Abbildung B.6: Vereinfachtes Blockschema des EZ-USB-Chips [Cyp04]

Anhang C

Abkürzungen und Akronyme

ADC	Analog to Digital Converter
ASIC	Application Specific Integrated Circuit
CASS/UCSD	Center for Astrophysics and Space Sciences of the University of California, San Diego, CA, USA
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
CPLD	Complex Programmable Logic Device
CRC	Cyclic Redundancy Check
CSR	Control and Status Register
CZT	CdZnTe, Cadmium-Zink-Tellurid
DAC	Digital to Analog Converter
DCM	Digital Clock Managers
EEPROM	Electrically Erasable PROM
EPROM	Erasable PROM
FIFO	First In First Out
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
GPIF	General Programmable InterFace
HXI	Hard X-ray Imager
IAAT	Institut für Astronomie und Astrophysik Tübingen
JTAG	Joint Test Action Group
LUT	Look-Up Table
LVDS	Low Voltage Differential Signaling
MIRAX	Monitor e Imageador de RAyos X
MUX	MUltipleXer
NRZI	Non Return to Zero Inverted
PCI	Peripheral Component Interconnect
PID	Packet-IDentifier
PLL	Phase-Locked Loop
PROM	Programmable Read-Only Memory
RAM	Random Access Memory
RCI	RENA Control Interface

RENA	Readout Electronics for Nuclear Application
SXI	Soft X-ray Imager

Literaturverzeichnis

[Ash02] Ashenden, Peter J.: *The designer's guide to VHDL*. 2nd edition. Morgan Kaufmann Publishers, 2002

[Com00] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips: *Universal Serial Bus Specification*. Revision 2.0. April 2000

[Cyp04] Cypress Semiconductor. 3901 North First Street, San Jose CA 95134: *EZ-USB Technical Reference Manual*. Revision 1.1. 2004

[Kel03] Kelm, K. J. (Hrsg.): *USB 2.0*. 2. Franzis' Verlag GmbH, 85586 Poing, 2003

[KM00] Kalemci, E. ; Matteson, J. L.: Investigation of charge sharing among electrode strips for a CdZnTe detector. In: *Preprint submitted to Elsevier Preprints* (2000)

[MTK+98] Maeding, D. G. ; Tumer, T. O. ; Kravis, S. D. ; Visser, G. I. ; Yin, S.: Readout electronics for nuclear applications (RENA) IC. In: *Proc. SPIE* 3445 (1998), S. 364–373

[Nat96] National Instruments Corporation: *PC-DIO-96/PnP User Manual*. September 1996

[NOV01] NOVA R&D Inc. 1525 Third St., Ste. C, Riverside CA 92507: *Specifications and Manual for the RENA Evaluation System*. Sept. 21 2001

[R+03] Rothschild, R. E. [u. a.] : Hard X-ray Imaging for Mirax / Center for Astrophysics and Space Sciences, University of California San Diego. 2003. – Forschungsbericht

[S+00] Slavis, Kimberly R. [u. a.] : Performance of a prototype CdZnTe detector module for hard X-ray Astrophysics. In: *Proc. SPIE* 4140 (2000), S. 24

[Sch01] Schanz, Thomas: *Entwicklung und Test eines Event-Pre-Prozessors für einen CdZnTe-Detektor*, Institut für Astronomie und Astrophysik der Universität Tübingen, Diplomarbeit, 2001

[SR+03] Staubert, R. ; Rothschild, R. E. [u. a.] : MIRAX: a hard X-ray imaging mission. In: *Proc. SPIE* 4851 (2003), S. 365–376

[ST04] Schanz, Thomas ; Tenzer, Christoph: Hardware Synthese mit VHDL / Institut für Astronomie und Astrophysik der Universität Tübingen. 2004. – Forschungsbericht

[Suc04] Suchy, Slawomir: *Aufbau einer Testumgebung und eines Röntgenmeßstandes für den Event-pre-Prozessor bei der MIRAX-Mission*, Institut für Astronomie und Astrophysik der Universität Tübingen, Diplomarbeit, 2004

Danksagung

An dieser Stelle möchte ich mich noch bei allen bedanken, die mir während meines Studiums und der Diplomarbeit mit ihrer Unterstützung sehr geholfen haben. Für das freundliche und respektvolle Miteinander, das ein sehr angenehmes Klima im gesamten Institut geschaffen hat, geht mein Dank an alle Mitglieder. Einige Personen will ich hier namentlich erwähnen:

Prof. Dr. Rüdiger Staubert, **Prof. Dr. Andrea Santangelo** und **Dr. Eckhard Kendziorra**, für meine Aufnahme in ihre Arbeitsgruppe und die Vergabe dieser Diplomarbeit; eine Aufgabe, die ich sehr gerne angenommen habe und die mir, neben der Mühe, auch sehr viel Freude bereitet hat.

Nikolai von Krusenstiern, für viele konstruktive Gespräche und seinen unermüdlichen Einsatz als Lektor dieser Diplomarbeit, der auch geholfen hat, manchen typografischen Unsinn zu vermeiden.

Thomas Schanz und **Christoph Tenzer**, für ihre hilfreichen Tipps, die mir den Einstieg in die Welt der digitalen Elektronik und VHDL erleichtert haben.

Olaf Luz, **Siegfried Vetter** und **Falko Vogt**, für ihre Hilfe bei der Platinenentflechtung und der Realisierung des Prototyps.

Kolja Giedke, **Nicolay Hammer**, **Daniel Kusterer**, **Michael Martin** und **Slawomir Suchy**, für ihre Hilfsbereitschaft in vielen Situationen, in denen es manchmal „ganz eng“ wurde und nicht zuletzt für einige entspannende Abende.

Meine Frau **Angelika**, die u.a. auch einiges Korrektur lesen musste, Meine **Eltern** und **Geschwister**, für ihre Geduld, ihr Verständnis und ihre Unterstützung bei der Realisirung dieses lang gehegten Wunsches.

Danke