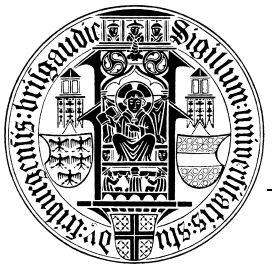
# Integration eines RISC-Mikroprozessors zur Überwachung des CATCH

Matthias Hoffmann



FAKULTÄT FÜR PHYSIK
ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG

# Integration eines RISC Mikroprozessors zur Überwachung des CATCH

**Diplomarbeit**
vorgelegt
von
**Matthias Hoffmann**

Fakultät für Physik
Albert-Ludwigs-Universität
Freiburg i. Br.

16. Oktober 2001

*La science, arrivée aux derniers abîmes, rencontre l'imagination.*

*Victor Hugo*

*It is now known to science that there are many more dimensions than the classical four. Scientists say that these don't normally impinge on the world because the extra dimensions are very small and curve in on themselves, and that since reality is fractal most of it is tucked inside itself. This means either that the universe is more full of wonders than we can hope to understand or, more probably, that scientists make things up as they go along*

*Terry Pratchett, Pyramids*

# Inhaltsverzeichnis


**1 Einleitung** **1**

**2 Das COMPASS-Experiment** **3**
2.1 Physikalische Zielsetzung . . . 3
2.1.1 Hadronenprogramm . . . 3
2.1.2 Myonenprogramm . . . 3
2.2 Spinstruktur des Nukleons . . . 4
2.3 Messung der Gluonenpolarisation $\Delta G/G$ . . . 6
2.4 Aufbau des Experiments . . . 9
2.4.1 Strahl und Target . . . 9
2.4.2 Aufbau des Detektors . . . 10

**3 Die Auslese-Elektronik bei COMPASS** **13**
3.1 Front-End-Karten . . . 13
3.1.1 Die Front-End-Karte für Straws . . . 15
3.2 HOTLink und HOTFibre . . . 15
3.3 CATCH . . . 17
3.3.1 Mezzanine-Karten . . . 17
3.4 Zwischenspeicherung und Filterung der Daten . . . 19
3.4.1 S-Link . . . 19
3.4.2 Read-Out-Buffer . . . 19
3.4.3 Computer . . . 20
3.4.4 Netzwerk und Data Storage . . . 20
3.5 Das Trigger-Control-System . . . 22

**4 CATCH** **23**
4.1 CATCH Überblick . . . 23
4.2 Funktionen des CATCH-Moduls . . . 25
4.2.1 Trigger und Clock . . . 26
4.2.2 Serielle Datenleitung . . . 26
4.2.3 VME-Interface . . . 26
4.3 Funktionen der FPGAs . . . 27
4.3.1 TCS-FPGA . . . 27

4.3.2 Merger-FPGA . . . 27
4.3.3 Formatter-FPGA . . . 28
4.3.4 S-Link-FPGA . . . 29
4.4 Der Control-FPGA . . . 29
4.4.1 Steuerfunkionen des Control-FPGAs . . . 30
4.5 XCONTROL - Ein Mikroprozessor im FPGA . . . 30

**5 XSOC** **33**
5.1 Prinzipieller Aufbau eines RISC-Prozessors . . . 33
5.1.1 Datenpfad (Data-Path) . . . 33
5.1.2 Kontrolleinheit (Control-Unit) . . . 35
5.1.3 Memory-Controller . . . 35
5.2 Der `xr16` Befehlssatz . . . 35
5.2.1 Register . . . 35
5.2.2 Befehlssatz . . . 36
5.3 Der Prozessorkern `xr16` . . . 38
5.3.1 ALU . . . 38
5.3.2 Data-Path: Pipelining . . . 41
5.4 Control-Bus und Peripherie . . . 44

**6 XCONTROL** **45**
6.1 Prinzipien . . . 45
6.2 RAM-Zugriff . . . 45
6.3 CPU starten und stoppen . . . 47
6.4 Display . . . 47
6.4.1 Adressierung des Displays - Memory Mapping . . . 48
6.5 Bus-Interface . . . 49
6.5.1 Ansteuerung des FPGA-Bus . . . 49
6.6 Reset der FPGAs . . . 50
6.7 Serielles Interface . . . 52
6.7.1 Daten schreiben . . . 52
6.7.2 Busy-Flag lesen . . . 53
6.7.3 Setup-Daten aus dem RAM schreiben . . . 53

**7 Implementierung** **55**
7.1 Aufbau eines FPGAs . . . 55
7.2 XSOC Quellen . . . 57
7.3 Anpassen von XSOC . . . 57
7.3.1 RLOC-Constraints . . . 57
7.3.2 VGA . . . 58
7.3.3 Clock . . . 58
7.3.4 Pull-Up Widerstand . . . 58
7.4 Implementierung der neuen Schnittstellen . . . 58
7.4.1 Umschaltung zwischen “legacy”-Modus und CPU-Modus . . . 58

7.4.2 RAM-Zugriff . . . 59
7.4.3 FPGA-Bus . . . 60
7.4.4 FPGA-Reset . . . 61
7.4.5 Display . . . 62
7.4.6 Serielles Interface . . . 62
7.5 Design-Software . . . 63
7.5.1 Entwurf und Implementierung . . . 64
7.5.2 Probleme . . . 65
7.6 Simulation . . . 66
7.6.1 Timing-Simulation . . . 67
7.6.2 Skripte . . . 67

**8 Fazit** **71**

**A Anwendung von XCONTROL** **73**
A.1 Programmierung des CATCH . . . 73
A.2 Ein Programm ins RAM schreiben . . . 73
A.2.1 Hex2ram . . . 73
A.3 Initialisierung der Front-End-Karten . . . 75
A.3.1 F1toram . . . 75
A.3.2 `ramtoreg` . . . 76
A.4 Compiler . . . 77
A.4.1 Installation . . . 77
A.4.2 Compilieren . . . 77
A.5 Die Runtime-Library `libxr16` . . . 78
A.5.1 `zeromem` . . . 78
A.5.2 `mulu2` . . . 78
A.5.3 Listing . . . 79
A.5.4 `divi2` und `modi2` . . . 80

**B Einige technische Einzelheiten** **83**
B.1 Technisches zum Display . . . 83
B.2 Listing für die Timingsimulation . . . 86

**C Schaltplan** **91**

**D Glossar** **97**

**Abbildungsverzeichnis** **103**

**Tabellenverzeichnis** **105**

**Literaturverzeichnis** **107**

**Erklärung** **109**

# Kapitel 1

# Einleitung

COMPASS ist ein Experiment mit stationärem Target, das sich zur Zeit am CERN in Genf im Betrieb befindet, und mit dem im Jahr 2001 die ersten physikalischen Daten genommen wurden. Die Ziele des Experiments sind vielfältig: einerseits soll Spektroskopie mit Hadronen betrieben werden, andererseits die Spin-Struktur des Nukleons untersucht werden. Ein wichtiges Ziel hierbei ist die Bestimmung der Gluonenpolarisation.

COMPASS verfügt über einen aufwändigen Detektor; er muss in der Lage sein, sehr hohe Ereignisraten zu verarbeiten. Dies stellt insbesondere hohe Anforderungen an die Geschwindigkeit der Auslese-Elektronik des Detektors. Hinzu kommt, dass insgesamt über 250 000 Kanäle ausgewertet werden müssen. Um eine schnelle Verarbeitung zu gewährleisten, werden die Daten direkt am Detektor digitalisiert und an die nächste Stufe der Verarbeitungskette übertragen. Sie können dort bereits gebündelt und vorsortiert werden, bevor die Aufzeichnung der Daten erfolgt. Dies setzt eine schnelle und effiziente Elektronik voraus, von der wesentliche Teile in Freiburg entwickelt wurden.

In Kapitel 2 werden die physikalischen Ziele von COMPASS kurz erläutert sowie auf den Aufbau des Detektors eingegangen. Das Konzept und die einzelnen Stufen der Auslese-Elektronik werden in Kapitel 3 vorgestellt.

Herzstück der Ausleseelektronik bei COMPASS ist das sogenannte CATCH-Modul. Das Akronym CATCH steht dabei für COMPASS Accumulate, Transfer and Control Hardware. Das CATCH-Modul übernimmt vielfältige Aufgaben wie zum Beispiel die Vorverarbeitung der digitalisierten Detektordaten, die Verteilung der Trigger und des Clock-Signals an die Elektronik am Detektor sowie deren Initialisierung.

Die vorliegende Arbeit beschäftigt sich mit der Implementierung eines Mikroprozessors, der Steuerungs- und Überwachungsfunktionen auf dem CATCH-Modul ausführt. Die Verwendung eines Mikroprozessors auf dem CATCH ermöglicht eine flexible Anpassung der Steuerfunktionen, da sie in einer Hochsprache wie C geschrieben werden können. Der Mikroprozessor liegt nicht als fest verdrahtetes elektronisches Bauteil vor, sondern ist Bestandteil eines “Programms” eines rekonfigurierbaren Logikbausteins, eines so genannten FPGAs. Eine kurze Einführung in die Möglichkeiten von FPGAs sowie einige Implementierungsdetails werden in Kapitel 7 gegeben.

Der Entwurf der Logik für den FPGA ist der Schwerpunkt dieser Diplomarbeit. Die Herausforde-

rung beim Entwurf war die Integration eines Mikroprozessors unter Beibehaltung schon existierender Funktionen. Als Mikroprozessorkern wurde ein bereits bestehendes Projekt angepasst, was die Umsetzung des Projekts innerhalb von neun Monaten ermöglichte. Die Architektur und der Befehlssatz des verwendeten Mikroprozessorkerns sind Thema von Kapitel 5.

Das Ergebnis der Entwicklungsarbeit ist XCONTROL, eine flexible, RISC-Prozessor-gesteuerte Kontrolleinheit für das CATCH-Modul.
Die wichtigsten Details zur Anwendung von XCONTROL werden in Anhang A erläutert.

Im Anhang B werden einige technische Details behandelt, in Anhang C ist der komplette Schaltplan von XCONTROL abgedruckt.

# Kapitel 2

# Das COMPASS-Experiment

COMPASS [1] steht für COmmon Muon and Proton Apparatus for Structure and Spectroscopy. Dabei handelt es sich um ein im Jahre 2001 am CERN anlaufendes Experiment mit stationärem Target.

## 2.1 Physikalische Zielsetzung

Die Ziele des Experiments sind in zwei Bereiche gegliedert: ein Programm mit Myonenstrahl und ein Programm mit Hadronenstrahlen.
Dabei wird ausgenutzt, dass am SPS[1] des CERN unterschiedliche Arten von Teilchenstrahlen von 100 GeV bis zu mehreren hundert GeV verfügbar sind. Sie werden aus einem hochenergetischen Protonenstrahl durch Wahl verschiedener Produktions-Targets sowie zwischengeschalteter Filter erzeugt.

### 2.1.1 Hadronenprogramm

Eines der Ziele des hadronischen Programms bei COMPASS ist es, exotische Gluonen-Zustände zu finden, die von der QCD vorausgesagt werden. Hinweise auf einen solchen Zustand, z.B. $f_0$ bei $1500\,MeV/c^2$, lieferte das Crystal Barrel-Experiment [2]. Weitere Hauptziele sind die Spektroskopie von charm-haltigen Hadronen und das Studium der hadronischen Struktur instabiler Teilchen mit Primakoff-Reaktionen. Die Experimente mit Hadronenstrahlen werden den zweiten, späteren Abschnitt des COMPASS-Programmes bilden. Daher soll im Folgenden ausschließlich auf das Myonenprogramm, das bereits im Jahre 2001 erste Daten liefert, eingegangen werden.

### 2.1.2 Myonenprogramm

Das Hauptziel des Myonenprogramms ist ein besseres Verständnis der Spinstruktur des Nukleons. Ende der achtziger Jahre wurde am EMC-Experiment (European Muon Collaboration) entdeckt [3], dass nur ein Bruchteil des Nukleonspins von den Quarks herrührt. Die EMC-Ergebnisse wurden an anderen Experimenten am CERN, SLAC und DESY bestätigt. Dies bedeutet, dass andere Komponenten wie etwa die Gluonenpolarisation einen großen Anteil zum Gesamtspin beitragen.

[1] Super Proton Synchrotron

$$Q^2 = -q^2 = (k - k')^2$$

$$\nu = \frac{P \cdot q}{M} \overset{lab}{=} E - E'$$

$$x = \frac{Q^2}{2P \cdot q} = \frac{Q^2}{2M\nu}$$

$$y = \frac{P \cdot q}{P \cdot k} \overset{lab}{=} \frac{E - E'}{E}$$

$$\hat{s} = (q + k)^2$$

$$\eta = x_g = \frac{\hat{s}}{2M\nu}$$

| | |
|---|---|
| $k$, $k'$ | : Impuls des einfallenden bzw. des gestreuten Myons |
| $P$ | : Impuls des Protons |
| $M$ | : Masse des Protons |
| $E$, $E'$ | : Energie des einfallenden bzw. des gestreuten Myons |

**Tabelle 2.1:** Kinematische Variablen bei der tiefinelastischen Streuung

## 2.2 Spinstruktur des Nukleons

Beim einfachsten Quark-Modell wird der Spin des Nukleons allein von den Valenzquarks getragen:

$$\frac{1}{2} = \frac{1}{2}\Delta\Sigma \tag{2.1}$$

$$\Delta\Sigma = \Delta u + \Delta d + \Delta s = 1$$

Hier bei sind $\Delta u$, $\Delta d$ und $\Delta s$ die Beiträge der Quarks zum Spin, genauer die über $x$ integrierten polarisierten Verteilungsfunktionen der Quarks. Diese werden im Allgemeinen folgendermaßen definiert:

$$\Delta q_f = \int_0^1 \{\left(q_f(x)^{\uparrow\uparrow} + \overline{q}_f(x)^{\uparrow\uparrow}\right) - \left(q_f(x)^{\downarrow\uparrow} + \overline{q}_f(x)^{\downarrow\uparrow}\right)\} \cdot dx \tag{2.2}$$

Die Funktionen $q_f(x)^{\uparrow\uparrow}$ bzw. $q_f(x)^{\downarrow\uparrow}$ geben im Formalismus der tiefinelastischen Streuung die Wahrscheinlichkeitsdichteverteilungen dafür an, dass an einem (Anti-)Quark der Sorte $f = u, d, s$ gestreut wird. Dabei bedeuten $\uparrow\uparrow$ und $\downarrow\uparrow$ parallele bzw. anti-parallele Spin-Einstellungen von Quark und Nukleon (Abbildung 2.1). Die Skalenvariable $x = x_{\text{Bjorken}} = \frac{Q^2}{2M\nu}$ gibt den Anteil des gestreuten Quarks am Viererimpuls des Nukleons in einem schnell bewegten Koordinatensystem[2] an.

[2] Infinite Momentum Frame

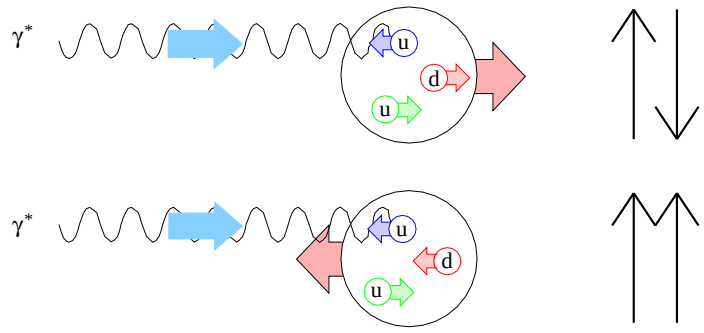



**Abbildung 2.1:** Streuung longitudinal polarisierter Myonen an longitudinal polarisierten Protonen

Betrachtet man das Nukleon als kompliziertes relativistisches Vielkörpersysytem, kann man den Spin als vektorielle Summe der Spins aller Quarks und Gluonen sowie ihrer Bahndrehimpulse betrachten und die Formel (2.1) verallgemeinern zu:

$$\frac{1}{2} = \frac{1}{2}\Delta\Sigma + \Delta G + \langle L_z^q \rangle + \langle L_z^g \rangle \tag{2.3}$$

Hierbei beinhaltet $\Delta\Sigma$ auch Drehimpulse $\langle L_z^q \rangle$ von Quarks und von Gluonen $\langle L_z^g \rangle$. Im Fall des simplen Quarkmodells ist $\Delta u = \frac{4}{3}$, $\Delta d = -\frac{1}{3}$ und $\Delta s$ sowie $\Delta G$ sind Null. Da sich die drei Valenzquarks in einem S-Zustand befinden ist $\langle L_z^q \rangle$ ebenfalls gleich Null, so dass sich aus (2.3) wieder (2.1) mit $\Delta\Sigma = 1$ ergibt.

Das Prinzip der Messung der polarisierten Quarkdichten beruht auf der tiefinelastischen Streuung (DIS[3]) von polarisierten Leptonen an polarisierten Nukleonen. Die Differenz im Wirkungsquerschnitt $\Delta\sigma$ der Streuung für parallele und antiparallele Spins hängt von zwei Strukturfunktionen $g_1$ und $g_2$ ab.

$$\Delta\sigma = a \cdot g_1(x, Q^2) + b \cdot g_2(x, Q^2) \tag{2.4}$$

Dies ist analog zur Situation bei der unpolarisierten tiefinelastischen Streuung, wo der Wirkungsquerschnitt von den Strukturfunktionen $F_1(x, Q^2)$ und $F_2(x, Q^2)$ abhängig ist. Die Koeffizienten $a$ und $b$ sind kinematische Faktoren und können mit Methoden aus der Quantenelektrodynamik berechnet werden. Bei longitudinaler Strahlpolarisation ist $b$ klein, so dass die Messung von $\Delta\sigma$ die Bestimmung von $g_1(x, Q^2)$ erlaubt. Umgekehrt kann man $g_2$ bei Targetpolarisation senkrecht zum Strahl messen.

Im Quark-Parton-Modell (siehe Lehrbücher wie [5]) sind die Strukturfunktionen $g_1$ und $F_1$ einfach mit den Quarkdichten $\Delta q_f(x, Q^2)$ und $q_f(x, Q^2)$verknüpft:

$$g_1(x, Q^2) = \frac{1}{2}\sum e_f^2 \cdot \Delta q_f(x, Q^2)$$

$$F_1(x, Q^2) = \frac{1}{2}\sum e_f^2 \cdot q_f(x, Q^2)$$

Durch Messung von $\Delta\sigma$ über einen weiten Bereich von $x$ und $Q^2$ kann man das erste Moment von $g_1$ bestimmen

[3] engl. Deep Inelastic Scattering

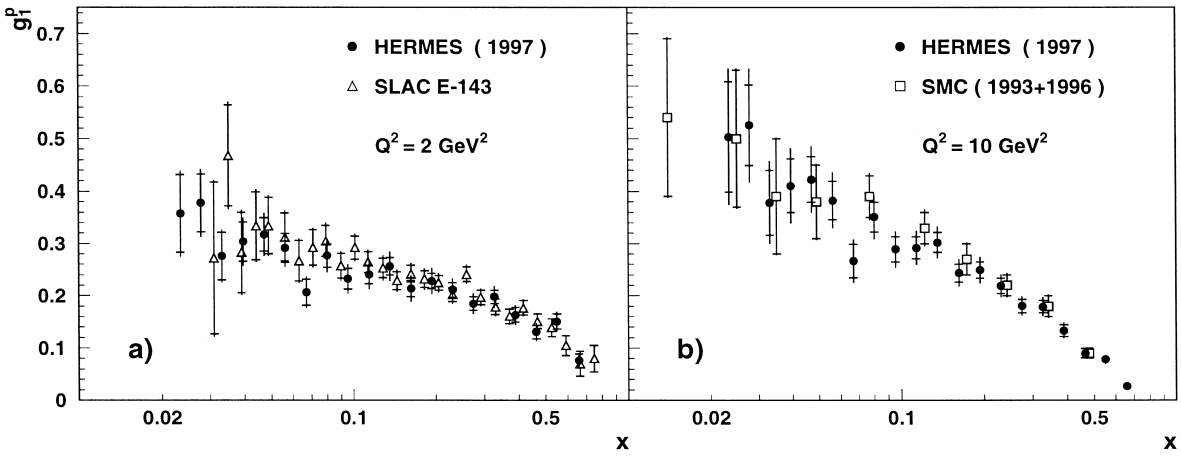



**Abbildung 2.2:** Die Funktion $g_1^p$ aufgetragen gegen $x$. Die Daten wurden bei $Q_0^2 = 2\,\mathrm{GeV}^2$ (links) und bei $Q_0^2 = 10\,\mathrm{GeV}^2$ (rechts) entwickelt (Quelle: [6]).

$$\Gamma_1^{p(n)}(Q^2) = \int_0^1 g_1^{p(n)}(x, Q^2)dx$$

Die Funktionen $\Delta q$ sind mit den SU(3)$_f$ Axialmatrixelementen $a_0, a_3$ und $a_8$ verknüpft so dass man schreiben kann:

$$\Gamma_1^{p(n)}(Q^2) = +(-)\frac{1}{12}a_3 + \frac{1}{36}a_8 + \frac{1}{9}a_0$$

$$a_3 = \Delta u - \Delta d, \quad a_8 = \Delta u + \Delta d - 2\Delta s, \quad a_0 = \Delta u + \Delta d + \Delta s = \Delta\Sigma$$

Dabei vernachlässigt man die Skalenverletzung und ersetzt $\Delta q_f(x, Q^2)$ durch $\Delta q_f(x)$. Aus den bekannten Daten für $a_3$ und $a_8$ aus dem Neutron und Hyperonzerfall kann man $\Delta\Sigma$ bestimmen. Das Ergebnis der EMC-Kollaboration war

$$\Delta\Sigma = 0.12 \pm 0.09 \pm 0.14 \ll 1$$

Die Ergebnisse von EMC wurden inzwischen von vielen Experimenten (SMC am CERN, HERMES am DESY, mehrere Experimente am SLAC) bestätigt. Daten dieser Messungen sind in Abb. 2.2 aufgetragen. Ein neueres Ergebnis der HERMES-Kollaboration [7] ergibt

$$\Delta\Sigma = 0.30 \pm 0.04 \pm 0.09$$

Die Daten bisheriger Experimente können also nicht die Herkunft des Spins des Protons und Neutrons erklären. Auch konnte der Anteil der Gluonen am Spin mit der bisherigen Methode der inklusiven tiefinelastischen Streuung nicht unabhängig gemessen werden.

## 2.3 Messung der Gluonenpolarisation $\Delta G/G$

Um die Spinstruktur des Nukleons zu verstehen, muss man die Gluonenpolarisation direkt messen. Dies geschieht bei COMPASS durch tiefinelastische Streuung longitudinal polarisierter Myonen an einem ebenfalls longitudinal polarisierten Target. Dabei betrachtet man die Photon Gluon-Fusion (Abb.

2.3) und misst diese Reaktion semi-inklusiv. Bei der Photon-Gluon-Fusion ensteht durch Wechselwir-

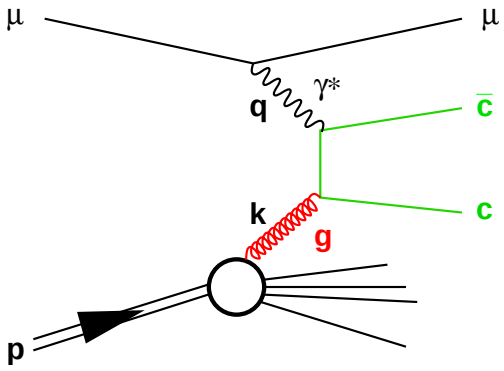



**Abbildung 2.3:** Feynmangraph der Photon-Gluon-Fusion

kung eines virtuellen Photons mit einem Gluon des Nukleons ein Quark-Antiquark-Paar. Bei COMPASS sucht man nach Paaren aus Charm-Quarks bzw. deren Produkten (open charm-Produktion). Durch subtrahieren der Anzahl der durch Streuung mit paralleler bzw. antiparalleler Spineinstellung

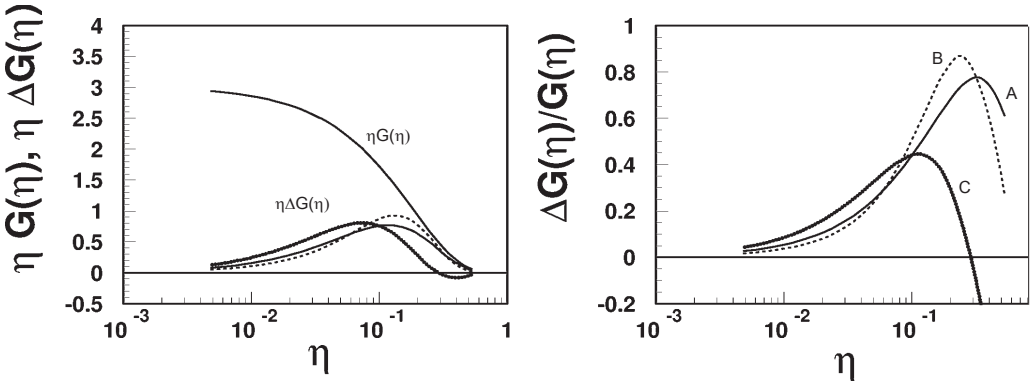



**Abbildung 2.4:** Mögliche Parametrisierungen der polarisierten Gluonverteilungsfunktion $\Delta G$. Links sind $\eta G(\eta)$ (oben) und drei Parametrisierungen von $\eta\Delta G(\eta)$ dargestellt. Rechts sind drei Parametrisierungen von $\Delta G(\eta)/G(\eta)$ abgebildet. Die Notation A, B und C bezieht sich auf Modelle von Gehrmann und Stirling [8].

von Strahl und Target erzeugten $c\bar{c}$-Paare ($N^{\uparrow\uparrow}_{c\bar{c}}, N^{\uparrow\downarrow}_{c\bar{c}}$) erhält man die gemessene Asymmetrie $A^{\text{exp}}$, die mit der Asymmetrie $A^{c\bar{c}}_{\mu p}$ der Reaktion $\mu + N \rightarrow c\bar{c} + X$ im folgendem Zusammenhang steht:

$$A^{\text{exp}} = \frac{N^{\uparrow\downarrow}_{c\bar{c}} - N^{\uparrow\uparrow}_{c\bar{c}}}{N^{\uparrow\downarrow}_{c\bar{c}} + N^{\uparrow\uparrow}_{c\bar{c}}} = P_S \cdot P_T \cdot f \cdot A^{c\bar{c}}_{\mu p}(x, y, Q^2). \tag{2.5}$$

Dabei ist $P_S$ die Polarisation des Strahls, $P_T$ die des Targets und $f$ der Bruchteil der polarisierbaren Nukleonen im Targetmaterial[4]. Aus $A^{c\bar{c}}_{\mu p}$ lässt sich die Asymmetrie der Photon-Gluon-Fusion $A^{c\bar{c}}_{\gamma p}$

[4]engl.: dilution factor, $f = 0.176$ für $NH_3$, $f = 0.5$ für LiD

berechen:

$$A_{\mu\mathsf{p}}^{\mathsf{c\bar{c}}} = D \cdot A_{\gamma\mathsf{p}}^{\mathsf{c\bar{c}}}.$$

$D$ ist der Depolarisationsfaktor, der den Anteil der virtuellen Photonen widergibt, die Polarisation des Myons übernehmen. $D$ hängt vom Verhältnis $y = \frac{\nu}{E}$ der Energien $\nu$ und $E$ von virtuellem Photon und Myon ab:

$$D(y) \approx \frac{1-(1-y)^2}{1+(1-y)^2}.$$

Die Asymmetrie $A_{\gamma\mathsf{p}}^{\mathsf{c\bar{c}}}$ ist das Verhältnis der Differenz des helizitätsabhängigen Wirkungsquerschnitts $\Delta\sigma^{\gamma\mathsf{p}\to\mathsf{c\bar{c}X}}$ und des helizitätsgemittelten Wirkungsquerschnitts $\sigma^{\gamma\mathsf{p}\to\mathsf{c\bar{c}X}}$ der Charm-Quark-Produktion. Diese Wirkungsquerschnitte können durch die Faltung der bekannten, elementaren Photon-Gluon-Wirkungsquerschnitte $\Delta\sigma(\hat{s})$ und $\sigma(\hat{s})$ mit den Gluon-Verteilungen $\Delta G$ und $G$ berechnet werden:

$$A_{\gamma\mathsf{p}}^{\mathsf{c\bar{c}}}(E,y) = \frac{\Delta\sigma^{\gamma\mathsf{p}\to\mathsf{c\bar{c}X}}}{\sigma^{\gamma\mathsf{p}\to\mathsf{c\bar{c}X}}} = \frac{\int_{4m_\mathsf{c}^2}^{2ME_y} \mathsf{d}\hat{s}\ \Delta\sigma(\hat{s})\Delta G(\eta,\hat{s})}{\int_{4m_\mathsf{c}^2}^{2ME_y} \mathsf{d}\hat{s}\ \sigma(\hat{s})G(\eta,\hat{s})} \tag{2.6}$$

mit

$$\sigma^{\gamma g\to\mathsf{c\bar{c}}} = \sigma(\hat{s}) + \lambda_\gamma\lambda_g\Delta\sigma(\hat{s}),$$

Die Faktoren $\lambda_\gamma$ und $\lambda_g$ sind die Helizitäten des Photons und des Gluons, $\hat{s} = (q+k)^2$ ist das Quadrat der Schwerpunktsenergie des Photon-Gluon-Systems, $M$ und $m_\mathsf{c}$ sind die Massen des Nukleons und des Charm-Quarks. Die Variable

$$\eta = x_g = \frac{\hat{s}}{2M\nu}$$

ist das Gluon-Äquivalent der Skalenvariable $x$ ($x_{\text{Bjorken}} = \frac{Q^2}{2M\nu}$).
Zusammenfassend ist es also auf die oben beschriebene Weise möglich, bei bekanntem Wirkungsquerschnitt für die Photon-Gluon-Fusion aus der Messung von $A^{\text{exp}}\ \Delta G/G$ zu extrahieren.

Die Rekonstruktion der Photon-Gluon-Fusionsereignisse geschieht durch die Detektion der dabei entstehenden charm-haltigen Endprodukte. Insgesamt rechnet man mit einer Ereignisrate $\mathrm{N}^{\mathsf{c\bar{c}}}$ von 82 000 pro Tag bei einer Luminosität von $4.4 \times 10^{37}\mathrm{c}m^{-2}\mathrm{d}^{-1}$. Genauer wird das entstehende $\mathrm{D}^0$-Meson betrachtet, in welches das Charm-Quark mit sechzigprozentiger Wahrscheinlichkeit fragmentiert. Das $\mathrm{D}^0$-Meson selbst zerfällt mit 3.85 Prozent Wahrscheinlichkeit gemäß:

$$\mathrm{D}^0 \to \mathrm{K}^- + \pi^+ \quad \text{(und ladungskonjugiert)} \tag{2.7}$$

Die K- und $\pi$-Mesonen können im COMPASS-Detektor verfolgt werden. Durch Einschränkung des Zerfallswinkels $\Theta_K^*$ (Abb. 2.5) auf Werte zwischen 60 und 120 Grad im Ruhesystem des D-Mesons ($|\cos\theta_K^*| < 0.5$) und ein Fenster für die $\mathsf{D}^0$-Masse von 40 MeV Breite kann der Untergrund unterdrückt werden. Man rechnet mit Ereignisraten fr $\mathsf{D}^0$ und $\overline{\mathsf{D}^0}$ von 880 pro Tag gegenüber einem kombinatorischen Untergrund von 3200 bis 3400 pro Tag. Das Verhältnis von Signal zu Untergrund kann weiter verbessert werden, wenn zusätzlich die Reaktion betrachtet

$$\mathsf{D}^* \to \mathsf{D}^0 + \pi_\mathsf{s}^0$$

und das beim Zerfall entstandene “weiche” Pion berücksichtigt.

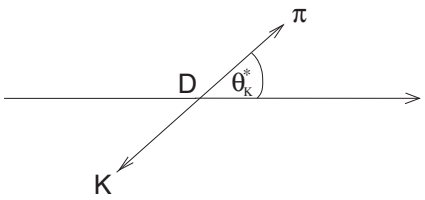



**Abbildung 2.5:** Zerfall des $D^0$-Mesons

## 2.4 Aufbau des Experiments

### 2.4.1 Strahl und Target

**Strahlpolarisation** Die Myonen für die $\Delta G/G$-Messung werden am SPS durch Beschuß eines 500mm dicken Berylliumtargets mit Protonen erzeugt. Dabei entstehen zunächst Hadronen (Protonen, Pionen, Kaonen), die mit einem Spektrometermagneten mit Apertur nach Impuls selektiert werden. Um einen polarisierten Myonenstrahl zu erhalten, wird ausgenutzt, dass der Zerfall der Pionen $\pi^+ \rightarrow \nu_\mu + \mu^+$ maximal paritätsverletzend ist. Dadurch sind die entstandenen Myonen mit hohen Impulsen entgegen ihrer Flugrichtung polarisiert. Mit einem zweiten Magneten und Blenden ("scaper") kann der Impuls der Myonen gewählt werden. Die Hadronen werden mit einem im zweiten Magneten integrierten Absorber herausgefiltert. Die Strahlpolarisation $P_\mathrm{s}$ der Myonen beträgt 0.8. Zur Messung des Impulses der Myonen dienen die Beam Momentum Stations, die ca. 60 m vor dem Target angebracht sind. Sie bestehen aus je einer Ebene Szintillator-Hodoskope vor und nach dem letzten Umlenkmagneten, der den Strahl vom unterirdischen Teil der Beamline nach oben zur Experimentebene lenkt.
Für das Experiment sind Strahlenergien zwischen 100 und 200 GeV (im Augenblick 160 GeV) vorgesehen, wobei eine Intensität von $2 \times 10^8$ Myonen pro Spill erreicht wird.

**Polarisiertes Target** Für das polarisierte Target wird das SMC-Target verwendet [9]. Um die gewünschte Hadronenakzeptanz von $\pm$180 mrad zu erreichen, wurde ein neuer supraleitender Magnet in Auftrag gegeben, der aber noch nicht zur Verfügung steht. Im Moment wird noch der alte SMC-Magnet verwendet. Das Target enthält zwei Zellen mit entgegengesetzter Polarisation, die jeweils 60 cm lang sind. Durch die vergleichsweise große Länge des Targets ist es nicht möglich, Vertexdetektoren einzusetzen. Als Targetmaterial wird $NH_3$ als Protonentarget und $^6$LiD als Deuterontarget eingesetzt. Das Targetmaterial $^6$LiD hat einen sehr vorteilhaften Verdünnungsfaktor-Faktor $f$ (vgl. Formel 2.5). Mit den Targetmaterialien und dem eingesetzten Magneten sind Polarisationen von 85% bzw. 50% erreichbar.

### 2.4.2 Aufbau des Detektors

Zu Anfang des Experiments steht aus finanziellen Gründen noch nicht der volle Aufbau des Spektrometers zur Verfügung. Ein reduzierter Aufbau (“initial layout”, vgl. Abb. 2.6) ist aber ausreichend für das Myonenprogramm und die Bestimmung von $\Delta G/G$.

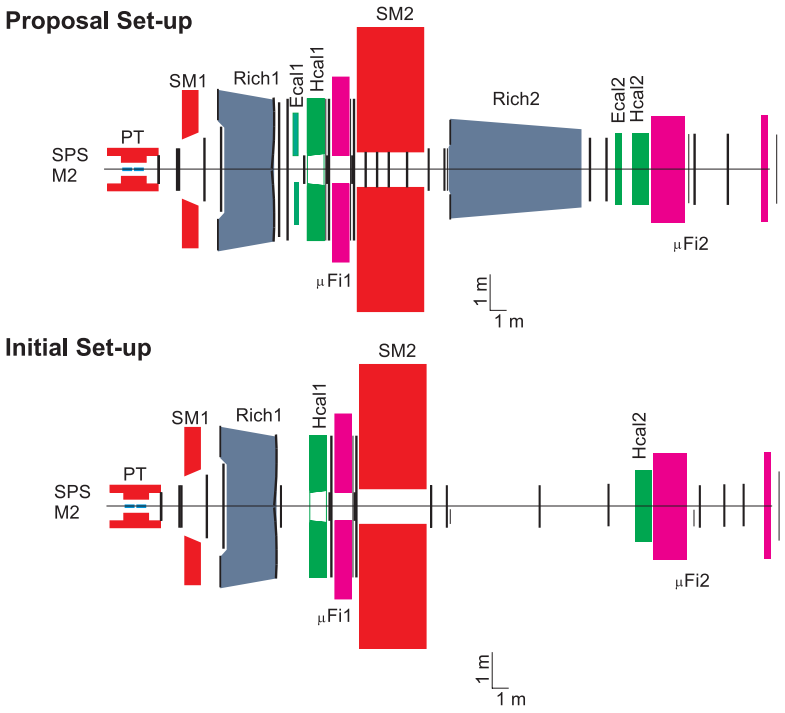



**Abbildung 2.6:** Aufbau des COMPASS-Detektors in der endgültigen Ausbaustufe (oben) und zu Anfang des Experiments

Das System aus Target und Detektor zeichnet sich durch eine große Winkelakzeptanz von 200 mrad aus, der für die Detektion der beim $D^0$-Zerfall entstandenen Kaonen und Pionen notwendig ist. Diese Akzeptanz wird erst mit dem neuen Targetmagneten ausgenutzt werden.

**Spektrometer** Das Experiment ist in zwei Spektrometerstufen unterteilt, eine für kleine und eine für große Ablenkinkel.

Die Spektrometerstufe für große Winkel – also für niederenergetische Myonen – besteht aus dem Spektrometermagneten SM1 mit einer Ablenkkraft von 1 Tm sowie je einer Trackingstation vor und hinter dem Magneten.

Für das Tracking werden verschiedene Arten von Detektoren eingesetzt, je nach zu erwartender Teilchendichte. Für Gebiete mit hohem Teilchenfluß werden Micromega (vor SM1) und Triple-GEM-Detektoren (hinter SM1) eingesetzt (Small Angle Tracking). In der Strahlregion selbst werden szintillierende Fasern verwendet. Für das Large-Angle-Tracking werden verschiedene Typen von Driftkammern mit 3 bis 5 mm Driftraum benutzt. Zwischen Target und SM1 befinden sich planare

Driftkammern mit vier Ebenen, hinter SM1 werden Driftröhrchendetektoren (Straws) eingesetzt. Der Driftröhrchendetektor hat eine aktive Fläche von $3.2 \times 2.4$ m$^2$ und ist aus über 13 000 bis zu 3.25 m langen Driftröhrchen aufgebaut, die in drei Richtungen orientiert sind. Der Detektor wird erst im Jahre 2002 vollständig ausgebaut sein.

Die zweite Spektrometerstufe besteht aus dem bereits bei SMC verwendeten Magneten SM2, der eine Stärke von 4.4 Tm hat. Das Large-Angle-Tracking der zweiten Stufe basiert auf 11 MWPCs mit einer nutzbaren Fläche von $150 \times 80$cm$^2$ und Drahtabständen von 2 mm mit insgesamt 23 000 Kanälen.

**Siliziumstreifenzähler** Unmittelbar vor dem polarisierten Target befindet sich ein Silizium-Streifenzähler-Detektor, der zur Verfolgung des Strahls dient.

**Teilchenidentifikation** Wichtig für die Teilchenidentifikation ist der Ring-Imaging Čerenkov-Detektoren (RICH). Er ist essentiell zur Unterscheidung zwischen Kaonen und Pionen aus dem $D^0$-Zerfall. Der Radiator hat eine Länge von 3 m, als Medium wird $C_4F_{10}$ verwendet. Als Photonendetektoren werden MWPCs mit CsI Photokathoden eingesetzt, die Wellenlängen kürzer als 200 nm detektieren können. Im Initial-Setup wird nur ein RICH zur Verfügung stehen, in einer späteren Ausbaustufe des Experiments ist noch ein zweiter RICH vorgesehen.

Die Identifikation der Myonen geschieht mit sogenannten Muon-Walls, die aus Eisenabsorbern und dahinterliegenden Driftröhren mit 3 cm Durchmesser bestehen.

**Trigger** Der physikalische Trigger basiert auf dem Energieverlust der Myonen [10]. Zur Erzeugung des Triggersignals werden zwei Paare von szintillierenden Hodoskopen verwendet, die sich 35 und 50 m hinter dem Target befinden. Ihre Signale zusammen mit den Informationen aus Kalorimetern erzeugen über eine schnelle Koinzidenzschaltung die physikalischen Trigger.

**Kalorimetrie** Für Kalorimetrie stehen die die elektromagnetischen Kalorimeter ECAL1 und ECAL2 sowie die Hadronkalorimeter HCAL1 und HCAL2 zur Verfügung. Die hadronischen Kalorimeter werden zur Erzeugung des Triggers eingesetzt.

# Kapitel 3

# Die Auslese-Elektronik bei COMPASS

Für den Erfolg des COMPASS-Experimentes ist es notwendig, sehr viele Daten zu sammeln, um damit eine möglichst gute Statistik zu erhalten. Um dies zu erreichen, wird mit sehr hohen Strahlintensitäten gearbeitet, wobei eine totzeitfreie Ausleseelektronik nötig wird. Dabei rechnet man mit einer Ereignisrate im Bereich von 100 kHz, wobei pro Ereignis bis zu 30 KB Daten anfallen können.

Die Grundidee der Datenerfassung beim COMPASS-Experiment besteht darin, möglichst viele Daten bereits am Detektor zu digitalisieren und mit schnellen Datenleitungen zur zentralen Datenerfassung weiterzuleiten. Dies erspart eine aufwändige und fehleranfällige Verkabelung für Analogsignale. Dieses Vorgehen bedeutet gleichzeitig eine nicht unerhebliche Kostenersparnis, da sehr viele Kanäle bei COMPASS ausgelesen werden müssen. Einen Überblick über die gesamten Auslese-Elektronik zeigt Abbildung 3.1.

Ein wichtiges Bindeglied in der Kette der Auslese-Elektronik ist das in Freiburg entwickelte CATCH-Modul, das die Verbindung zur Front-End-Elektronik herstellt und die einlaufenden Daten bereits vorverarbeitet.

## 3.1 Front-End-Karten

Zur Digitalisierung unmittelbar am Detektor dienen so genannte Front-End-Karten. Diese sind speziell auf die Eigenschaften des auszulesenden Detektorelementes angepasst. Für die Detektoren, die zur Teilchenverfolgung (Tracking) dienen, müssen die Front-End-Karten eine genaue Zeitmessung ermöglichen. Für das Tracking in unmittelbarer Nähe des Strahls (Small Angle Tracking, SAT, vgl. Abb. 3.2) werden Detektortypen wie szintillierende Fasern (Sci-Fi), MicroMegas etc. benutzt. Sie liefern eine sehr hohe Datenrate. Andererseits gibt es auch Detektoren zur Verfolgung von Teilchen weiter vom Strahl entfernt (Large Angle Tracking, LAT), hierbei handelt es sich um Driftröhrchen (Straws), Driftkammern, MWPCs sowie die Myonenkammern 1 und 2. Für die Zeitmessung wurde in Freiburg mit dem $\mathcal{F}$1-Chip ein spezieller TDC (Time to Digital Converter) entworfen, der die gewünschte extrem genaue Zeitauflösung ermöglicht [11].

Bei den Detektoren zur Teilchenidentifikation (RICH) und für das Kalorimeter müssen dagegen analoge Messgrößen in Digitalsignale gewandelt werden. Die zugehörigen Front-End-Boards werden

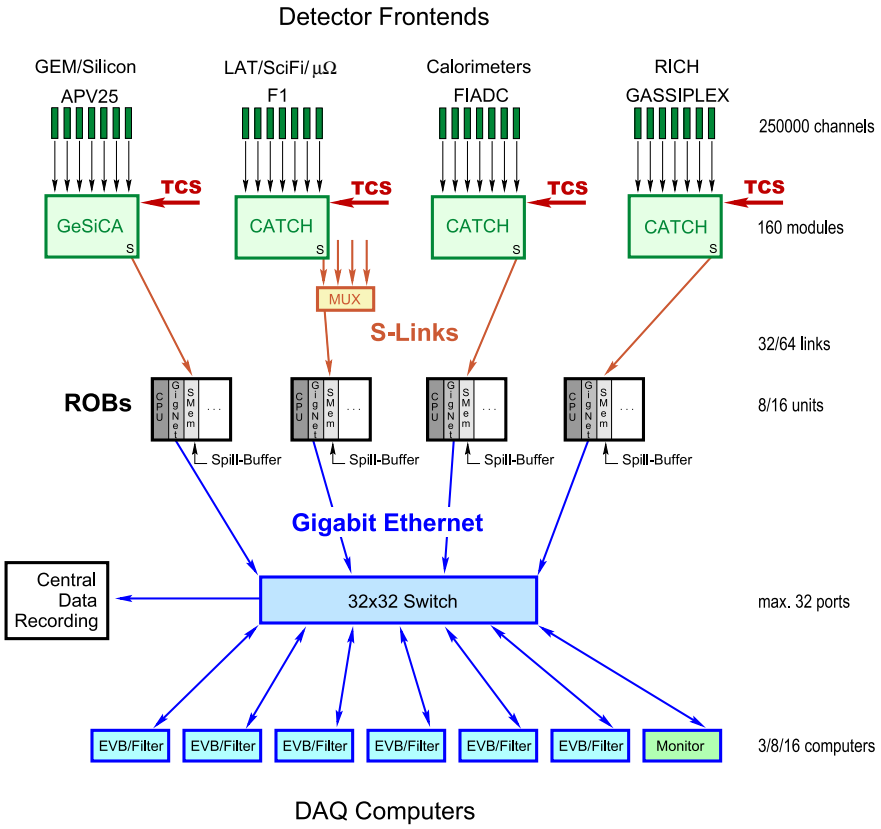



**Abbildung 3.1:** Schematischer Aufbau der Datenerfassung bei COMPASS (aus [20])

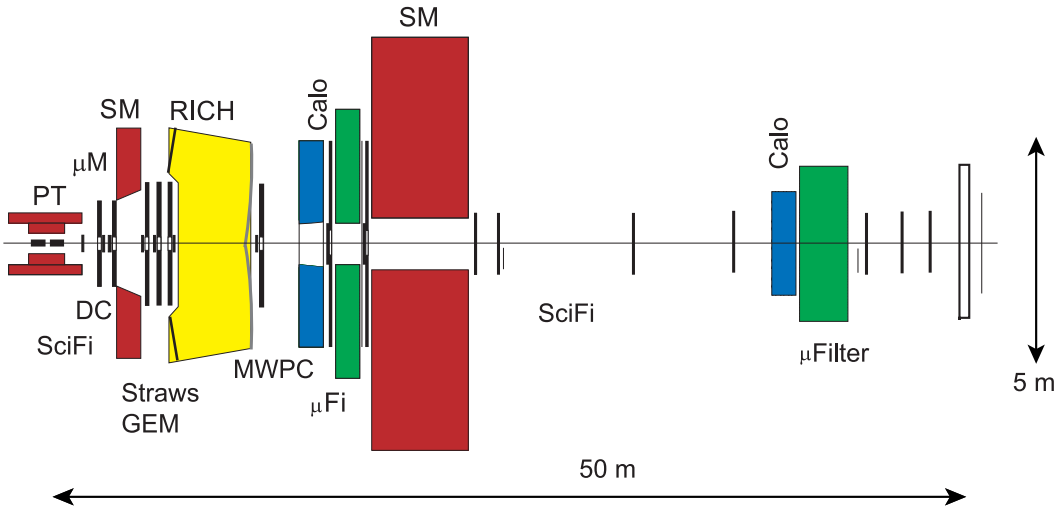



**Abbildung 3.2:** Der COMPASS-Detektor

dazu mit ADCs-Chips (Analog to Digital Convertern) ausgestattet.

### 3.1.1 Die Front-End-Karte für Straws

Ein Beispiel für eine Front-End-Karte ist die ebenfalls in Freiburg entwickelte Karte für Driftröhrchen-Detektoren (engl. “straws”). Auf ihr befinden sich acht ASD8b Vorverstärkerchips und acht $\mathcal{F}1$-TDC-Chips, siehe Abbildung 3.3. Mit dem $\mathcal{F}1$ können die Driftzeiten mit einer sehr guten Zeitauflösung von bis zu 130 ps gemessen werden. Die Front-End-Karte besitzt 64 Eingangskanäle für Ladungsimpulse aus den Driftröhrchen. Der ASD8b-Chip dient als Verstärker und Diskriminator, der die Ladungspulse in für den $\mathcal{F}1$ verarbeitbare differentielle Sigale umsetzt. Die Schwellen der ASD8b-Chips können für jeden Kanal einzeln über eine serielle Verbindung vom CATCH aus gesetzt werden, wobei der HP2400 Optokoppler [12] als Empfangsbaustein dient. Da der ASD8b ein rein analoges Bauteil ist, müssen die Werte für die Schwellenspannung erst mit den Digital-Analogwandlern AD8842 von den digitalen Steuer-Ausgängen des F1 zu analogen Signalen umgewandelt werden. Die korrekte Funktion dieser Initialisierungsprozedur sowie die Implementierung der seriellen Schnittstelle waren wichtige Teilaspekte beim Entwurf des XCONTROL-Projekts.

Zur Übertragung der digitalisierten Mess-Daten sowie zum Empfang der experimentweiten Clock, der Trigger-Signale sowie der Schwellwert-Daten vom CATCH zur Front-End-Karte wird ein achtadriges Twisted-Pair-Kabel verwendet. Jede Front-End-Karte besitzt eine eindeutige Identifikationsnummer (ID) sowie eine vom Steckplatz am Detektor abhängige Nummer (“geographical ID”). Diese Nummern werden von der Karte nach dem Einschalten gesendet und solange wiederholt, bis die Karte initialisiert wird und somit bereit zum Digitalisieren der Detektordaten ist. Dieses Vorgehen vereinfacht die Installation der Karten und die Verkabelung, da nicht mehr auf die Reihenfolge der Kabelverbindungen geachtet werden muss.

## 3.2 HOTLink und HOTFibre

Zur Übertragung der digitalisierten Daten von der Front-End-Karte zur nächst höheren Ebene der Datenerfassung wird eine serielle Hochgeschwindigkeitsdatenleitung mit dem Namen “HOTLink” benutzt, die auf dem fibre channel Standard (ANSI X3.230) basiert.

Das Übertragungsmedium kann dabei entweder ein Paar verdrillter Kupferleitungen (“twisted pair”) sein, es kann aber auch Glasfaser verwendet werden. Der Handelsname der für das Übertragungsprotokoll verwendeten Bausteine ist HOTLink [13]. Dieser Name hat sich bei COMPASS auch für das Übertragungsprotokoll selbst sowie für die auf dem CATCH verwendeten Tochter-Karten zum Empfang der Daten eingebürgert. Die Karten, die eine Glasfaser-Verbindung verwenden, heißen “HOTFibre”-Karten und sind mit den HOTLink-Karten bis auf die Glasfaser-Transceiver, der statt der Buchsen für Kupferkabel verwendet wird, identisch. Die HOTFibre-Karte wird bis jetzt nur zum Auslesen der RICH-Frontendboards eingesetzt. Die genaue Art beider Kabeltypen und einige Details zur Funktionsweise und zum Test der HOTLink- bzw. HOTFibre-Karten werden in [14] besprochen.

Der Datenfluss vom Detektor bis zum CATCH verläuft wie in Abb. 3.4 gezeigt.

Die Front-End-Boards folgender bei COMPASS verwendeter Detektoren werden über HOTLink ausgelesen:

- Straws (TDC), siehe Abschnitt 3.1.1

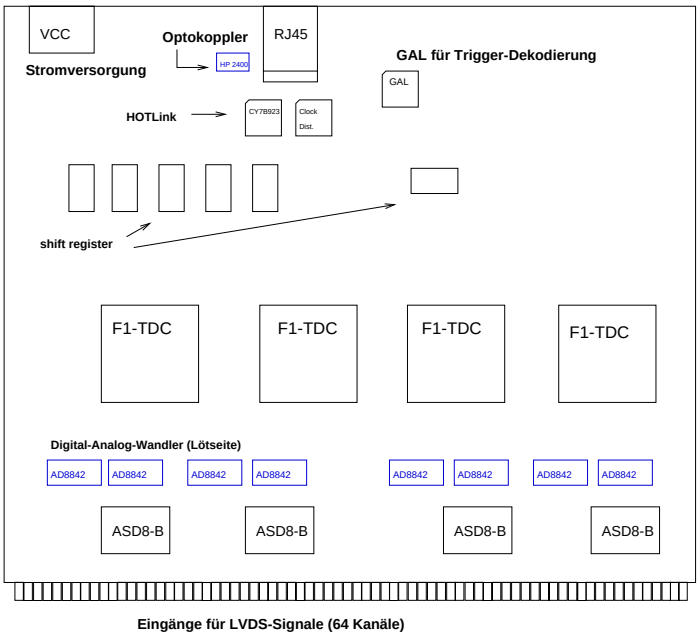



**Abbildung 3.3:** Skizze einer Front-End-Karte für Straws. Vier der acht $\mathcal{F}1$- und ASD8b-Chips befinden sich auf der Lötseite der Platine und sind hier nicht sichtbar. Die Digital-Analogwandler und der Opltokoppler befinden sich ebenfalls auf der Lötseite und sind hell dargestellt

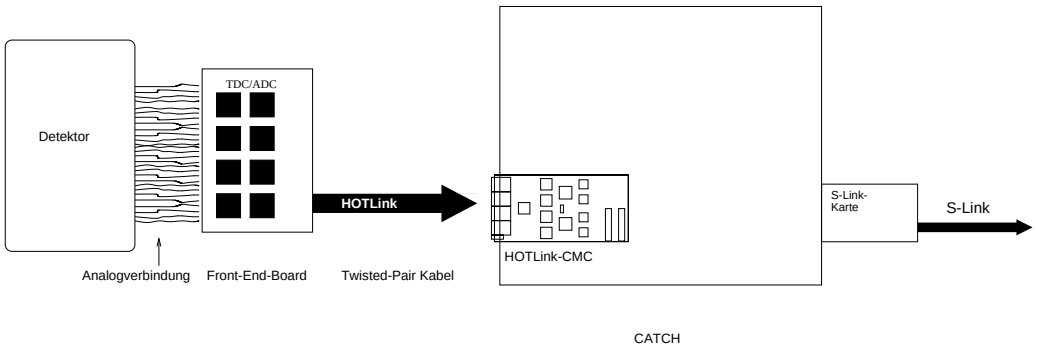



**Abbildung 3.4:** Datenübertragung vom Detektor via HOTLink

- $\mu$-Wall2 (TDC) Freiburg/Protvino
- RICH (“Bora”-Boards, ADC), Triest
- MWPC (TDC), entwickelt in Turin
- Driftkammern (TDC), entwickelt in Saclay
- Micromegas (TDC), entwickelt in Saclay
- Kalorimeter (ADC), entwickelt in München
- $\mu$-Wall1 (TDC) entwickelt in Dubna und Turin
- ein Zähler FE-Board (scaler) entwickelt in München (ADC-Datenformat)

## 3.3 CATCH

Die nächste Stufe der Datenerfassung ist das “CATCH”-Modul, wo die Rohdaten aus den Frontend-Karten zwischengespeichert und vorsortiert werden. Dafür ist natürlich eine vergleichsweise komplexe und zugleich schnelle Elektronik erforderlich. Bis zu 16 dieser CATCH-Module befinden sich einem Crate[1], welcher jeweils einen Prozessrechner besitzt, der an ein Netzwerk angeschlossen ist. Der Rechner kann zur Steuerung, Überwachung oder zur Programmierung der CATCH-Module benutzt werden. Insgesamt werden ca. 160 CATCH-Module verwendet. Auf das CATCH-Modul, seinen Aufbau und seine Funktion wird ausführlich in Kapitel 4 eingegangen.

An dieser Stelle soll jedoch bereits der modulare Aufbau des CATCH angesprochen werden, durch den es möglich ist, CATCH für fast alle Detektortypen einzusetzen: Auf der eigentlichen CATCH-Platine gibt es nämlich vier Steckplätze für Tochterplatinen (sogenannte Mezzanine[2]-Karten), die passend zum Detektortyp verwendet werden können. Diese Karten folgen dem sogenannten Common-Mezzanine-Card Standard [15] und werden im folgenden als CMC-Karten bezeichnet.

### 3.3.1 Mezzanine-Karten

Bisher sind vier Arten von CMC-Karten für das CATCH entwickelt worden:

1. HOTLink-CMC zum Empfang von Daten von den Front-End-Karten über Kupferkabel.
2. HOTFibre-CMC: ähnlich der HOTLink-CMC, nur wird als Medium Glasfaser statt Kupferkabel eingesetzt.
3. $\mathcal{F}$1-TDC-CMC: Sie dienen zur Zeitmessung bei der Auslese der Triggerhodoskope und der Detektoren aus szintillierenden Fasern.
4. Scaler-CMC: ein 250 MHz schneller Zähler für 32 Kanäle mit totzeitfreier Auslese [16].

[1] dt.: Überrahmen, wörtlich: Kiste
[2] mezzanine engl.: Zwischengeschoß

| Detektor | Zahl der CATCH-Module | Mezzanine Kartentyp |
|---|---|---|
| Straw-Driftröhren und Driftkammern | 24 + 2 | HOTLink |
| Vieldraht-Proportionalkammern (MWPC) | 14 | HOTLink |
| Myon-Walls | 7 | HOTLink |
| Sci-Fi | 54 | TDC-CMC |
| RICH | 16 | HOTFibre |
| Kalorimeter | 10 | HOTLink (ADC) |
| Beam Momentum Stations | 4 | TDC-CMC |
| Micro-Megas ($\mu$M) | 13 | HOTLink |

**Tabelle 3.1:** Aufteilung der benötigten CATCH-Module und CMC-Karten auf die Komponenten des Detektors

Die beiden letztgenannten CMC-Typen werden unten genauer erläutert. Die Verwendungszwecke der verschiedenen Karten am Detektor sind in Tabelle 3.1 aufgelistet. Alle CMC-Karten haben eine einheitliche Schnittstelle zum CATCH. Die verarbeiteten Daten werden auf den CMC's in FIFOs zwischengespeichert, von wo sie vom CATCH ausgelesen werden können. Dieses modulare Konzept erlaubt es, später zusätzliche Arten von Karten in das Experiment zu integrieren oder existierende Designs zu verbessern. Die Schnittstellen und Datenformate für die CMC-Karten für COMPASS sind in [17] festgelegt.

### TDC-CMC

Einige Detektoren werden nicht über HOTLink, sondern direkt mit einer auf dem CATCH montierten Tochterplatine ausgelesen, die ebenfalls mit acht $\mathcal{F}$1-TDC-Chips bestückt ist. Die Detektoren liefern Impulse im LVDS-Format, die über Flachbandkabel an die TDC-Karten gegeben werden (siehe Abb. 3.5). Die Detektoren, die auf diese Weise ausgelesen werden, sind:

- Hodoskope
- Szintillierende Fasern (SciFi) (Deutschland, Japan)
- Beam momentum stations
- die Referenzzeit (Trigger-Zeit) wird ebenfalls mit TDC-CMC gemessen.

### Scaler-CMC

Zur Messung der Strahlintensität und des zeitlichen Strahlprofils existiert ein schneller Zähler [16], der an szintillierende Fasern oder an die Hodoskope angeschlossen werden kann. Er ist als Mezzanine-Karte für das CATCH-Modul ausgelegt und besitzt 32 LVDS-Eingänge.

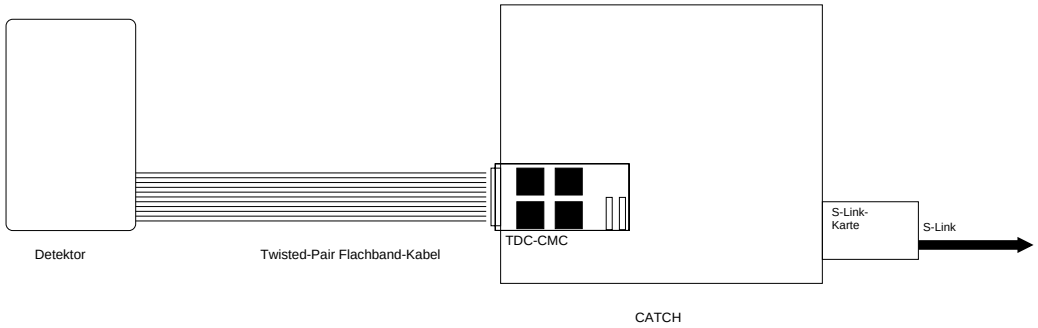



**Abbildung 3.5:** Datenübertragung vom Detektor an die TDC-CMCs

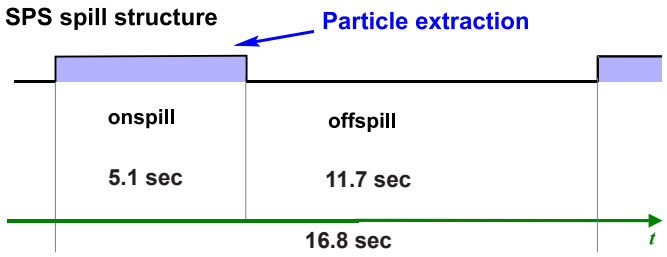



**Abbildung 3.6:** Struktur eines SPS-Spills

## 3.4 Zwischenspeicherung und Filterung der Daten

### 3.4.1 S-Link

Die sortierten und vorgefilterten Daten werden vom CATCH zu einem Read-Out-Buffer (ROB) übertragen. Dafür wird der am CERN entwickelte S-Link-Standard [18] verwendet. Dabei handelt es sich um eine schnelle Glasfaser-Verbindung, die zunächst einen maximalen Datendurchsatz von 100 MB/s besaß und inzwischen auf 160 MB/s aufgerüstet werden konnte. Verwendet werden hierbei S-Link-Karten, die auf der Rückseite des Crates an die CATCH-Moduls aufgesteckt werden können. Seit kurzem existieren auch S-Link-Multiplexer, die Daten von vier verschiedenen CATCH-Modulen lesen und auf einen S-Link-Glasfaserstrang ausgeben können. Diese Multiplexer werden bei Detektoren eingesetzt, die relativ wenig Daten pro Zeiteinheit produzieren.

### 3.4.2 Read-Out-Buffer

Das Prinzip der Read-Out-Buffer (ROB) beruht darauf, dass das SPS eine Spillstruktur hat, die aus 5.1 Sekunden eigentlichem Strahl und 11.7 Sekunden Pause besteht, wie in Abb. 3.6 dargestellt ist. Die Puffer dienen dazu, die während des Strahls (“on spill”) angefallenen Daten zwischenzuspeichern, so

dass sie in der vergleichsweise langen Spillpause weiterverarbeitet werden können.

Bei der Hardware der Read-Out-Buffer handelt es sich um sechzehn handelsübliche PCs, in deren PCI[3]-Steckplätzen spezielle Karten, die Spillbuffer eingebaut sind. Sie besitzen einerseits eine Schnittstelle zur Aufnahme einer S-Link-Receiver-Karte, andererseits haben sie einen Steckplatz für bis zu 512 MB SDRAM, so dass sie alle Daten eines Spills aufnehmen können. Bis zu vier dieser Spillbuffer-Karten können in einem PC untergebracht werden. Die PCs sind mit Gigabit-Ethernet[4]-Netzwerkkarten ausgestattet. In der Spillpause werden die Spillbuffer über den PCI-Bus ausgelesen und auf das Gigabit-Ethernet weitergegeben. Die Spillbuffer-Karten besitzen ein 32-Bit PCI-Bus-Interface, das auf 133 MB/s eingeschränkt ist. Für das nächste Jahr geplante Versionen sollen mehr RAM aufnehmen können und ein 64-Bit PCI-Interface besitzen, das den maximalen Datendurchsatz erhöht. Als Software wird auf den Read-Out-Buffer-PCs das Betriebssystem Linux und das DATE-System der Alice Kollaboration [19] eingesetzt. Letzteres sammelt die Daten der Spill-Buffer und stellt sie zur Weiterverarbeitung über das Netzwerk zur Verfügung.

### 3.4.3 Computer

Die Daten aus den Read-Out Buffern werden dann über das Netzwerk zu den sogenannten Eventbuildern weitergeleitet. Dies sind Rechner mit hoher Rechenleistung und I/O-Bandbreite, die die Daten der unterschiedlichen Teile des Detektorsystems zu kompletten Ereignissen zusammenfügen. Die Daten werden ereignisweise auf mehrere Rechner verteilt, wobei immer ein Rechner mit der Bearbeitung aller Daten eines Events beschäftigt ist. Die Rechner verfügen prinzipiell auch über genug Rechenleistung, um die Daten zu filtern, d.h., unsinnige Ereignisse zu verwerfen. Die Daten eines Runs werden zunächst auf lokalen Festplatten der Eventbuilder gespeichert und dann über das Netzwerk an die zentrale Datenerfassung weitergeleitet. Als Hardware werden Dual Pentium III Server PCs unter dem Betriebssystem Linux verwendet.

### 3.4.4 Netzwerk und Data Storage

Die Datenübertragung im Read-Out-Buffer/Eventbuilder-Netzwerk erfolgt über Gigabit-Ethernet. Im Moment werden zwei Typen von Gigabit-Ethernet-Switches der Firma 3COM eingesetzt: Zur Verbindung der DAQ-Computer untereinander wurden im Jahr 2001 zwei SuperStack-4900 Switches (1000 BaseT) benutzt, die über Glasfaser-Uplink-Ports (1000 BaseSX) verfügen. Auf der Backend-Seite verbindet ein SuperStack-9300-Switch (1000 BaseSX) die beiden 4900er-Switches, dasselbe Gerät wird auch zur Verbindung zum Rechenzentrum eingesetzt.

Die vorgefilterten Daten werden über eine 1000Base LX Glasfaser-Verbindung zum Rechenzentrum des CERN geschickt und dort einerseits archiviert und andererseits in einer föderalistisch organisierten Datenbank vorgehalten, wo sie online zugänglich sind.

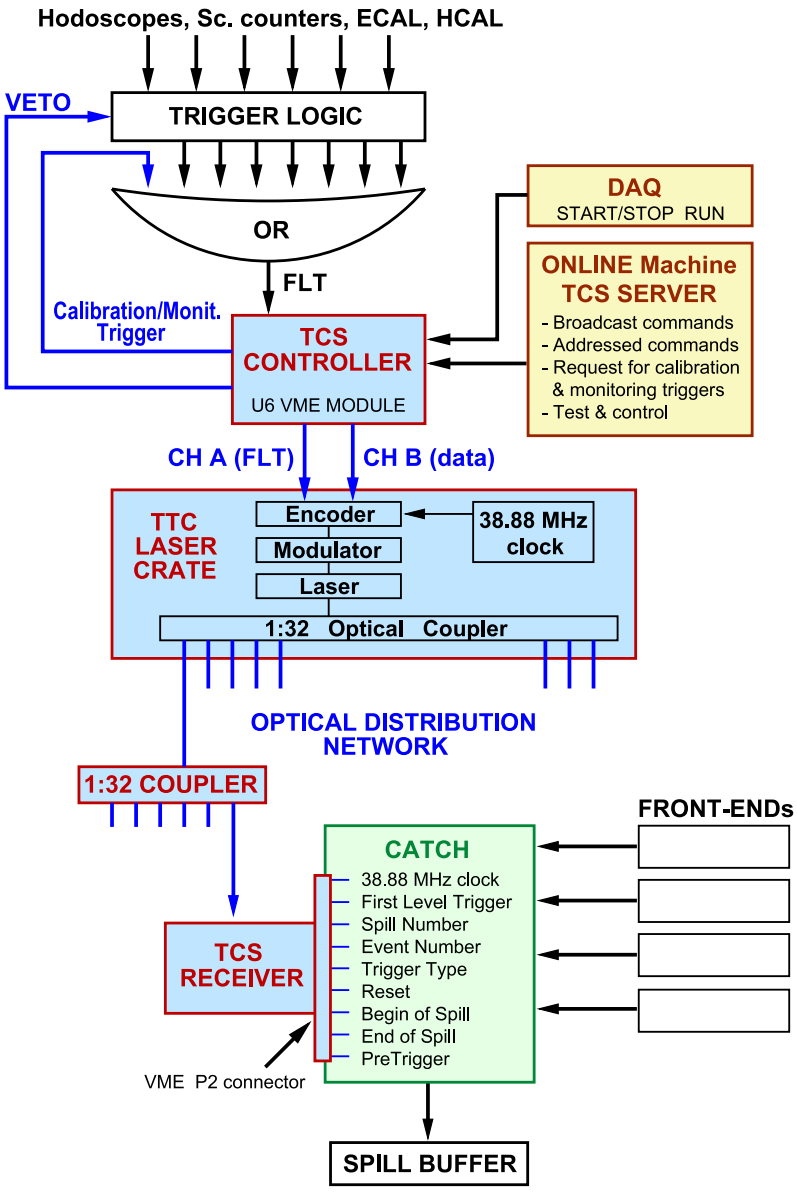



**Abbildung 3.7:** Aufbau des Trigger-Systems (aus [20])

## 3.5 Das Trigger-Control-System

Das Triggersystem bei COMPASS basiert auf zwei Paaren von Szintillationszählerhodoskopen 35 und 50 m hinter dem Target, die zusammen mit den Kalorimetern sowie einer schnellen Matrixkoinzidenz die physikalisch bedeutsamen Trigger (sogenannte First-Level-Trigger) definieren. Dafür ist eine ausgefeilte und sehr schnelle Elektronik notwendig - die eigentliche Triggerlogik [10]. Für die Verteilung der physikalischen Trigger am gesamten Experiment ist das Trigger-Control-System (TCS) zuständig [20]. Das Trigger-Control-System besteht aus drei Einheiten (vgl. Abb 3.7). :

- TCS Controller
- Laser Crate
- TCS Receiver

Die First-Level-Trigger, die von der Trigger-Logik erzeugt wurden, werden in den TCS-Controller eingespeist. Für jeden solchen Trigger erzeugt der TCS-Controller eine Spillnummer, Eventnummer und den Trigger-Typ, die zusammen den Event-Header bilden. Für die Verteilung dieser Information über das serielle optische Netzwerk werden zwei unabhängige Kanäle A und B mit einer Bandbreite von jeweils 40 MBit/s verwendet. Die First-Level-Trigger werden auf Kanal A übertragen, die anderen Daten auf Kanal B. Die beiden Ausgänge des TCS-Controllers gehen in das Laser Crate, das die Umsetzung in einen Bit-Stream auf dem optischen Glasfasernetzwerk vornimmt. Die TCS Receiver extrahieren die Trigger-Daten sowie die experimentweite Clock aus der optischen Verbindung. Die Clock kann mit einem Jitter von unter 45 ps übertragen werden. Die Receiver befinden sich an der Backplane der VME-Crates und leiten die Informationen an die CATCH-Module weiter. Die TCS-Receiver sind vom TCS-Controller aus auch mittels spezieller Broadcasts konfigurierbar.

[3] Peripheral Component Interconnect — Standardschnittstelle für Geräte in einem PC

[4] Ethernet ist der am weitesten verbreitete Standard für lokale Netzwerke (LANs). Seit Juni 1998 ist Gigabit-Ethernet über Glasfaser als IEEE 802.3z standardisiert

# Kapitel 4

# CATCH

## 4.1 CATCH Überblick

Beim CATCH-Modul handelt es sich um einen zentralen Baustein bei der Datenauslese von COMPASS. CATCH steht für COMPASS Accumulate, Transfer and Control Hardware.
Seine Hauptaufgaben sind:

- Initialisierung der Frontend-Elektronik
- Verteilung der experimentweiten Clock und des Triggersignals an die Front-End-Karten
- Auslesen der Frontend-Elektronik
- Bearbeitung und Vorsortieren der gelesenen Daten
- Weiterleitung der Daten via S-Link

Die Hardware selbst ist als 9U-Board[1] mit VME-Bus konzipiert, von denen bis zu 16 in einem Crate untergebracht werden können. Einen schematischen Überblick zeigt Abb. 4.1.

Um maximale Flexibilität beim Einsatz am Experiment zu haben, ist das CATCH-Modul modular aufgebaut. Auf der Eingangsseite kann das Board vier Tochterplatinen (auch mezzanine-Platinen genannt) aufnehmen. Ihr Format ist nach dem CMC-Standard (common mezzanine card [15]) genormt, was z.B. ihre Größe und die Art der Steckverbindung zum CATCH festlegt. Die wichtigsten Details der Mezzanine-Karten wurden bereits in Abschnitt 3.3.1 beschrieben. Auf der Einschubseite des Boards befinden sich drei Steckplätze, die zur Kommunikation mit der Außenwelt dienen (Abb. 4.1). Sie werden teilweise zur Kommunikation mit dem im Crate befindlichen VME-Rechner benutzt, dienen aber auch zum Empfang der Triggerdaten über den TCS-Receiver und zur Verbindung mit dem S-Link, der zur Weiterleitung der von den Mezzanine-Karten empfangenen Daten dient.

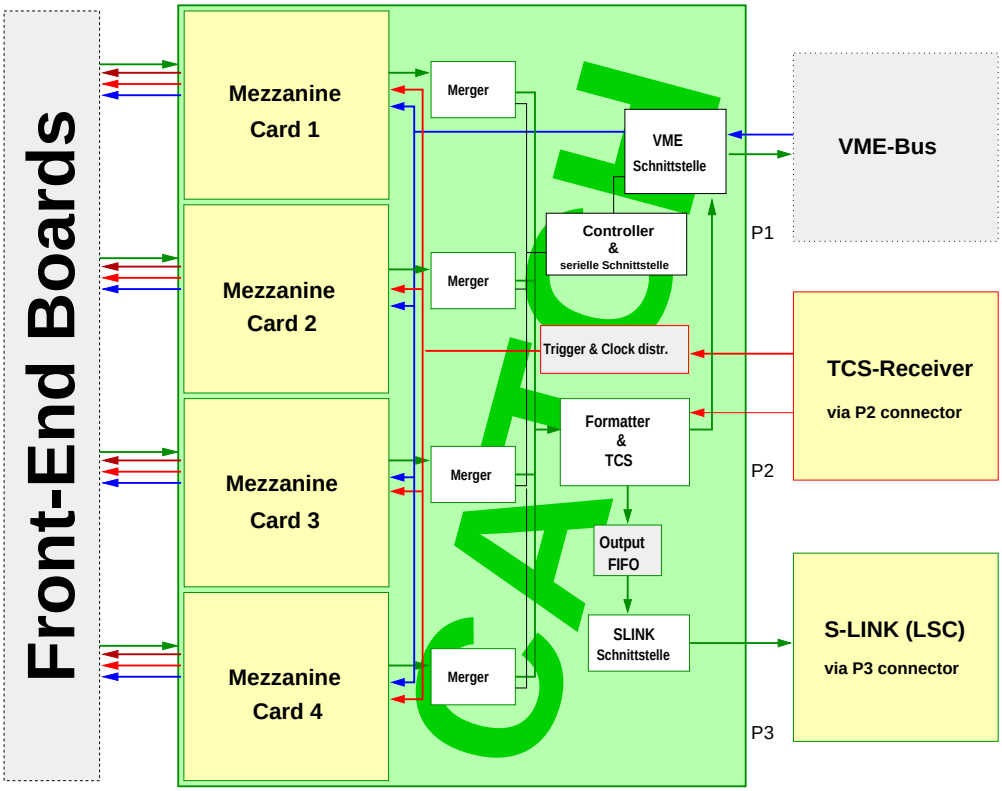



**Abbildung 4.1:** Schematischer Aufbau eines CATCH Moduls

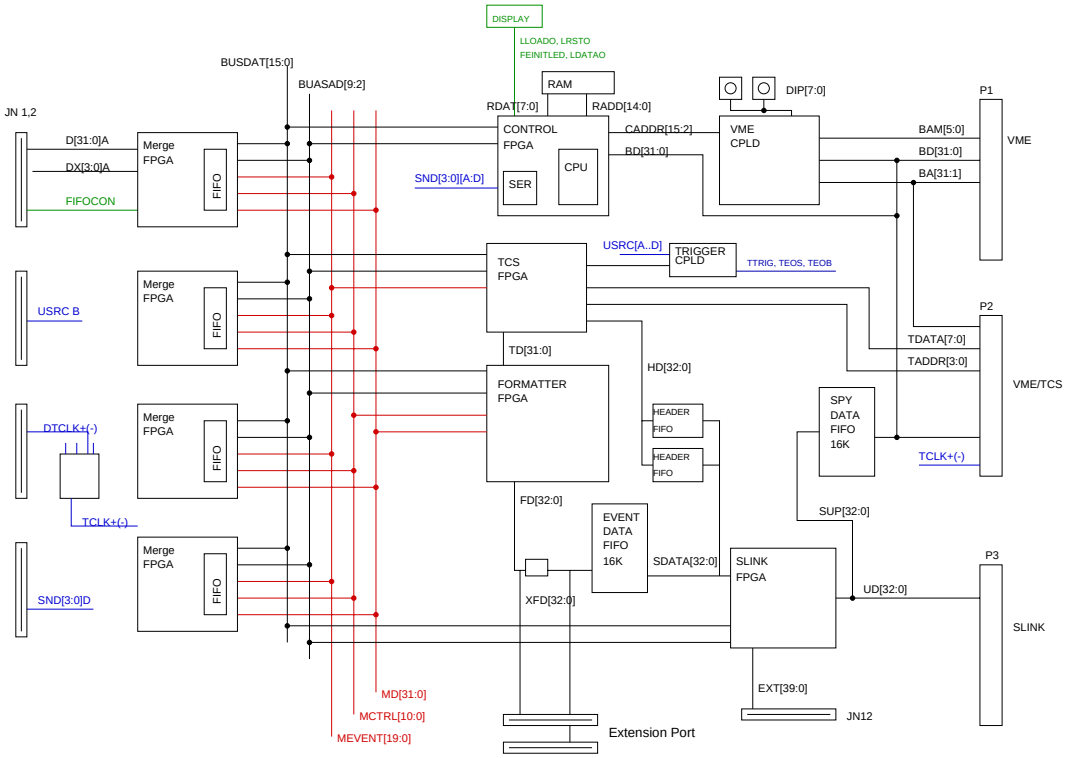



**Abbildung 4.2:** Vereinfachtes Schaltbild eines CATCH-Moduls

## 4.2 Funktionen des CATCH-Moduls

In Abbildung 4.2 sind die wichtigsten Bauteile und Bussysteme des CATCH dargestellt. Die hervorstechendsten Bauteile sind die acht FPGAs. Die vier Merge-FPGAs lesen die Messdaten aus FIFOs der CMC-Karten. Über spezielle Steuerleitungen (`FIFOCON`) können sie diese auch konfigurieren. Über das insgesamt 43 Bit breite Bussystem(`MD[31:0]` und `MCTRL[10:0]`) kann der Formatter-FPGA die Merger reihum auslesen und die Daten formatieren.

Zwischen den FPGAs existiert weiterer Bus mit 16 Bit Daten- und 8 Bit Adressbreite, über den Register in den FPGAs zur ihrer Konfiguration bzw. zur Abfrage ihres Status gelesen bzw. geschrieben werden können (`BUSDAT[15:0], BUSAD[9:2]`).

Zwischen Formatter-FPGA und Event Data-Fifo befindet sich noch ein Extension-Port über den in Zukunft weitere Bausteine in den Datenweg eingeschleift werden können. Geplant ist ein Modul, das die Daten eines Spills in Echtzeit komprimiert und in einem RAM-Baustein zwischenpuffert, um den Flaschenhals S-Link (beschränkt auf 160 MB/s) zu umgehen.

Auf der Rückseite des CATCH-Moduls befinden sich die Stecker P1, P2 und P3. Über P1 und P2 wird die Schnittstelle zum VME-Bus realisiert, die Verbindung zum Trigger-Control-System erfolgt ebenfalls über P2. Die Schnittstelle zum S-Link benutzt den Stecker P3.

[1] das ist eine standardisierte Grössenangabe für Eurocard-Module - 9U entsprechen $367 \times 400\text{mm}^2$

### 4.2.1 Trigger und Clock

An die Mezzanine-Karten (und von dort aus an die Front-End-Karten) kann entweder die auf dem CATCH durch einen lokalen Oszillator erzeugte Clock oder die vom TCS-Receiver erhaltene experimentweite 38.88 MHz-Clock weitergegeben werden. Die Clocksignale werden im differentiellen PECL-Standard über die Leitungen `DTCLK`$\pm$ an die Mezzanine-Karten verteilt, wozu ein spezieller Clock-Verteilerbaustein dient.
Auch die Triggersignale werden an die Mezzanine-Karten übertragen. Hierzu dient der Trigger-CPLD. Der Trigger-CPLD erhält die Signale für den Beginn und das Ende eines Bursts, sowie die Triggersignale vom TCS-Receiver. Er kodiert diese Signale zeitlich, d.h. unterschiedliche Signallängen haben unterschiedliche Bedeutung. Ein normales Triggersignal hat beispielsweise eine Länge von 25 ns, der Beginn eines Spills 50 ns, das Resetsignal für die Front-End-Karten eine Länge von 100 ns. Diese Signale werden an die Mezzanine-Karten weitergereicht, die sie wiederum - unter Benutztung eines geeigneten Signalstandards für die Datenübertragung - an die Front-End-Elektronik weitergeben. Zusätzlich gibt der Trigger-CPLD die Trigger-, Begin-Of-Spill-, End-Of-Spill- und Reset-Signale auch unkodiert auf dedizierten Leitungen an die Mezzanine-Karten weiter.

### 4.2.2 Serielle Datenleitung

Die Front-End-Boards senden nach dem Einschalten über die Datenleitung ihre Seriennummer und ihre geographische Lage am Detektor.
Das CATCH erkennt diese Adressen und kann aus einer Datenbank Daten zur Konfiguration des entsprechenden Front-End-Boards holen. Dies sind z.B. Spannungen für Diskriminatorschwellen oder die gewählten Betriebsmodi für die $\mathcal{F}$1-TDCs. Zur Initialisierung werden serielle Verbindungen mit 10 Mbit/s benutzt. Dies sind die Leitungen `SNDA[3:0]` bis `SNDD[3:0]`, so dass jeder Port einzeln initialisiert werden kann. Der Control-FPGA übernimmt die Ansteuerung dieser seriellen Schnittstelle. Übertragen werden Worte zu je 24 Bit, das Protokoll ist an die weitverbreitete UART-Schnittstelle angelehnt. Genauer wird auf diese Schnittstelle in Kapitel 6.7 eingegangen.

### 4.2.3 VME-Interface

Das CATCH-Modul verfügt über eine VME-Schnittstelle, über die man mit einem in das Crate eingebauten Rechner Zugriff auf das Board hat. Es können so über den VME-Bus Daten an das CATCH gesendet werden. Man kann zur Fehlersuche aber auch lesend über diesen Bus auf das CATCH zugreifen und so interne Register lesen oder schreiben oder sich Rohdaten ansehen.

Der VME-CPLD dient als VME-Slave gegenüber des Busmasters (CPU im VME-Rechner). Er verarbeitet einen Teil der VME-Anfragen, wie zum Beispiel das globale Reset des CATCH selbst und kann so auch die Programmierung der FPGAs steuern. Die anderen VME-Adressen werden an den Control-FPGA weitergeleitet.

Der VME-CPLD benutzt die Leitungen BD[31:0], BA[31:1] zur Kommunikation mit dem VME-BUS. Die Hardware-Adresse des CATCH auf dem Bus wird mit zwei 16-stelligen Schaltern eingestellt (Leitungen DIP[7:0]). Über den VME-CPLD kann man auch den sogenannten Spy-Buffer auslesen. Dieser ist quasi dem Datenausgang zum S-Link parallelgeschaltet und ermöglicht es, direkt vom Prozessrechner aus die aktuellen Daten einzusehen, ohne den Umweg über die Auslese mit S-Link machen zu müssen.

## 4.3 Funktionen der FPGAs

### 4.3.1 TCS-FPGA

Der TCS-FPGA erhält Daten vom TCS-Receiver mittels der Leitungen TDATA[7:0] und TADDR[3:0]. Diese Informationen werden mit MEVENT[19:0] an die Merger-FPGA weitergegeben, und mit TD[31:0] an den Formatter. Wichtig ist hierbei, dass die vom TCS-Receiver erhaltenen Trigger-Signale und Trigger-Informationen mit 38.88 MHz verarbeitet werden müssen, was die Basisfrequenz der eingesetzten Glasfaserverbindung ist. Der TCS-FPGA muss sicherstellen, dass die Daten korrekt an die anderen FPGAs weitergegeben werden, die mit der internen CATCH-Taktfrequenz von 40 MHz betrieben werden. Vom Formatter erhält der TCS-FPGA auch Daten zurück, die er benutzt, um Headerinformationen zu erzeugen, die im Header-FIFO zwischengespeichert werden. Die Header enthalten Informationen wie die Ereignisnummer, den Typ und die Anzahl der Daten im Ereignis usw. Der genaue Typ der Header ist in der COMPASS-Online-Data-Format-Note[21] definiert.

### 4.3.2 Merger-FPGA

Die Merger-FPGAs übernehmen das Auslesen der Mezzanine-Karten. Für jeden Steckplatz einer Mezzanine-Karte gibt es einen Merger, so dass insgesamt vier Merger-FPGAs auf dem CATCH vorhanden sind. Diese FPGAs erfüllen folgende Funktionen:

1. Erkennen der gesteckten Mezzanine-Karte
   - HOTLink-CMC
   - HOTFibre-CMC
   - Scaler-CMC
   - TDC-CMC
2. Die CMCs können Daten in zwei Formaten liefern: das ADC-Format, das jeweils vier Byte breite Datenworte benutzt, oder das TDC-Format, das 3 Byte breite Datenworte benutzt. Bei der Scaler-CMC und bei der HOTFibre-CMC liegen die Daten immer im ADC-Modus vor. Bei der TDC-CMC wird immer das TDC-Format benutzt, bei der HOTLink-CMC sind beide Formate möglich. Abhängig von der auf dem CATCH verwendeten CMC schaltet der Merger automatisch seinen Datenmodus um:
   - ADC-Modus (4 Byte Daten), wenn Scaler-CMC oder HOTFibre-CMC angeschlossen sind,
   - TDC-Modus (3 Byte Daten), wenn TDC-CMC oder HOTLink-CMC angeschlossen sind.

3. Warten auf Initialisierungsdaten von den Front-End-Boards. Bei einer HOTLink-CMC kann man den verwendeten Datenmodus (ADC oder TDC) erst aus den von den Front-End-Karten gesendeten Initialisierungsdaten erkennen. Sie werden vom Merger ausgewertet und ein eventuell nötiger Wechsel des Datenmodus wird vorgenommen, je nachdem welcher Typ von Front-End-Board erkannt wurde.

4. Handelt es sich um ein FE-Board, das sich nicht korrekt ausweisen kann, dann kann der Datenmodus und die Identifikationsnummer auch "von Hand" gesetzt werden, d.h. mit einem VME-Befehl. Geschieht dies nicht, dann ist es später nicht mehr möglich, die Herkunft der Daten festzustellen.

5. Vor dem eigentlichen Auslesen der Daten wird auf die Eventnummer vom TCS-System gewartet, die der TCS-FPGA zur Verfügung stellt. Sie kann durchaus eine Mikrosekunde später als die Daten eintreffen, da insgesamt 50 Bit Information vom TCS-Controller erzeugt und übertragen werden müssen, die der TCS-FPGA dann verarbeitet. Nach dem Abholen der Eventnummer durch den Merger wird eine Rückmeldung an den TCS-FPGA gegeben und mit dem Auslesen der eingangsseitigen FIFOs begonnen.

6. Datenauslese:
   Die Daten, die von den Front-End-Karten geschickt wurden bzw. auf den TDC oder Scaler CMCs erzeugt wurden, werden auf der Mezzanine-Karte in FIFOs zwischengespeichert. Auf den HOTLink und HOTFibre-CMCs gibt es FIFOs für jeden Port, so dass 4 unabhängige FIFOs vorhanden sind, die reihum ausgelesen werden. Die FIFOs werden mit der experimentweiten 38.88 MHz-Clock betrieben und mit der CATCH-internen Clock von zur Zeit 40 MHz ausgelesen. Der Merger kann die Daten aus dem FIFO im Prinzip mit jeder steigenden Clockflanke (alle 25 ns) auslesen. Zunächst wird ein Header erwartet, in dem die Eventnummer steht, die lokal auf dem FE-Board zu jedem empfangenen Triggerimpuls hochgezählt wurde. Diese Nummer wird vom Merger mit der vom TCS über den Bus `MEVENT[19:0]` erhaltenen Nummer verglichen. Sind die Nummern nicht identisch, so hat entweder das FE-Board Trigger zu viel gezählt (Glitch), Trigger verloren, oder das TCS-Systems hat versagt, so dass falsche Nummern an das CATCH übermittelt wurden. Die Datenpakete werden dann entsprechend mit einem Fehlerwort markiert, bevor sie ausgelesen werden.

7. Werden sehr viele Daten geliefert (zu lange Datenpakete, rauschende Triggerleitungen) füllt sich das FIFO auf der CMC. Der Merger erkennt dies und markiert die ausgelesenen Daten mit einem dafür vorgesehenen Fehlerwort. Dafür wird der Bus `MCTRL[10:0]` verwendet.

### 4.3.3 Formatter-FPGA

Der Formatter liest die Daten von den Mergern und bringt sie in das in [21] dokumentierte Format. Im Detail besitzt er folgende Funktionen:

- Auslesen der Merger-internen FIFOs bis zum Ende eines Events ("primary trailer"). Das Ende eines Events wird vom Formatter mit dem Hexadezimalwert `0xCFED1200` markiert.

- ADC-Daten (32 Bit) werden im wesentlichen unverändert gelassen. Bei TDC-Daten wird eine vierbittige Port-ID hinzugefügt.

- Wurde vom Merger ein Fehler erkannt, setzt er Bits auf dem Bus `MCTRL[10:0]` entsprechend. Dies wird vom Formatter ausgewertet: zu jedem Wort, bei dem diese Flags gesetzt waren, wird ein komplettes 32 Bit Fehlerwort erzeugt, das vor das eigentliche (fehlerhafte) Datenwort eingeschoben wird.
- Der Formatter erhält Informationen vom TCS-FPGA über die Leitungen `TD[31:0]`.
  Bei der Mitteilung "Skip-Event" werden die Merger-Fifos zwar ausgelesen, die Daten des Events jedoch verworfen.
  Beim First Event Of Run werden zusätzliche Headerworte erzeugt.
- Sparsified-Mode (für TDC-Daten): Jeder einzelne TDC-Chip auf den Front-End-Boards oder TDC-CMCs überträgt normalerweise zusätzliche Header- und Trailerworte innerhalb der für ein Ereignis festgelegten Datenstruktur. Diese werden im sogenannten Sparsified-Mode nicht weitergeleitet, da sie redundante Informationen enthalten, die zur späteren Rekonstruktion des Ereignisses nicht notwendig sind. Dies spart in den nachfolgenden Stufen der Ausleseelektronik eine erhebliche Datenmenge ein (bis ca. 25%), die erst gar nicht übertragen werden muss.

### 4.3.4 S-Link-FPGA

Der S-Link-FPGA überträgt zuerst den Header eines Events aus dem Header-FIFO und danach die Daten aus dem Daten-FIFO an die S-Slink-Schittstelle. Auch die Steuerung und Überwachung der S-Link-Karte und des Spy-Buffers gehört zu seinen Aufgaben.

## 4.4 Der Control-FPGA

Der achte FPGA auf dem CATCH heisst Control-FPGA. Wie bereits aus dem Namen ersichtlich, dient der Control-FPGA zur Steuerung und Überwachung des CATCH. Er hat - wie in Abb. 4.3 zu sehen - folgende Verbindungen zur Außenwelt:

- VME-Interface vom VME-CPLD, das sind die Adressleitungen `CADDR[15:2]` und die Datenleitungen `BD[31:0]` sowie Steuerleitungen (nicht abgebildet).
- Interface zu einem 32KB SRAM-Baustein, Adressleitungen sind hier `RADDR[14:0]`, Datenleitungen `RDAT[7:0]`.
- Ein serielles Interface zur Ausgabe der Konfigurationsdaten über die HOTLink-CMCs an die Font-End-Karten, dies sind die Leitungen `SNDA[3:0]` bis `SNDD[3:0]`.
- ein Bus-Interface, mit dem der Control FPGA mit den anderen sieben FPGAs kommunizieren kann. Dies sind `BUSDAT[15:0], BUSAD[9:2]` sowie Steuerleitungen (nicht abgebildet).
- Steuerleitungen für ein vierstelliges Dot-Matrix-Display, das an der Frontblende des CATCH montiert ist. Dafür sind die Leitungen `LLOADO, LDATAO, LRSTO` sowie `FEINITLED` vorgesehen.
- Leitungen zum Zurücksetzen der anderen FPGAs (`GRES-FPGA[7:0]`) sowie der Mezzanine-Karten (`RST[3:0]`).

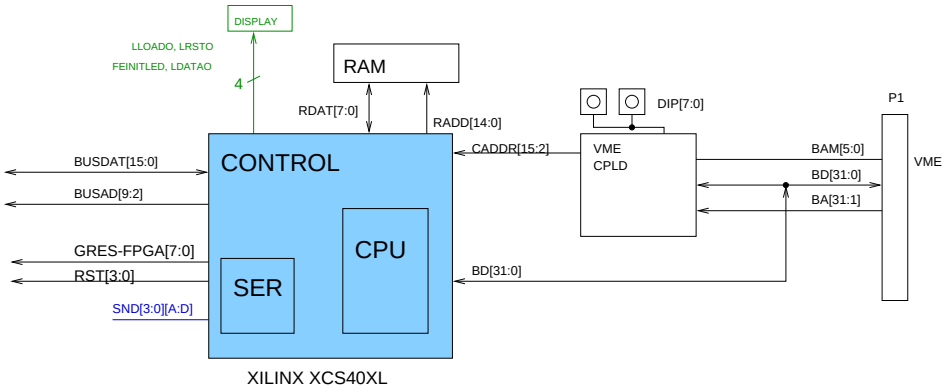



**Abbildung 4.3:** Schnittstellen des Control FPGAs

### 4.4.1 Steuerfunkionen des Control-FPGAs

Der Control-FPGA kann die anderen FPGAs steuern: Er ist in der Lage, Register, die diese zur Verfügung stellen, zu lesen. Er kann solche Register beschreiben und damit Funktionen in den FPGAs auslösen und jeden FPGA und jede CMC zurücksetzen (Reset). Die Kommunikation mit der Aussenwelt erfolgt hierbei ausschliesslich über den VME-Bus. Der VME-Bus-Master (der Rechner im Crate) schreibt dazu VME-Read- und Write-Befehle auf den Bus. Der VME-CPLD leitet einen Teil des Adressraums an den Control-FPGA weiter, der sie teilweise selbst verarbeitet (z.B. serielles Interface) und teilweise auf den internen FPGA-Bus (`BUSDAT`) weitergibt [22].

## 4.5 XCONTROL - Ein Mikroprozessor im FPGA

In bisherigen Versionen des Controller-Designs diente der Control-FPGA nur zum Weiterreichen der vom VME-CPLD empfangenen Daten an die anderen FPGAs sowie zum Auslesen ihrer internen Register. Die ausgelesenen Daten der Register wurden dann wieder über den VME-Bus ausgegeben.

Auch die Initialisierung der Front-End-Karten geschah mittels Kommandos auf dem VME-Rechner. Statt der Eingabe von Kommandozeilen auf dem VME-Rechner kann man auch die grafische Oberfläche `Catch-o-Matic` benutzten [23].

Ein Überwachungsprogramm für das CATCH hätte also auf dem VME-Rechner laufen müssen. Mehrere CATCH-Module sind dabei über den VME-Bus regelmässig abzufragen (Abb. 4.4).

#### Mikroprozessor im FPGA

Ein anderer Ansatz ist es, das CATCH “intelligent” zu machen und die Steuerfunktionen autonom ausführen zu lassen. Ein Steuerprogramm wird dabei einmal zu Anfang der Datennahme in das RAM des CATCH geladen. Das Programm übernimmt dann selbstständig die Überwachung der FPGAs sowie der CMC-Karten. Eine Möglichkeit dies zu erreichen, ist, diese Steuerfunktionen in Hardware zu implementieren, d.h. in einem speziellen Design für den Control-FPGA. Dieses muss dann im

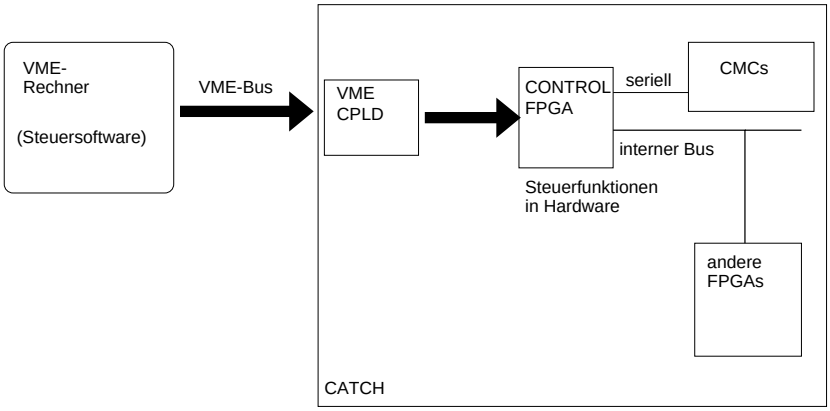



**Abbildung 4.4:** Bisherige Steuerung der CATCH-Funktionen

Schematics-Editor oder einer HDL erstellt werden. Diese Hardwarelösung hätte zwar den Vorteil sehr schnell zu sein, ist jedoch extrem unflexibel. Schon kleine Veränderungen der gestellten Anforderungen können größere Designänderungen im FPGA bewirken. Hinzu kommt, dass ein von einer Person erstelltes Design von einer anderen Person oft nur schwer zu durchschauen ist, was eine lange Einarbeitungsphase bedingt. Aufgrund des zu erwartenden langen Einsatzes der bisher beschriebenen Elektronik (geplante Dauer des Experiments: 10 Jahre) ist eine leichte Wartbarkeit und gute Anpassbarkeit einer solchen Steuerlogik essentiell. Die Idee war nun, in den Control-FPGA einen Mikroprozessor zu integrieren, der Zugriff auf alle auch dem bisherigen Controller-Design zur Verfügung stehenden Ressourcen hat (Abb. 4.5). Möglichst sollte für diesen auch ein Compiler für eine Hochsprache existieren. Ein Steuerprogramm kann dann recht einfach in dieser Hochsprache (z.B. C) verfasst werden und Anpassungen können vorgenommen werden, ohne sich auf die Hardware-Ebene der Register und Gatter eines FPGAs begeben zu müssen. Ein solcher Prozessor existierte bereits in Form des XSOC-Projekts, das im nächsten Kapitel beschrieben wird. Ergebnis all dieser Überlegungen war der verbesserte Controller `XCONTROL` für das CATCH.

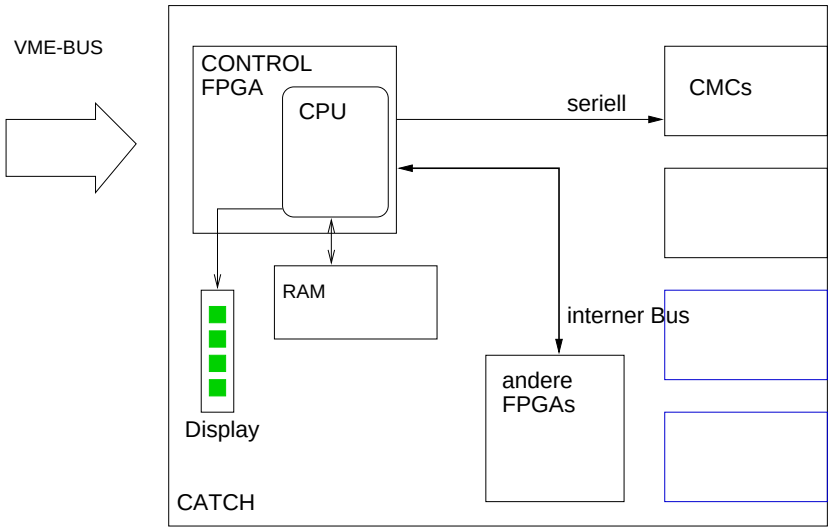



**Abbildung 4.5:** XCONTROL soll CATCH autonom steuern können

# Kapitel 5

# XSOC

Der bei der Verwirklichung des erweiterten Controllerdesigns verwendete Mikroprozessorkern ist der sogenannte `xr16`-Prozessor. Er ist Teil des `XSOC`-Projekts von Jan Gray, welches ein komplettes System-On-A-Chip für Lernzwecke auf einem Demonstrationboard für XilinX-FPGAs implementiert [24].
Obwohl andere, ebenfalls frei zugängliche Implementierungen von Mikroprozessoren verfügbar sind [25, 26, 27], wurde das `XSOC` Projekt gewählt, da es einen eigenen Compiler und Assembler schon mitbringt und klein, überschaubar und sehr gut dokumentiert ist. Außerdem wurde die Integration in das CATCH stark vereinfacht, da Jan Gray das Projekt für einen FPGA entwickelte, der sehr ähnlich zu den im CATCH verwendeten ist (siehe Kapitel 7 für weitere Details).

In diesem Kapitel werden zunächst einige grundlegende Konzepte moderner Mikroprozessoren erläutert. Anschließend wird der dem `xr16`-Prozessor zugrundeliegende Befehlssatz (engl: Instruction Set Architecture, ISA) sowie einige Implementierungsdetails besprochen. Mit recht bescheidenen Mitteln ist es möglich, die gundlegende Funktionalität der Programmiersprache C umzusetzen. Der zu diesem Zweck verwendete Befehlssatz ist stark an den der MIPS-Prozessorfamilie angelehnt, aber von 32 Bit auf 16 Bit breite Instruktionen eingeschränkt.

## 5.1 Prinzipieller Aufbau eines RISC-Prozessors

Ein klassisches Computersystem besteht aus einer zentralen Recheneinheit, CPU, oft auch MPU (Micro-Processor-Unit) genannt, einem Arbeitsspeicher (RAM) für Daten und Instruktionen sowie verschiedenen Ein- und Ausgabeschnittstellen. Den Prozessor selbst kann man in Datenpfad und Control-Unit unterteilen, so dass insgesamt ein Bild wie in Abb. 5.1 entsteht.

### 5.1.1 Datenpfad (Data-Path)

Im sogenannten Datenpfad (Data-Path) werden die eigentlichen arithmetischen Operationen ausgeführt. Er besteht aus Registern, die die zu bearbeitenden Daten (Variablen etc.) enthalten und Logik, die die Register untereinander verknüpft und so Rechenoperationen ausführt. Die Ergebnisse der Operation werden wieder in Registern gespeichert.

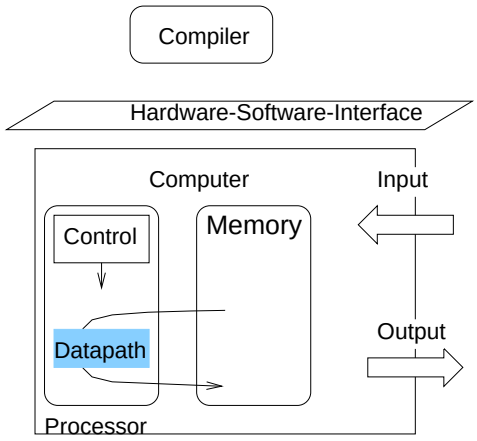



**Abbildung 5.1:** Die fünf klassischen Komponenten eines Computers: Input, Output, Memory, Data-Path und Control

### Register

Register sind interne Speicherstellen der CPU, in denen Daten enthalten sind, die unmittelbar von der CPU berarbeitet werden können. Sie sind dem Programmierer in Maschinensprache (Assembler) direkt zugänglich. In einer Programmiersprache wie C werden die Variablen darauf umgesetzt. Je nach Typ eines Prozessors kann die Anzahl der Register sehr unterschiedlich sein. Bei alten Rechnerarchitekturen gibt es oft nur ein Register (oft Akkumulator genannt), beim Intel i386 sind es acht Register und beim Intel Itanium 128 Integer und 128 Gleitkommaregister. Die Breite der Register in Bit wird im Allgemeinen als Bezeichnung für die Generation des Prozessors verwendet (z.B. ist der `xr16` ein 16-Bit Prozessor).

### ALU

Die Arithmetik-Logik-Einheit (ALU) führt die eigentlichen Rechenoperationen durch. Die Operationen werden auf die Inhalte der Register angewandt und das Ergebnis wieder in einem Register abgespeichert. Bei manchen Prozessortypen können Daten auch direkt vom RAM gelesen, von der ALU verarbeitet und in das RAM geschrieben werden. Die meisten modernen Prozessortypen besitzen verschiedene spezialisierte Arten von Recheneinheiten, z.B. eine ALU für Ganzzahloperationen, eine Floating-Point-Unit (FPU) für Gleitkommaoperationen oder Vektoreinheiten für Operationen auf überbreiten Registern. Oft sind auch mehrere Recheneinheiten gleichen Typs vorhanden, so dass Instruktionen parallel ausgeführt werden können (superskalarer Prozessor).

### Programmzähler (Program-Counter)

Der Programmzähler (Program-Counter) verweist auf die Speicherstelle, aus der die nächste abzuarbeitende Instruktion geholt werden muss. Ist die vorangegangene Instruktion ein normaler Befehl, so wird der Program Counter einfach inkrementiert. Handelte es sich jedoch um einen Verzweigungs-

| Register | Verwendung |
|---|---|
| r0 | immer Null |
| r1 | reserviert für Assembler |
| r2 | function return value |
| r3-r5 | Funktionsargument |
| r6-r9 | temporär |
| r10-r12 | Variablen-Register |
| r13 | stack pointer (sp) |
| r14 | interrupt return address |
| r15 | return address |

**Tabelle 5.1:** Die im xr16-Prozessor verfügbaren Register

oder Sprungbefehl, so zeigt der Program-Counter nun auf die durch den Befehl spezifizierte Speicherstelle. Bei einem Reset der CPU wird der Program-Counter auf Null gesetzt. Bei Interrupts gibt es bestimmte Zieladressen, an denen spezieller Programmcode für die Behandlung des Interrupts vorgesehen ist.

### 5.1.2 Kontrolleinheit (Control-Unit)

Die Control-Unit fordert die Instruktionen aus dem RAM an und dekodiert sie. Je nachdem, um welchen Befehl es sich handelt, weist sie Data-Path, Memory-Controller und Input/Output an, was zu tun ist.

### 5.1.3 Memory-Controller

Der Memory-Controller stellt die Verbindung zwischen Control-Unit, Data-Path und RAM her. Er führt die eigentlichen Lese- oder Schreibzugriffe in das RAM aus und muss eventuell auch kompliziertere RAM-Protokolle verarbeiten. Je nachdem, welche RAM-Adressen angesprochen werden, wird entweder ein echter Zugriff auf das RAM ausgeführt, oder aber ein Input-Output-Baustein wird angesteuert.

## 5.2 Der xr16 Befehlssatz

### 5.2.1 Register

Der Prozessorkern des XSOC-Projekts (auch xr16-Prozessor genannt) besitzt 16 Register. Einige davon haben feste Funktionen, andere können als general purpose-Register verwendet werden. Die verfügbaren Register sind in Tabelle 5.1 aufgeführt.

### 5.2.2 Befehlssatz

Die Befehle des XSOC - Projektes sollen möglichst einfach gehalten sein, um effizient in Hardware umgesetzt werden zu können (Prinzip: "Simplicity favors regularity"). Insbesondere das Decoding und Pipelining wird dadurch vereinfacht. Andererseits sollen die Befehle aber auch die gesamten Operationen von (integer) C abbilden können. Tablelle 5.2.2 zeigt alle 43 im `xr16-Prozessor` zur Verfügung stehenden Befehle.
Der von Jan Gray gewählte Befehlssatz orientiert sich stark an dem der MIPS-Prozessoren der Firma MIPS Technologies Inc., die bereits seit mehr als 15 Jahren entwickelt werden und entsprechend gut dokumentiert sind. Ein hervorragendes Lehrbuch stammt zum Beispiel von Patterson und Hennessy[28].

Bei der MIPS-Architektur handelt es sich um eine sogenannte Drei-Operanden-Maschine. Jeder (Arithmetik-) Befehl ist von der Form:

Ziel = Quelle1 Operation Quelle2
bzw.
Ziel = Quelle1 Operation Immediate

Da es sich beim `xr16` um einen 16-Bit-Prozessor handelt, müssen die Befehle in 16 Bit breite Maschinenbefehle im Binärformat umgesetzt werden. Diese sogenannten Opcodes können dann direkt vom Prozessorkern verarbeitet werden. Sie können nach der Art der Verknüpfung der Register untereinander klassifiziert werden und sind folgendermaßen aufgebaut:

| **Format des Befehls** | Bit 15-12 | Bit 11-8 | Bit 7-4 | Bit 3-0 |
|---|---|---|---|---|
| rrr | op | rd | ra | rb |
| rri | op | rd | ra | imm |
| rr | op | rd | fn | rb |
| ri | op | rd | fn | imm |
| i12 | op | imm12 | ... | ... |
| br | op | cond | disp8 | ... |

Die ersten vier Bit geben immer die Art des Befehls an. Die Bedeutung der folgenden zwölf Bits ist unterschiedlich je nach Typ des Befehls. Für reine Arithmetik-Operationen zwischen Registern (Typ `rrr`) benötigt man beispielsweise die Angabe dreier Register, für Logik-Befehle (`rr`) die Angabe von Quell und Ziel-Register sowie die Art der anzuwendenden Operation, für Branch-Befehle (`br`) benötigt man eine Sprungbedingung sowie ein Spungziel usw. Im Folgenden werden einige wichtige Beispiele (und die in der obigen Tabelle verwendeten Abkürzungen) erläutert.

#### Arithmetik

Bei den Befehlen `add` und `sub` werden zwei Register `ra` und `rb` miteinander verknüpft und das Ergebnis im Zielregister `rd` abgelegt. Für die Logikoperationen wird der Inhalt des Zielregisters `rd` mit dem des Registers `rb` verknüpft und das Ergebnis im Register `rd` abgelegt.

### Immediates

Es gibt auch die Möglichkeit, Berechnungen mit einem festen Wert durchzuführen. Dieser Wert wird als "immediate" bezeichnet. Aufgrund der auf 16 Bit beschränkten Befehle kann ein Immediate nur 4 Bit enthalten. Um diese Beschränkung zu umgehen, kann man mithilfe des Befehls `imm` einen Wert von 12 Bit bereits einen Taktzyklus vorher laden. Dieser wird dann als die oberen 12 Bit einer 16-Bit-Zahl aufgefasst, die unteren vier Bit sind die vier bit aus dem eigentlichen Immediate Befehl. Um also das Register `r10` um eins zu inkrementieren genügt der Befehl `addi r10,r0,1;`, um aber das Register r10 um 17 zu erhöhen muss man zwei Befehle ausführen:

```
imm 0x0010;
addi r10,r0,1;
```

Dies wird vom Assembler aber automatisch berücksichtigt und muss nicht vom Programmierer explizit angegeben werden. Genauso kann man auch logische Operationen nach dem Schema *ri* (siehe tabelle) mit Immediates durchführen:

```
imm 0xFFF;
andni r11,F;
```

invertiert beispielsweise das Register `r11`. Der Befehl `andni` bildet das bitweise UND mit dem Komplement der Immediate (hier: `F`) und speichert das ergebnis in `rd` (hier Register 11).

### Sprünge und Funktionsaufrufe

Der Befehl `jal` bewirkt einen unbedingten Sprung zu einer 16 Bit breiten Adresse. Diese wird mittels eines Immediate-Prefix festgelegt.

```
imm imm12
jal rd,imm(ra)
```

Der Stand des Program-Counter vor dem Sprung wird im Register `rd` abgespeichert.
Der Befehl `call` wird zum Ausführen einer Unterfunktion verwandt. Argumente werden in den Registern `r3` bis `r5` untergebracht, die Rücksprungaddresse wird in das dafür vorgesehene Register `r15` geschrieben.

### Branches

Um Verzweigungen in Programmen realisieren zu können, besitzt der Befehlssatz so genannte branch-Befehle. Sie haben das Format:

| 15 12 | 11 8 | 7 0 |
|---|---|---|
| B | cond | disp |

Hier steht `B` für einen Branch-Befehl, `cond` ist die Sprungbedingung und `disp` die Sprungweite, die auch negativ sein kann. Der Program-Counter bekommt also folgenden Wert:

$$\texttt{pc = pc + 2} \times \texttt{sign\_ext(disp}_{7:0}\texttt{) + 2 ;}$$

Für `cond`=0 wird ein unbedingter Sprung ausgeführt. Leider stehen für die Festlegung des Sprungziels (dispatch) nur acht Bit zur Verfügung, d.h. man kann nur kurze Spünge (short branches) von 127 Adressen vorwärts oder rückwärts vornehmen. Dies ist eine starke Beschränkung für die Erstellung von Programmen in C für XCONTROL, da schon bei wenigen verschachtelten Schleifen/if-Abfragen die Grenze der short branches erreicht wird. Um dies zu umgehen, muss man Far-Branches mit einer Kombination aus normalen Branches und absoluten Sprüngen implementieren. Beispiel: Der Befehl

```
blt L2
```

muss ersetzt werden durch:

```
bge skip
j L2
skip:
```

Dies bläht natürlich den Instruktionscode etwas auf. Leider ist die automatische Ersetzung von Far-Branches durch eine Kombination wie die oben beschriebene in der aktuellen Version des `xr16` Assemblers nicht implementiert. Falls die Benutzung von Far-Branches notwendig werden sollte, ist die Anpassung aber leicht möglich, da der Assembler im Quellcode zur Verfügung steht.

#### Store- und Load-Befehle

Zum Laden bzw. Speichern von 16-Bit-Wörtern aus dem RAM bzw. in das RAM dienen die Befehle `lw` und `sw`. Es gibt auch die Möglichkeit, mit den Befehlen `lb` und `sb` jeweils nur ein Byte zu lesen oder zu schreiben.

#### Weiterführende Dokumentation

Die komplette Beschreibung aller Befehle von `xr16` ist in [29] abgedruckt.

## 5.3 Der Prozessorkern `xr16`

Der Prozessorkern `xr16` des XSOC-Projekts ist unterteilt in Data-Path (rechts) und Control-Unit (s. links in Abb. 5.2). Die Control-Unit bezieht die als nächstes auszuführende Instruktion vom Memory Controller über `INSN`. Im Inneren der Control Unit befinden sich Register für die Instruktionen, ein Dekoder für die Instruktionen, eine Control-State-Machine und die Operandenauswahl. Letztere teilt dem Datenpfad über die Leitungen `RNA[3:0]` und `RNB[3:0]` mit, welche Register er als Operanden für die nächste Berechnung auszuwählen hat.

### 5.3.1 ALU

Die ALU des `xr16`-Prozessors (siehe Abb. 5.3) besteht aus einem 16-Bit Addierer/Subtrahierer und einer 16-Bit Logikeinheit, die simultan auf den Registern `A` und `B` arbeiten. `LOGIC` berechnet das 16-Bittige Ergebnis der Operationen A UND B, A ODER B, A XOR B oder A ANDNOT B, die mit

| HEX | Format | Assembler | Semantik |
|---|---|---|---|
| **0**dab | rrr | **add** rd, ra, rb | rd = ra + rb; |
| **1**dab | rrr | **sub** rd, ra, rb | rd = ra - rb; |
| **2**dai | rri | **addi** rd, ra, imm | rd = ra + imm; |
| **3**d*b | rr | **{and or xor andn adc sbc}** rd, rb | rd = rd *op* rb; |
| **4**d*i | ri | **{andi ori xori andni adci sbci slli slxi srai srli srxi}** rd, imm | rd = rd *op* imm; |
| **5**dai | rri | **lw** rd, imm(ra) | rd = *(int*)(ra+imm); |
| **6**dai | rri | **lb** rd, imm(ra) | rd = *(byte*)(ra+imm); |
| **8**dai | rri | **sw** rd, imm(ra) | *(int*)(ra+imm) = rd; |
| **9**dai | rri | **sb** rd, imm(ra) | *(byte*)(ra+imm) = rd; |
| **A**dai | rri | **jal** rd,imm(ra) | rd = pc, pc = ra + imm; |
| **B**dd | br | **{ br brn beq bne bc bnc bv bnv blt bge ble bgt bltu bgeu bleu bgtu** } label | if(*cond*) pc +=2*disp8; |
| **C**iii | i12 | **call** func | r15 = pc, pc = imm12«4; |
| **D**iii | i12 | **imm** imm12 | imm'next$_{15:4}$ = imm12; |
| **7**xxx | — | reserved | |
| **E**xxx | — | reserved | |
| **F**xxx | — | reserved | |

**Tabelle 5.2:** Der xr16 Mikroprozessorkern benötigt nur 43 Befehle um eine Integer Teilmenge von C zu implementieren

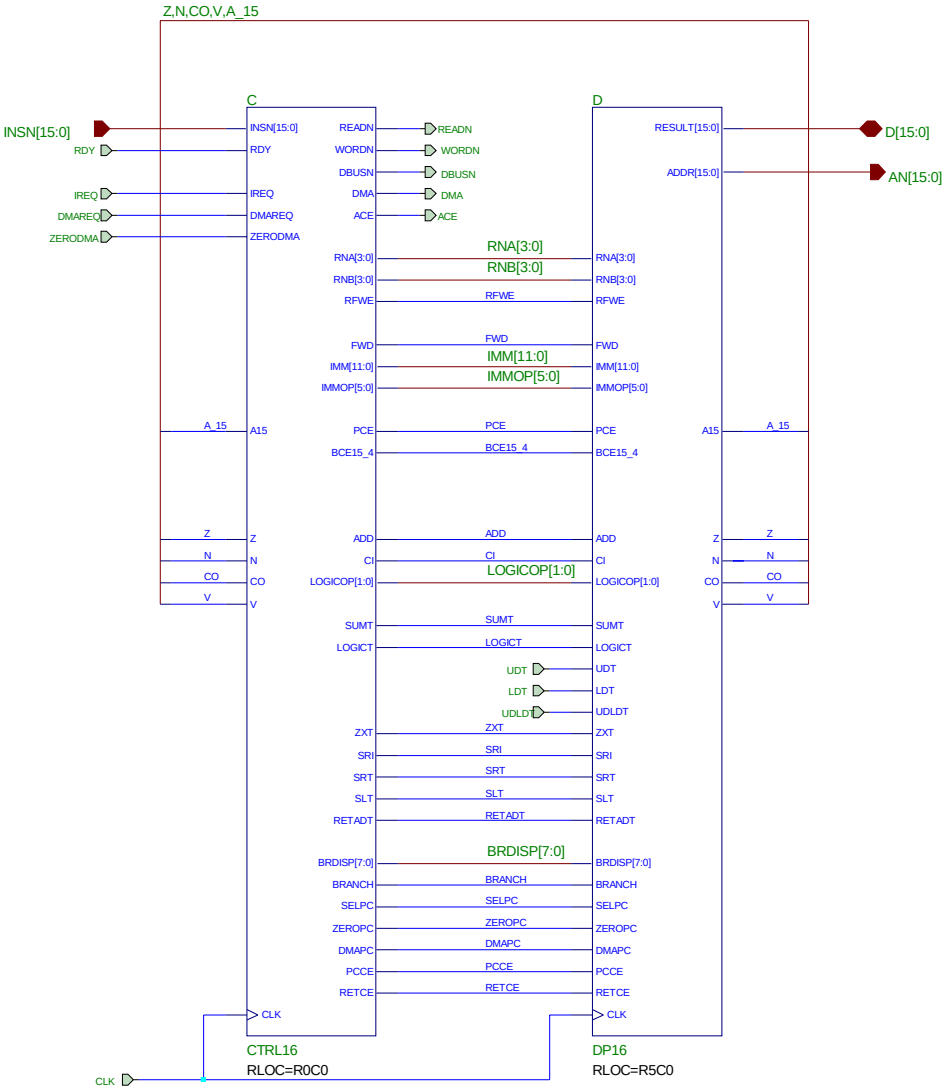



**Abbildung 5.2:** Der Kern des XSOC-Projekts: der `xr16`-Prozessor. Die Control-Unit (links) erhält Instruktionen, die vom Memory Controller aus dem RAM geholt wurden über `INSN[15:0]`. Sie werden dort dekodiert und in Steuersignale umgesetzt (Mitte). Der Data-Path (rechts) enthält Register und ALU. Daten können mit `D[15:0]` in die Register geladen oder von dort auf den Bus geschrieben werden. Über den Bus `AN[15:0]` teilt der Data-Path dem Memory-Controller Adressen mit, von denen Daten gelesen bzw. an die Daten geschrieben werden sollen.

dem Signal `LOGICOP[1:0]` ausgewählt werden können. Jedes Bit ist dabei unabhängig und bei der Implementation auf dem FPGA passt die Logik für je ein Bit in eine Look-Up-Table (LUT), was das Design klein und schnell macht. Der Teil `ADDSUB` der ALU addiert Register `B` zu `A` bzw. subtrahiert `B` von `A` je nachdem, welchen Wert `ADD` hat.

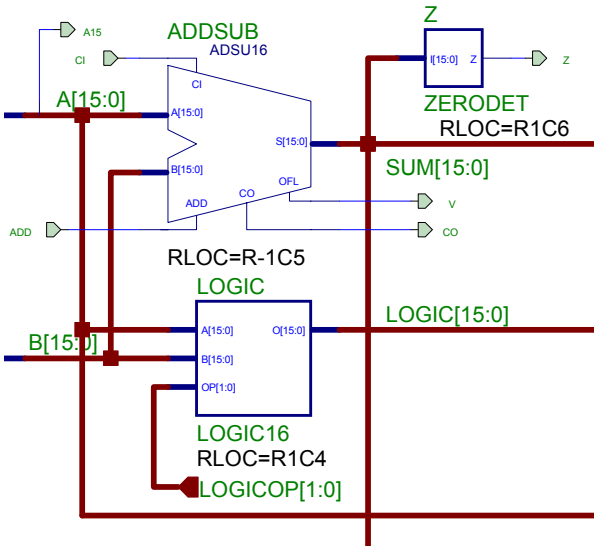



**Abbildung 5.3:** Die ALU des `xr16`-Prozessors

### 5.3.2 Data-Path: Pipelining

Pipelining ist eine Technik, um die Taktfrequenz in Mikroprozessoren zu erhöhen. Die Abarbeitung eines Befehls teilt sich normalerweise in mehrere Stufen auf: zuerst muss der Befehl aus dem Hauptspeicher geholt werden (instruction fetch), dann muss die Art des Befehls festgestellt werden (decode) und schliesslich muss der Befehl selbst ausgeführt werden. Dies alles nimmt eine nicht unerhebliche Zeit in Anspruch, die sich aus den Schaltzeiten der Transistoren sowie aus den Signallaufzeiten auf der Verdrahtungsebene des Chips zusammensetzt. Dies bedeutet, dass die Taktrate entsprechend niedrig gewählt werden muss, um den ganzen Zyklus von fetch bis execute durchlaufen zu können. Insbesondere das Holen der Instruktionen aus dem Hauptspeicher kann bei modernen Computersystemen sehr lange dauern, da der verwendete DRAM sehr viel langsamer arbeitet als die CPU selbst. Um die Taktrate dennoch zu erhöhen, wendet man folgenden Trick an: Man teilt die Ausführung eines Befehls in mehrere Teilschritte auf, die jeweils innerhalb eines Taktes ausgeführt werden können. Dafür kann man die einzelnen Aufgaben parallel ausführen. Diese Herangehensweise ist vergleichbar mit einem Fließband in der Automobilindustrie: Um ein Auto von Anfang bis Ende zu bauen, benötigt man eine relativ lange Zeit. Die Erfindung des Fließbandes löst dieses Problem. Ein Arbeiter bearbeitet immer denselben Teilschritt; dies kann er sehr schnell und effizient tun, während das Fließband die teilweise zusammengebauten Autos von Arbeiter zu Arbeiter transportiert. Diese Art der Arbeitseinteilung verringert zwar nicht die Gesamtzeit, die für ein einzelnes Auto benötigt wird, aber am Ende des Fließbandes kommt ein fertiges Auto mit der Geschwindigkeit des Fließbandes heraus, vorausgesetzt, die Arbeitsschritte sind so fein aufgeteilt, dass sie schnell genug ausgeführt werden können. Der Gesamteffekt ist ein schnellerer Durchsatz durch Parallelisierung der Arbeitslast.

Dasselbe Prinzip wird auch bei Mikroprozessoren eingesetzt. Die Ausführung einer Instruktion wird in mehrere Schritte eingeteilt, die von verschiedenen Teilen des Prozessors ausgeführt werden.

| $\mathbf{t}_1$ | $\mathbf{t}_2$ | $\mathbf{t}_3$ | $\mathbf{t}_4$ | $\mathbf{t}_5$ |
|---|---|---|---|---|
| $\mathbf{IF}_1$ | $\mathsf{DC}_1$ | $\mathsf{EX}_1$ | | |
| | $\mathbf{IF}_2$ | $\mathsf{DC}_2$ | $\mathsf{EX}_2$ | |
| | | $\mathbf{IF}_3$ | $\mathsf{DC}_3$ | $\mathsf{EX}_3$ |
| | | | $\mathbf{IF}_4$ | $\mathsf{DC}_4$ |

**Tabelle 5.3:** `xr16` besitzt eine einfache Pipeline mit drei Stufen

Bei `xr16` wird eine dreistufige Pipeline verwendet. Das Schema ist dabei wie folgt: Zuerst wird Instruktion 1 geholt, im nächsten Takt wird Instruktion 1 dekodiert, gleichzeitig aber schon Instruktion 2 geholt. Im dritten Takt wird Instruktion 1 ausgeführt, Instruktion 2 dekodiert und Instruktion 3 geholt (vgl. Tabelle 5.3)

Dies ermöglicht einen höheren Durchsatz (throughput) von Befehlen durch Parallelisierung. Die Zeit, die für die Ausführung der kompletten Instruktion 1 vergeht, bleibt aber weiterhin gleich (hier: drei Takte). Man spricht hier von Latenzzeit (latency).
Indem man die Befehle in noch kleinere Häppchen unterteilt, die jeder für sich nur sehr wenig Logik und also Schaltzeiten von Transistoren benötigen, kann man die Taktrate weiter in die Höhe treiben. Dies ist insbesondere für FPGAs mit ihren recht hohen Signallaufzeiten auf der Routing-Ebene sehr vorteilhaft. Moderne RISC-artige Mikroprozessoren haben sehr viel mehr Pipeline-Stufen als die von XSOC. Beispielsweise hat die Intel P6-Mikroarchitektur (Pentium Pro, Pentium II, Pentium III) zehn Pipelinstufen, während die P7-Architektur (Pentium 4) nicht weniger als 20 Stufen hat (“hyperpipelined”) und so sehr hohe Taktraten erzielen kann. Als Nachteil handelt man sich aber eine oft unnötig hohe Latenzzeit und weitere Probleme ein, die in den nächsten Abschnitten diskutiert werden. Die Herausforderung an den Prozessordesigner ist, eine ausgewogene Länge der Pipeline zu finden, die eine optimale Geschwindigkeit des Prozessors ermöglicht.

### Data-Hazards

Beim Einsatz von Pipelining ist zu beachten, dass keine Gefahr von Datenverlust (Data-Hazard) auftreten darf. Die Control-Unit des Prozessors muss diese Gefahren erkennen und den Data-Path entsprechend steuern.

Betrachten wir den Fall einer Instruktion, die das Ergebnis ihres unmittelbaren Vorgängers benutzt:

```
I1:  andi r1,7
I2:  addi r2,r1,1
```

Zum Zeitpunkt $\mathbf{t}_3$ in Tabelle 5.3 würde der Schritt $\mathsf{EX}_1$ den Wert `r1=r1 & 7` berechnen, während gleichzeitig die Pipelinestufe $\mathsf{DC}_2$ den alten Wert des Registers `r1` zu lesen versucht. Im Schritt $\mathbf{t}_4$ würde dann 1 zum falschen Wert von `r1` addiert. In `xr16` wird dieses Problem durch *result forwarding* umgangen. In Abb. 5.4 ist ein vereinfachter Ausschnitt aus dem Data-Path von `xr16` gezeigt. Die 16 Register der CPU sind zweimal vorhanden, hier implementiert als `AREGS` und `BREGS`. Über die Signale `RNA[3:0]` und `RNB[3:0]` werden die einzelnen Register ausgewählt. In den Flip-Flops

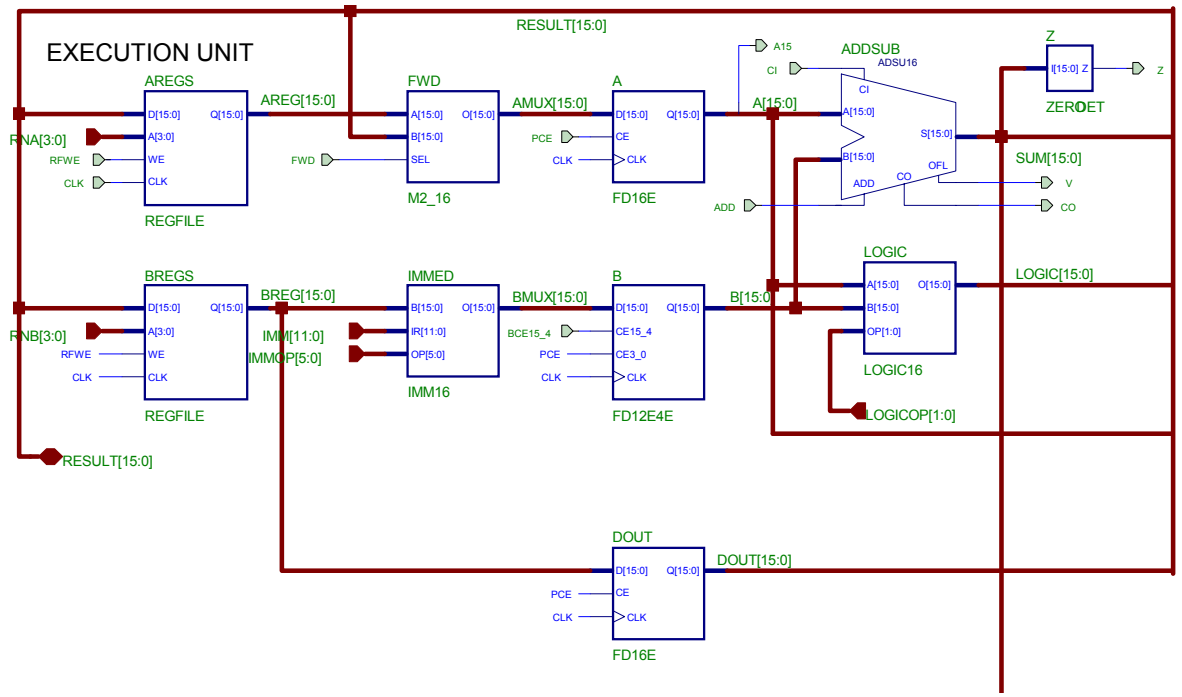



**Abbildung 5.4:** Register und ALU des `xr16`

A und B werden die von der ALU zu verknüpfenden Operanden gespeichert. Der Weg von AREGS nach A benötigt normalerweise einen Taktzyklus. Das Ergebnis der Rechenoperation wird über den `RESULT[15:0]`-Bus wieder in die Register zurückgespeist. Der Multiplexer FWD leitet normalerweise den Inhalt der Register AREGS an das Operandenregister A weiter. Stellt die Control-Unit jedoch den oben beschriebenen Data-Hazard fest, dann wird der Multiplexer umgeschaltet, so dass das Ergebnis der aktuellen Berechnung direkt über den Bus `RESULT[15:0]` in das Register A zurückfließt. Dies ist rechtzeitig, um im nächsten Rechenschritt die richtigen Daten zu verwenden.

**Gepipelinete Ausführung des Load-Befehls** Auch die Memory-Zugriffe sind gepipelined. In Tabelle 5.4 ist das RAM noch nicht bereit, während ein Fetch der Instruktion $I_3$ zum Zeitpunkt $\mathbf{t}_3$ ausgeführt werden soll. Deshalb wird in $\mathbf{t}_4$ der Schritt $\mathbf{t}_3$ nochmal wiederholt (kursiv dargestellt).

**Branches** Wird ein Branch zur Zeit $t_3$ ausgeführt, so muss die $EX_2$-Stufe von $I_2$ annulliert werden, genauso die $DC_3$ und $EX_3$ von $I_3$ (hier durchgestrichen dargestellt).

| $\mathbf{t}_1$ | $\mathbf{t}_2$ | $\mathbf{t}_3$ | $\mathbf{t}_4$ | $\mathbf{t}_5$ |
|---|---|---|---|---|
| $\mathbf{IF}_B$ | $DC_B$ | $EX_B$ | | |
| | $\mathbf{IF}_2$ | $DC_2$ | ~~$EX_2$~~ | |
| | | $\mathbf{IF}_3$ | ~~$DC_3$~~ | ~~$EX_3$~~ |
| | | | $\mathbf{IF}_4$ | $DC_4$ |

| $\mathbf{t}_1$ | $\mathbf{t}_2$ | $\mathbf{t}_3$ | $\mathbf{t}_4$ | $\mathbf{t}_5$ | $\mathbf{t}_6$ |
|---|---|---|---|---|---|
| $\mathbf{IF}_L$ | $\mathsf{DC}_L$ | $\mathsf{EX}_L$ | $\underline{\mathsf{EX}_L}$ | | |
| | $\mathbf{IF}_2$ | $\mathsf{DC}_2$ | $\underline{\mathsf{EX}_2}$ | | |
| | | $\mathbf{IF}_3$ | $\underline{\mathsf{IF}_3}$ | $\mathsf{EX}_3$ | |
| | | | | $\mathbf{IF}_4$ | $\mathsf{DC}_4$ |

**Tabelle 5.4:** `xr16` kann Load-Befehle gepipelined ausführen

## 5.4 Control-Bus und Peripherie

Zur Kommunikation mit den On-Chip-Peripheriebausteinen verfügt XSOC über einen Controlbus (`CTRL[15:0]`) und Aktivierungssignale (`IOSEL[7:0]`) für die verschiedenen Peripheriebausteine. Durch diese standardisierte Schnittstelle ist ein vereinfachtes Design für Peripherie sowie flexible Erweiterbarkeit möglich. Für die Ansteuerung der Peripherie wird das Prinzip des `Memory Mapping` verwendet: Um On-Chip-Peripherie wie die serielle Schnittstelle, den FPGA-Bus etc. anzusprechen, ordnet man jedem Peripheriebaustein eine Speicheraddresse, zu. Der Memory-Controller addressiert dann bei einem Zugriff der CPU auf diese Adresse nicht das RAM, sondern steuert über den Controlbus die On-Chip-Peripherie an. Natürlich kann man die für das Memory-Mapping verwendeten Speicheraddressen nicht mehr als normalen Arbeitsspeicher nutzen. Die Umsetzung dieses Konzeptes bei `XCONTROL` wird im nächsten Kapitel erklärt.

# Kapitel 6

# XCONTROL

In diesem Kapitel werden die wichtigsten Prinzipien und Anwendungsmöglichkeiten von XCONTROL erläutert.

Die Details der Implementierung und die dabei aufgetretenen Probleme werden im nächsten Kapitel behandelt. Konkrete Hinweise zur Anwendung des fertigen Projekts und der dafür geschriebenen Programme werden in Anhang A gegeben. Eine englischsprachige Dokumentation (`XCONTROL-User-Manual`) steht ebenfalls zur Verfügung [30].

## 6.1 Prinzipien

Die größte Herausforderung beim Entwurf von XCONTROL war es, das bisher schon existierende Design des Control-FPGAs mit den neuen Funktionen für die CPU zu vereinen, ohne dabei die Abwärtskompatibilität zu verlieren. Als einfachste Lösung kristallisierte sich nach kurzer Zeit das grundlegende Konzept heraus, zwei getrennte Modi für XCONTROL einzuführen. Im sogenannten "legacy-mode"[1] ist die CPU angehalten. XCONTROL verhält sich wie das ursprüngliche Controller-Design, das die volle Kontrolle über RAM und On-Chip-Peripherie hat. Alle bisher existierende Software zur Steuerung des CATCH lässt sich weiterverwenden. Der zweite Modus is der CPU-Modus. Hier hat die CPU die volle Kontrolle über RAM und Peripheriebausteine. Ein Zugriff von aussen ist nur zum Stop der CPU und Wechsel in den Legacy-Mode möglich, sowie zum Reset. Diese ungewöhnliche Lösung war nötig, um die korrekte Funktion aller bisherigen CATCH-Steuerfunktionen zu garantieren, da Konflikte zwischen CPU und Legacy-Funktionen beim Zugriff auf die Peripherie ausgeschlossen sind. Auch das Lesen und Schreiben des RAMs von der CPU bzw. vom VME-Rechner aus wird so möglich, ohne starke Veränderungen am Memory-Controller des XSOC-Projektes vornehmen zu müssen.

## 6.2 RAM-Zugriff

Die wichtigste Grundfunktion, die in XCONTROL zu implementieren war, ist der Zugriff vom VME-Bus auf den auf dem CATCH befindlichen RAM-Baustein. Dies ist deshalb so wichtig, weil es die

[1] legacy, engl: Hinterlassenschaft, Erbschaft, Altlast

einzige Möglichkeit ist, ein auszuführendes Programm dem CPU-Kern innerhalb des FPGA zugänglich zu machen. Das Problem dabei ist, dass ja die CPU ebenfalls auf das RAM zugreifen muss, aus dem sie ihre Instruktionen bezieht, und den sie zum Arbeiten benutzt. Das XSOC-Projekt unterstützt zwar den Zugriff auf das RAM an der CPU vorbei (DMA[2]), der dort für die VGA-Schnittstelle verwendet wird. Der Zugriff kann aber nur lesend erfolgen. Daher war die naheliegendste Lösung das Entkoppeln von Zugriffen von aussen auf das RAM und den Zugriffen der CPU auf das RAM, indem die CPU angehalten wird.

Für die Umsetzung des RAM-Zugriffs mussten allerdings einige Eigenheiten der vorhandenen Hardware beachtet werden: Zum einen können die Speicheradressen leider nicht direkt in den Adressbereich des VME-Bus eingeblendet werden, da dieser bereits anderweitig belegt ist. Für den RAM-Zugriff ist deshalb eine etwas andere Vorgehensweise nötig: Die insgesamt 32-Bit des Datenbereichs des VME-Busses werden aufgeteilt in 16 Bit für die RAM-Adresse und 16 Bit für die zu übertragenden Daten. Zum anderen ist der SRAM-Baustein (Cypress CY7C199-10) nur acht Bit breit, so dass XCONTROL intern immer zwei Zugriffe für ein 16-Bit Datenwort ausführen muss.

Der eigentliche RAM-Zugriff erfolgt über VME-Befehle. Diese können entweder direkt von der Kommandozeile des VME-Rechners oder mit Hilfe der `vme_access`-Bibliothek aus einem C-Programm heraus ausgeführt werden.

**Schreibzugriff** Der Schreibzugriff erfolgt mit dem Befehl:

```
vme_write e0xx0804 aaaadddd
```

Dies bewirkt einen Schreibzugriff an die Adresse `0xaaaa` des RAM, wobei das 16-Bit-Wort `0xdddd` geschrieben wird. Die Zahlen `xx` stehen hier für die hexadezimale Seriennummer des CATCH. Zum Schreiben führt XCONTROL intern zwei RAM-Zugriffe aus. Die Adresse wird dabei intern automatisch inkrementiert. Man kann also von aussen nur *gerade* Speicheradressen ansprechen.

**Lesezugriff** Beim Lesezugriff ist zu beachten, dass bei der oben angegebenen Adressierungsmethode zwei VME-Zugriffe ausgeführt werden müssen, um ein 16-Bit-Wort vom CATCH zu lesen. Dies liegt natürlich daran, dass ja der Datenbereich des VME-Bus zur Übertragung der RAM-Adressen verwendet wird und man nicht gleichzeitig Daten lesen und schreiben kann. Der Befehl

```
vme_write e0xx0808 aaaa0000
```

liest ein 16-bit-Wort an der Adresse `0xaaaa` des RAM in ein internes Register des FPGA. Es müssen dazu intern wieder zwei Zugriffe ausgeführt werden. Mit

```
vme_read e0xx0800
```

wird dann der Wert des Registers auf dem VME-Bus gelegt und kann von der CPU des VME-Rechners ausgelesen werden. Nur die oberen 16 Bit sind gültige Daten. Die unteren 16 Bit Bestehen aus der Revisionsnummer von `XCONTROL` sowie dem Status der CMC Karten (Busy-Flag, s. Abschn. 6.7.2).

| bits 31:16 | bits 15:8 | bits 7:0 |
|---|---|---|
| RAM Daten | `0 0 CMC_BUSY[3:0] 0 0` | Revisionsnummer |

[2]Direct Memory Access

## 6.3 CPU starten und stoppen

Für das Umschalten zwischen den beiden Modi von XCONTROL wurden ebenfalls VME-Befehle gewählt. XCONTROL startet nach der Programmierung im normalen “legacy”-Modus, d.h. alle Funktionen des alten Controller-Designs (Version 12 und niedriger) sind verfügbar. Zusätzlich ist der Zugriff über VME auf das RAM möglich (siehe oben). Die CPU ist zunächst inaktiv, XCONTROL befindet sich im Legacy-Modus.

- CPU reset

  ```
  vme_write e0xx080c 2
  ```

- CPU aktivieren:

  ```
  vme_write e0xx080c 1
  ```

- CPU anhalten - “legacy”-Modus aktivieren

  ```
  vme_write e0xx080c 0
  ```

Während die CPU aktiv ist, ist kein RAM-Zugriff über VME möglich, ein Lesezugriff zeigt dann nur `FFFF FFFF` als Daten an. XCONTROL ist so entworfen worden, dass die CPU wirklich angehalten wird, indem der interne Clock Divider, der den Takt für den CPU-Kern herstellt, deaktiviert wird. Die CPU sollte also wieder gestartet werden können, wobei das gerade geladene Programm einfach an der Stelle weitergeführt wird, wo die CPU gestoppt wurde. Dies funktioniert natürlich nur dann zuverlässig, wenn man indessen nicht aktiv über VME den Inhalt des RAMs - insbesondere den auszuführenden Code - verändert. Ansonsten gerät die CPU in einen undefinierten Zustand. Deshalb muss man jedes mal, wenn man die CPU angehalten und ein neues Programm hineinprogrammiert hat, zuerst ein Reset ausführen!! Das Reset bewirkt das Zurücksetzen aller Flip-Flops des FPGA; das eigentliche Design sowie das im SRAM befindliche auszuführende Programm sind davon nicht betroffen.

## 6.4 Display

Auf der aktuellen Revision des CATCH (Rev. 1) ist ein vierstelliges Dot-Matrix-Display untergebracht wie es in Abb. 6.1 zu sehen ist. Es kann zum Beispiel dafür verwendet werden, die Seriennummer des CATCH oder dessen Source-ID anzuzeigen. Die Source-ID ist eine hardwareunabhängige Identifikationsnummer, die einem CATCH zugeordnet wird, je nachdem, an welchem Detektor es angeschlossen wird. Die Source-ID wird bei der Initialisierung in das CATCH hineinprogrammiert und wird in den Headern der Messdaten vom CATCH mitgeschickt. Dies ermöglicht den Austausch von CATCH-Modulen am Experiment, ohne festverdrahtete Seriennummern ändern zu müssen.

Eine andere Möglichkeit wäre zum Beispiel, die Triggerrate oder die verarbeitete Datenmenge auf dem Display anzuzeigen.

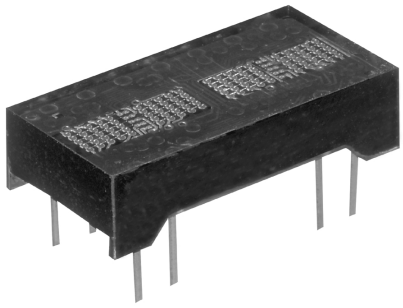


**Abbildung 6.1:** Das im CATCH integrierte Display (Foto: Infineon)

Auch Fehlermeldungen können von der CPU ausgegeben werden. Auf dem Display können vier Zeichen dargestellt werden. Jedes Zeichen besteht aus einer Matrix aus $5 \times 5$ Punkten, so dass quasi beliebige Zeichen zusammengesetzt werden können.

Genaue Einzelheiten zum Typ und zur elektrischen Ansteuerung des Displays finden sich in Anhang B.1.

### 6.4.1 Adressierung des Displays - Memory Mapping

Das Display kann von der CPU aus angesprochen werden, indem man einen Schreibzugriff auf eine bestimmte Addresse ausführt. Dabei wird jeweils ein Byte übertragen, das im Display-Interface in ein passendes serielles Format umgewandelt und an das Display geschickt wird. Die Speicherstellen für die on-chip Peripherie des XCONTROL wie z.B. das Display oder den internen FPGA-BUs (siehe nächsten Abschnitt) beginnen ab der Adresse 0x8000. Dies liegt daran, dass der 32K SRAM des CATCH nur 15 Bit für Adressen hat, das höchste Bit, das der Prozessor ansprechen könnte, bliebe also für den RAM-Zugriff ungenutzt und wird zur Kennzeichnung eines Peripheriezugriffs verwendet. Die Aufteilung des Peripherieaddressraums erfolgt dabei so:

| Bits | 15 | 14 - 12 | 11 - 8 | 7 - 0 |
|---|---|---|---|---|
| Inhalt | 1 | SEL | ADDRESS-DATA | ADDRESS-DATA |

Hier ist SEL eine Zahl zwischen 0 und 7, die die Nummer des Peripheriebausteins angibt. Für das Display gilt SEL = 010

| 15 | 14-12 | 11-8 | 7-0 |
|---|---|---|---|
| 1 | 0 1 0 | - - - - | - - - - - - - - |

Hierbei ist 010 die Kennzeichnung für den (on-chip) Peripheriebaustein Display, die restlichen Bits der Adresse spielen keine Rolle. Die Adresse des Displays ist also aus Sicht der CPU 0xA000. Um ein Byte Daten auf das Display zu schicken, schreibt man einfach ein 16-Bit-Wort an die entsprechende Adresse, die unteren acht Bit des Wortes werden dann ausgegeben. Man könnte vermutlich

auch ein Byte an diese Adresse schreiben, dies macht aber keinen Unterschied, da der 16-Bit-Zugriff genauso lange dauert.

## 6.5 Bus-Interface

CATCH besitzt einen internen Bus, der die FPGAs untereinander verbindet. Über ihn kann auf interne Register der verschiedenen FPGAs zugegriffen werden. Sie enthalten teilweise Statusinformationen, die ausgelesen werden können. In manche Register kann man auch Daten schreiben, die zur Steuerung der FPGAs dienen. Beispielsweise kann die Zahl der empfangenen Trigger von der CPU ausgelesen werden. Diese befindet sich in einem dafür vorgesehenen Register des TCS-FPGA. Einige der FPGAs haben auch Register, die den aktuellen Zustand der internen State-Machines ausgeben, so dass festgestellt werden kann, ob der FPGA noch arbeitet oder abgestürzt ist. Ein kleines C-Programm, das in `XCONTROL` läuft, kann diese Informationen auswerten und selbstständig FPGAs zurücksetzen oder abschalten.

Der FPGA-Bus ist 16 Bit breit. Das Busprotokoll ist eng an das des VME-Bus angelehnt. Die FPGAs (Formatter, Merger, S-Link, TCS) stellen Register zur Verfügung, die über den Bus gelesen und geschrieben werden können.

### 6.5.1 Ansteuerung des FPGA-Bus

Der Bus ist ähnlich wie das Display in die normalen Adressen des Prozessors eingebunden (“Memory Mapping”).

| Bits | 15 | 14-11 | 10-8 | 7-0 |
|---|---|---|---|---|
| Inhalt | 1 | SEL | CHIP | Register |

SEL ist hier `0 0 1` für Lesezugriffe und `0 1 1` für Schreibzugriffe auf den Bus. CHIP gibt die Nummer des anzusprechenden FPGAs auf dem CATCH an. Jeder FPGA besitzt eine gewisse Anzahl von Registern, deren Bedeutung im CATCH-User-Manual dokumentiert ist.

Es stehen folgende Adressen zur Verfügung:

| | SEL | CHIP | REG | HEX | Funktion |
|---|---|---|---|---|---|
| 1 | 001 | 0100 | Register | 9400 | Formatter lesen |
| 1 | 001 | 0101 | Register | 9500 | TCS lesen |
| 1 | 001 | 0110 | Register | 9600 | S-Link lesen |
| 1 | 001 | 1000 | Register | 9800 | Merger 1 lesen |
| 1 | 001 | 1001 | Register | 9900 | Merger 2 lesen |
| 1 | 001 | 1010 | Register | 9a00 | Merger 3 lesen |
| 1 | 001 | 1011 | Register | 9b00 | Merger 4 lesen |
| | | | | | |
| 1 | 011 | 0100 | Register | b400 | Formatter schreiben |
| 1 | 011 | 0101 | Register | b500 | TCS schreiben |
| 1 | 011 | 0110 | Register | b600 | S-Link schreiben |
| 1 | 011 | 1000 | Register | b800 | Merger 1 schreiben |
| 1 | 011 | 1001 | Register | b900 | Merger 2 schreiben |
| 1 | 011 | 1010 | Register | ba00 | Merger 3 schreiben |
| 1 | 011 | 1011 | Register | bb00 | Merger 4 schreiben |

Die Funktionen der einzelnen Register sind alle in Abschnitt 3 des CATCH-User-Manual [31] aufgeführt. Die Nummerierung der Register weicht jedoch aufgrund der unterschiedlichen VME-Adressbereiche voneinander ab (XCONTROL spricht den Bus direkt an). Die Adresse der Register errechnet sich aus der im CATCH Manual angegebenen, indem man die Hex-Zahl zwei Bit nach rechts verschiebt, und eine 9 oder ein B für Lese/Schreibzugriff davorschreibt. So wird beispielsweise aus dem Register `10 08` im Formatter, das die CATCH Seriennummer enthält, die Speicheradresse `0x9402` für den Lesezugriff über die CPU. In C errechnet sich die Speicheraddresse wie folgt:

$$memory\_address = (1 << 15) + (SEL << 12) + vme\_address >> 2$$

`vme_address` ist hier die Adresse des Registers, wie sie im CATCH-User-Manual dokumentiert ist. Die Tabelle 6.1 gibt als Beispiel die Register des Formatter FPGAs und ihre Funktionen an. Für die anderen FPGAs siehe das CATCH-Manual.

## 6.6 Reset der FPGAs

Über das Reset-Interface ist es möglich, einzelne FPGAs zurückzusetzen, wenn festgestellt wird, dass sie hängengeblieben sind. Auch die vier CMCs des CATCH verfügen über eine Reset-Leitung, die ebenfalls von der CPU kontrolliert werden kann.

Das Reset von FPGAs und CMCs wird ähnlich wie das Bus-Interface gehandhabt. Vom CONTROL-FPGA gehen Reset-Leitungen zu den sieben anderen FPGAs (Merger, S-Link, TCS, Formatter) sowie zu den CMC-Karten. Ausserdem gibt es noch je eine Reset-Leitung zum Trigger-CPLD sowie zu einer (optionalen) Extension CMC. Diese Reset-Leitungen sind alle active low.
Die CPU in XCONTROL kann die Reset-Leitungen ansteuern. Dazu wird ein 16-Bit-Wort an die Speicheradresse `0xC000` geschrieben. Ein Reset wird ausgelöst, wenn das für den jeweiligen FPGA zuständige Bit 1 gesetzt wird. Eine Übersicht über die verwendbaren Bits gibt Tabelle 6.2

| Register | Read/Write | Bits | Funktion |
|---|---|---|---|
| 00 | R | 0-7 | Formatter Design Versionsnummer |
| | | 8-13 | State of readout state machine |
| | | 14 | Data FIFO almost full |
| | | 15 | Data FIFO full |
| 01 | R/W | 0 | 1=Switch off readout CMC-1 (Merger A) |
| | | 1 | 1=Switch off readout CMC-2 (Merger B) |
| | | 2 | 1=Switch off readout CMC-3 (Merger C) |
| | | 3 | 1=Switch off readout CMC-4 (Merger D) |
| | | 4 | ADC type readout, 0=TCD readout (default) |
| | | 5 | 0=compressed mode, 1=uncompressed (default) |
| 02 | R/W | 15-0 | CATCH serial number |
| 03 | R/W | 7-0 | MERGE software revision |
| | | 15-8 | S-Link software revision |
| 04 | R/W | 0-15 | CMC-1 id |
| 05 | R/W | 0-15 | CMC-2 id |
| 06 | R/W | 0-15 | CMC-3 id |
| 07 | R/W | 0-15 | CMC-4 id |

**Tabelle 6.1:** Register des Formatter-FPGAs und ihre Funktion.

| Speicheradresse | Bit | Funktion |
|---|---|---|
| `0xC000` | 0 | reset CMC A |
| | 1 | reset CMC B |
| | 2 | reset CMC C |
| | 3 | reset CMC D |
| | 4 | reset MERGE-FPGA A |
| | 5 | reset MERGE-FPGA B |
| | 6 | reset MERGE-FPGA C |
| | 7 | reset MERGE-FPGA D |
| | 8 | reset Extension CMC |
| | 9 | reset FORMAT-FPGA |
| | 10 | reset TCS-FPGA |
| | 11 | reset SLINK-FPGA |
| | 12 | reset TRIGGER-CPLD |

**Tabelle 6.2:** Resets an die FPGAs und CMCs werden über die Speicheradresse `0xC000` gegeben

## 6.7 Serielles Interface

Vom CATCH aus können serielle Daten an die Front-End-Karten verschickt werden. Sie dienen zur Konfiguration der Front-End-Elektronik, beispielsweise kann der $\mathcal{F}1$-Chip damit initialisiert werden. Der $\mathcal{F}1$ besitzt 16 Register mit denen unter anderem Schwellwerte für die ASD8-Vorverstärkerchips gesetzt werden, Zeitfenster für die Digitalisierung gewählt sowie die Auflösung eingestellt werden können.

Die Daten werden in Worten von 24 Bit übertragen. Das Datenformat ist analog zu einem seriellen UART-Interface, wie es auch für PCs verwendet wird. Bei jedem Datenwort wird zunächst eine Startkennung (Hi-Low) geschickt, dann folgen 24 Bit Daten und dann eine Stopkennung (Hi-Low). Für einen $\mathcal{F}1$-TDC-Chip sind die 24 Bit Daten unterteilt in 8 Bit Adressen und 16 Bit Setup-Daten (siehe Abb. 6.2). Die achtbittige Adressinformation is unterteilt in 3 Bit, die die Nummer des angesprochenen Chips (es können ja mehrere TDCs auf einer Front-End-Karte sein) und 4 Bit, die das zu schreibende Register kennzeichnen. Ein weiteres Bit ("common bit" dient dazu, alle Chips auf dem Board gleichzeitig zu programmieren.

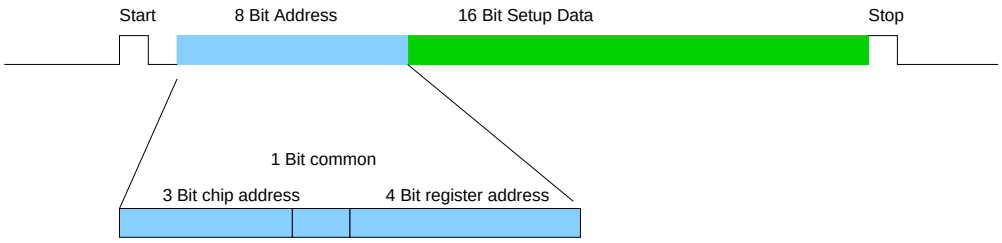



**Abbildung 6.2:** Serielles Datenformat zur Initialisierung des $\mathcal{F}1$-TDC-Chip

XCONTROL bietet zwei Arten, diese seriellen Daten zu verschicken. Zum einen können direkt VME-Befehle ausgeführt werden, um je ein Datenwort an einen oder alle Front-End-Karten zu senden. Die dazu notwendigen Befehle sind im CATCH-Manual beschrieben [31].
Die zweite Möglichkeit ist der Zugriff auf die serielle Schnittstelle von der CPU aus, wobei die Daten vorher im RAM des CATCH gespeichert werden müssen. Dies wird im Folgenden beschrieben.

### 6.7.1 Daten schreiben

Für das Senden von 24 Bit Daten über die serielle Schnittstelle sind drei Zugriffe notwendig, da der Bus der CPU nur 16 Bit breit ist. Der erste Zugriff schreibt die ersten acht der 24 Bit in ein internes Register. Sie entsprechen den Adressen der Register in den $\mathcal{F}1$-Chips. Der zweite Zugriff schreibt die eigentlichen Setup-Daten in ein Register. Der dritte Zugriff gibt dann ein Write-Enable.

| Adresse | Funktion |
|---|---|
| 0xd001 | schreibe 8 Bit Daten |
| 0xd002 | schreibe 16 Bit Daten |
| 0xd004 | schreibe 16 Bit Write Enable |
| 0xd008 | lese 4 Bit Busy-Flag |

Je nachdem, welche der Bits des Write-Enable Wortes gesetzt sind, werden die 24 Bit Daten an die einzelnen Karten versandt. Die Zuordnung der Ports zu den gesetzten Bits im Enable-Wort erfolgt dabei von links nach rechts: ganz rechts ist das Flag für Port 0, ganz links das Flag für Port 15. Benutzt man zum Beispiel `0x0003` als Write-Enable, werden Daten auf den ersten beiden Kanälen der ersten HOTLink-Karte im CATCH gesendet, also Daten an die ersten zwei der 16 maximal möglichen Front-End-Karten geschickt.

| **Port 15** | ··· | **Port 0** |
|---|---|---|
| Bit 15 | ··· | Bit 0 |

Im Abschnitt A.3 des Anhangs gibt es dazu ein Beispiel-Programm.

### 6.7.2 Busy-Flag lesen

Der Schreibzyklus der seriellen Schnittstelle, vom Schreiben des Write-Enables bis zum Stop-Bit des 24-Bit-Wortes, dauert 2.8 $\mu$s. Während dieser Zeit wird ein Busy-Flag auf High gesetzt, das anzeigt, dass die CMC-Karte bereits einen seriellen Transfer durchführt. Diese vier Flags (pro CMC eines) werden im "legacy"-Mode von `XONTROL` über das VME-Interface ausgegeben, wenn man ein `vme_read exxx0800` durchführt. Die Flags sind aber auch der CPU zugänglich, wenn man lesend auf die Adresse `0xd008` zugreift. Das serielle Interface liefert dann einen Wert zurück, dessen unterste vier Bits jeweils für die aktive CMC-stehen. Die Bits können den CMC-Karten wie folgt zugeordnet werden:

| **CMC 3** | ··· | **CMC 0** |
|---|---|---|
| Bit 3 | ··· | Bit 0 |

### 6.7.3 Setup-Daten aus dem RAM schreiben

Um Daten zur Initialisierung von Front-End-Boards (das können bis zu 16 Stück pro CATCH sein) nicht fest in ein C-Programm einbinden zu müssen, sollte ein Speicherbereich im RAM fest reserviert werden. Ein Beispiel für eine sinnvolle Aufteilung zeigt Abb. 6.3.

Diese Daten können dann etwa bei Inbetriebnahme des CATCH bei abgeschalteter CPU ins RAM geschrieben werden. Dafür wird ein Programm benötigt, das auf dem VME-Rechner läuft. Beispielsweise kann `f1toram` (siehe Anhang A.3.1) verwendet werden. Ein darauf abgestimmtes CATCH-Überwachungsprogramm, das auf XCONTROL läuft, kann dann die Initialisierung über die serielle Schnittstelle bei Bedarf vornehmen. Ein Beispielprogramm, das dies ausführt ist `ramtoreg` (s. Anhang A.3.2).

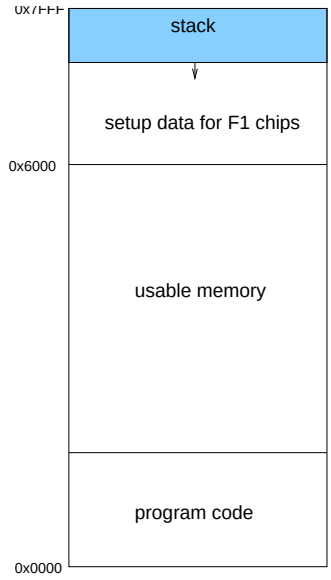



**Abbildung 6.3:** Aufteilung des Adressraums für XCONTROL

# Kapitel 7

# Implementierung

In der Auslese-Elektronik des COMPASS-Experiments werden rekonfigurierbare Logikbausteine verwendet. Dies ermöglicht im Gegensatz zum Einsatz von anwendungsspezifischen integrierten Schaltungen, die fest verdrahtet sind und ASICs[1] genannt werden, eine spätere Anpassung der entworfenen Logik sowie eine schnellere Entwickungszeit. Für zeitkritische Anwendungen werden hauptsächlich sogenannte Field Programmable Gate Arrays (FPGAs) als rekonfigurierbare Bausteine eingesetzt. Diese können aufgrund des rasanten Wachstums im Halbleiterbereich [33] inzwischen mit der Komplexität von ASICs mithalten.

Im folgenden Kapitel wird beschrieben, wie das Projekt `XCONTROL` in einem solchen FPGA implementiert wurde.

## 7.1 Aufbau eines FPGAs

FPGAs sind rekonfigurierbare Logikbausteine. Aufgrund ihrer sehr grossen Flexibilität gegenüber festverdrahteten ASICs werden sie auf dem CATCH sowie bei anderen Komponenten des Auslesesystems von COMPASS eingesetzt. Um die Funktionsweise des FPGA-Designprozesses und die dabei auftretenden Probleme zu verdeutlichen, soll hier zunächst sehr knapp der prinzipielle Aufbau eines FPGAs erläutert werden. Die Angaben beziehen sich im Folgenden auf den beim Entwurf von `XCONTROL` verwendeten Typ `XCS40XL` der Firma XilinX [34].

Die Chipfläche von FPGAs ist in Blöcke unterteilt. Hierbei unterscheidet man Input-Output-Blöcke, die am Rand des Chips untergebracht sind und konfigurierbare Logikblöcke (sog. CLBs), die in einer Matrix im Inneren des FPGAs liegen und die eigentlichen Logikfunktionen übernehmen, wie in Abbildung 7.1 dargestellt ist. Der XCS40XL besitzt beispielsweise 784 CLBs (28 mal 28). Die Blöcke sind untereinander durch Leitungen verbunden, die Mithilfe von Routing-Matrizen miteinander verschaltet werden können.

Die CLBs selbst bestehen, wie in Abb. 7.2 zu sehen, aus Look-Up-Tabellen (kurz: LUT), die die Realisierung beliebiger Logikfunktionen mit vier Eingängen und einem Ausgang ermöglichen. Die LUTs sind als SRAM-Zellen ausgeführt, was den Herstellungsprozess der Bauteile stark vereinfacht und sie praktisch beliebig oft reprogrammierbar macht. Je zwei dieser Look-Up-Tabellen befinden sich

[1] Application Specific Integrated Circuit

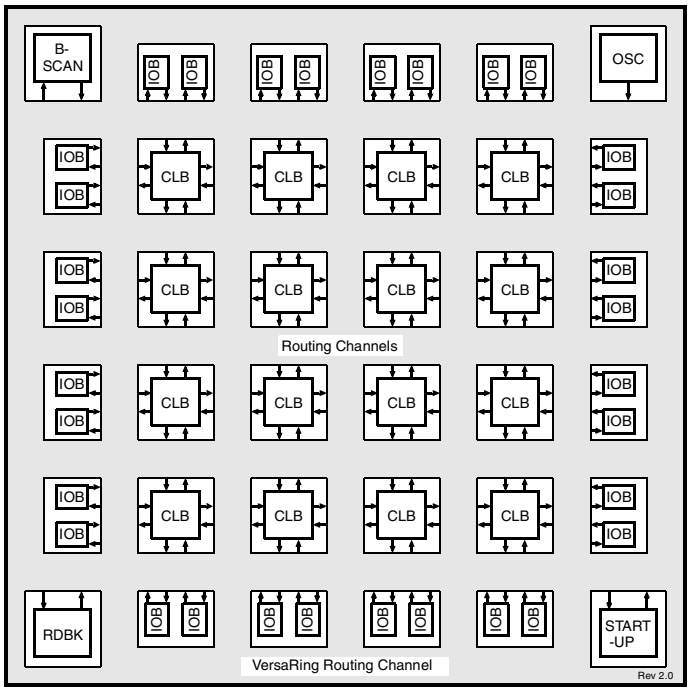



**Abbildung 7.1:** Die CLBs eines XCS40XL sind in einer Matrix angeordnet, die von IO-Blocks umgeben ist, welche zur Ein- und Ausgabe von Signalen auf den Pads des FPGAs dienen (Aus [35])

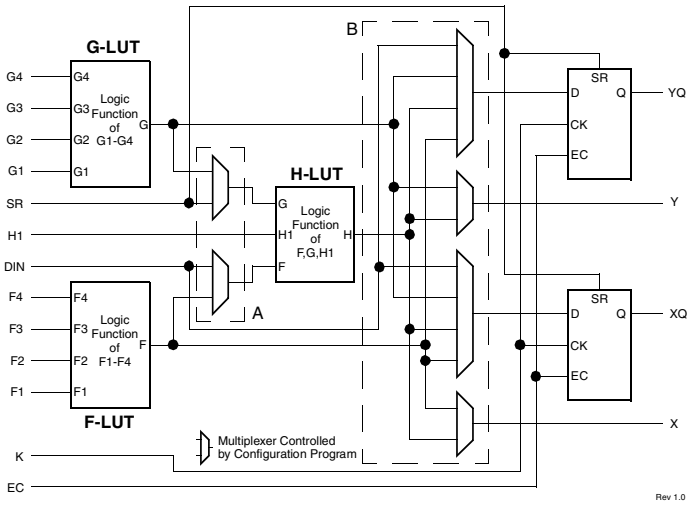



**Abbildung 7.2:** Ein CLB des XCS40XL besteht aus zwei Look-Up-Tables mit je vier Eingängen (G-LUT, F-LUT), einem mit drei Eingängen (H-LUT), Multiplexern sowie zwei Flip-Flops (Aus [35])

in einem CLB und sind durch eine dritte LUT sowie Multiplexer miteinander verbunden, was komplexere Logikfunktionen erlaubt. Ausserdem enthält jedes CLB zwei Flip-Flops, die für synchrone Logik

(Register, RAM etc.) verwendet werden können.

Beim Entwurf eines Designs für den FPGA wird zunächst aus dem vom Benutzer in einer HDL oder Schematics erstellten Entwurf eine Netzliste erzeugt, die alle Logikelemente und ihre Verknüpfungen untereinander enthält. Aufgabe der Software des FPGA-Herstellers ist es jetzt, die Logikelemente sinnvoll auf die verfügbaren CLBs zu verteilen (Placement) und diese dann mittels der Routing-Matrizen zu verknüpfen (Routing). Dabei muss berücksichtigt werden, dass die Signallaufzeiten zusammen mit den Schaltzeiten der Logikelemente die festgelegten Anforderungen an das Design (Taktfrequenz, Signaltiming) nicht überschreiten. Der Place-and-Route-Schritt ist daher sehr kompliziert und beansprucht die meiste Rechenzeit (20 Min. und mehr) bei der Implementierung des Designs.

## 7.2 XSOC Quellen

Die von Jan Gray implementierte XSOC CPU lag im Schematics-Format vor. Dies ermöglicht einerseits einen sehr einfachen Überblick über die Arbeitsweise des Prozessors, da man den einzelnen Bestandteilen bei der Arbeit quasi zusehen kann. Andererseits ermöglicht das Xilinx-Schematics-Format einige sehr spezielle Anpassungen auf die Zielarchitektur, die zu Inkompatibilitäten beim Wechsel auf einen anderen FPGA führen könnten. Auch eine Integration in eine vollständig VHDL-basierte Designumgebung, wie sie beim Entwurf des HOTLink-Tester-Designs [14] verwendet wurde, ist damit nicht möglich. Es existiert jedoch eine Version des XSOC-Projektes in der Hardwarebeschreibunssprache `Verilog`. Um das Erlernen einer weiteren HDL zu vermeiden, wurde die Schematics-Verion von XSOC gewählt.

## 7.3 Anpassen von XSOC

Das XSOC-Projekt wurde von Jan Gray ursprünglich für ein Demonstrationsboard für Lernzwecke konzipiert. Auf ihm befindet sich ein XilinX-FPGA des Typs XC4005. Durch die Tatsache, dass für die gesamte rekonfigurierbare Logik des CATCH ebenfalls XilinX-FPGAs mit der passenden Design-Software verwendet werden, wurde die Anpassung von XSOC für den Control-FPGA erst möglich. Weiter vereinfacht wurde das Übernehmen des Prozessorkerns durch die starke Ähnlichkeit der FPGA-Architekturen der Serie XC4000 und dem für den Control-FPGA verwendeten Spartan XCS40XL. Dennoch mussten einige Änderungen vorgenommen werden, auf die nun kurz eingegangen werden soll.

### 7.3.1 RLOC-Constraints

Normalerweise entscheidet die Design-Software selbst anhand der vorgegebenen Randbedingungen über die Verteilung der Logikelemente auf die CLBs. Dies geschieht mittels heuristischer Algorithmen, liefert jedoch nicht immer ein optimales Ergebnis. Daher sieht die von XilinX gelieferte Design-Software die Möglichkeit vor, selbst über das Placement zu bestimmen. Dies kann mit sogenannten “RLOC”-Constraints direkt im Schematics-Editor geschehen. Eine vergleichbare Möglichkeit auf HDL-Ebene existiert übrigens nicht.

Die Verwendung der RLOC-Constraints zur Designoptimierung wurde von Jan Gray beim Erstellen des ursprünglichen XSOC-Designs benutzt, was zu einem sehr kompakten und schnellen Pro-

zessordesign führte. Leider sind diese Constraints immer auf einen bestimmten FPGA-Typ ausgelegt. Auf Grund der engen Verwandschaft zwischen den FPGAs der XC4000-Serie und dem verwendeten XCS40XL liessen sich jedoch die relativen Placement-Constraints beibehalten, nur wenige absolute Constraints mussten geändert werden, um ein verbessertes Timing zu erhalten.

### 7.3.2 VGA

Auf der ursprünglichen Version des XSOC-Projekts ist eine einfache Grafikschnittstelle (VGA) vorgesehen. Diese wird auf dem CATCH nicht benötigt und wurde deshalb entfernt.

### 7.3.3 Clock

Die Taktfrequenz, mit der das XSOC-Design auf dem ursprünglich vorgesehenen Demonstrationsboard betrieben wird, beträgt 12.5 MHz, was für den Betrieb eines Monitors an der eingebauten VGA notwendig ist. Die Frequenz des auf dem CATCH zum Betrieb der FPGAs verwendeten Taktgenerators ist 40 MHz. Das Timing des XSOC-Projekts ist aber nicht gut genug, um bei dieser Taktfrequenz auf einem XCS40XL-FPGA zu funktionieren. Deshalb wird mit einem auf dem FPGA implementierten Clock-Divider ein Takt von 10 MHz erzeugt, mit dem der Prozessorkern von `XCONTROL` betrieben wird. Die anderen Teile des Control-FPGAs (Bus-Interface, “legacy”-Teil etc.) laufen jedoch weiterhin mit 40 MHz. Lediglich die serielle Schnittstelle bezieht ihren Takt nicht vom CATCH-internen Oszillator, sondern verwendet die experimentweite 38.88 MHz-Clock vom TCS-Receiver.

### 7.3.4 Pull-Up Widerstand

Bei der Implementierung des Memory-Controllers des XSOC-Projekts wurde im Schematics-Editor ein Pull-Up-Symbol verwendet. Es zieht das Signal `CTRL0` auf High, wenn es nicht benötigt wird. Der Spartan XCS40XL enthält aber keine internen Pullup-Widerstände, so dass das Symbol im Schematics-Editor entfernt werden musste, um `XSOC` im Spartan FPGA zum laufen zu bringen.

## 7.4 Implementierung der neuen Schnittstellen

### 7.4.1 Umschaltung zwischen “legacy”-Modus und CPU-Modus

Das bisherige Controller Design ermöglichte den Zugriff auf die verschiedenen Schnittstellen direkt vom VME-Rechner aus. Die Vorgabe war, diese Funktionalität beizubehalten und zusätzlich alle Schnittstellen der CPU zugänglich zu machen. Da ein paralleler Zugriff auf die Schnittstellen einen erheblichen Design- und Verifikationsaufwand bedeutet hätte, wurde die in Abschnitt 6.3 gewählte Möglichkeit zur Umschaltung zwischen der bisherigen Zugriffsart über VME und der Kontrolle durch die CPU gewählt. Implementiert wurde dies durch Erweiterung des ABEL-Codes des `CONTRL`-Makros des alten Controller Designs. Dieses verarbeitet von aussen angelegte VME-Befehle mit der Adresse `e0xx08xx` und wurde so verändert, dass es das Signal `CPU_ENA` (active low) anlegt, wenn der VME-Befehl zum aktivieren der CPU gegeben wurde. Dieses Signal steuert verschiedene Multiplexer, die entweder die Signale der Legacy-Funktionen oder aber die der CPU an die Schnittstellen weitergeben. Dies ist in Abb. 7.3 zu sehen:

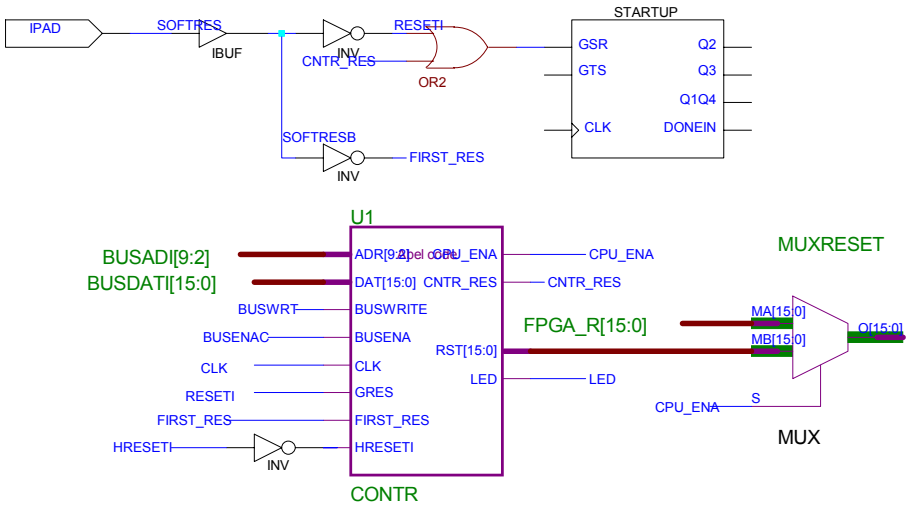



**Abbildung 7.3:** Das ABEL-Makro `CONTR` übernimmt die Dekodierung der VME-Befehle für das Reset der FPGAs und die Umschaltung zwischen CPU- und Legacy-Modus

Das ABEL-Makro `CONTR` dekodiert die an `BUSADI[9:2]` anliegenden Adressen und stellt das Signal `CPU_ENA` zur Verfügung, je nachdem, welcher Wert an `BUSDATI` anliegt. Das Signal `CPU_ENA` steuert dann zum Beispiel den Multiplexer `MUXRESET`, der entweder die ebenfalls von `CONTR` dekodierten, über VME gesendeten Reset-Signale für die einzelnen FPGAs weitergibt, oder aber die von der CPU generierten (siehe unten).

Liegt an `BUSDATI` der Wert `2` an, wird das Signal `CNTR_RES` angelegt, welches an die fest in der FPGA-Hardware vorliegende `STARTUP`-Logik geht und ein Reset des ganzen FPGAs auslöst. Dies geschieht, wenn man den VME-Befehl `vme_write e0xx0800 2` (Reset) gibt.

### 7.4.2 RAM-Zugriff

Wie bereits in Kapitel 6.2 erwähnt, war die Implementierung des RAM-Zugriffs von aussen auf das CATCH die erste große Herausforderung des Projekts, da die CPU prinzipiell ständig auf das RAM zugreifen muss. Als Ausweg wurde schliesslich das Umschalten zwischen CPU-Modus, in dem die CPU exklusiven Zugriff auf das RAM besitzt, und “legacy-Modus”, in dem von aussen auf das RAM zugegriffen werden kann, gewählt.

In der frühen Phase des Projekts wurde aber zunächst ein anderer Weg beschritten. Da die Ansteuerung des RAMs so grundlegend ist und von vornherein nicht klar war, ob der CPU-Kern ohne weiteres auf einem neuen FPGA laufen würde, wurden zwei getrennte Designs verwendet. Das erste bestand aus dem modifizierten Original-Controller Design, das um das RAM-Interface erweitert wurde, das andere Design bestand aus dem mehr oder weniger unveränderten XSOC-Projekt mit lediglich kleinen Anpassungen (Pinout). Um zu testen, ob ein Programm überhaupt ausgeführt werden konnte, musste es erst mit dem ersten Design in das RAM geladen werden, dann musste das zweite Design in den FPGA programmiert werden, woraufhin die XSOC-CPU startete und das Programm abarbeitete. Um die Ergebnisse des Programms zu sehen (korrekte Funktion einmal vorausgesetzt),

musste man erst wieder das erste Design (Controller mit RAM-Zugriff) laden. Erst nachdem auf diese – zugegebenermaßen umständliche Weise – ein Nachweis der prinzipiellen Funktion des CPU-Kerns auf dem CATCH erbracht worden war, wurde die Vereinigung beider Designs (alter Controller plus RAM-Zugriff und XSOC-CPU-Kern) unter Benutzung der oben beschriebenen Möglichkeit zur Umschaltung zwischen beiden, vorgenommen.

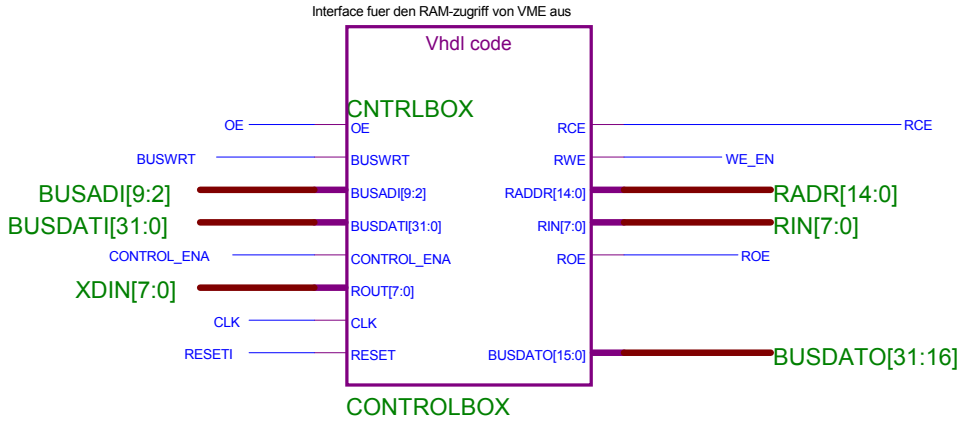



**Abbildung 7.4:** Das VHDL-Makro `CONTROLBOX` implementiert die Schnittstelle, mit der man über den VME-Bus auf das RAM zugreifen kann.

Implementiert wurde der RAM-Zugriff als VHDL-Makro, wie es in Abb. 7.4 dargestellt ist. Wenn an `BUSADI[9:2]` die Adresse für den RAM-Zugriff anliegt, führt das Makro einen Lese- oder Schreibzugriff auf das RAM durch. Dafür werden `RADR` als Adressleitungen und `RIN` bzw. `ROUT` als Datenleitungen verwendet. Das Ergebnis eines Lesezugriffs wird auf `BUSDATO` und damit auf den VME-Bus gegeben.

### 7.4.3 FPGA-Bus

Das Busprotokoll des CATCH-internen Bus ist stark mit dem VME-Bus verwandt (vgl. Abb. 7.5). Es besitzt die Steuerleitungen `BUSENA, BUSOE, BUSRDY`, die achtbittigen Adressleitungen `BUSAD` sowie die sechzehnbittigen Datenleitungen `BUSDAT`. Die Signale `BUSENA` und `BUSOE` sind jeweils für jeden der sieben anzusprechenden FPGAs einmal vorhanden.

Aufgabe des neuen Bus-Interfaces ist es, Lese- und Schreibzugriffe, die die CPU auf den FPGA-Bus ausführen will, auf dieses Busprotokoll umzusetzen. Die Implementierung der Logik für das Businterface kann man in Abbildung 7.6 sehen. Die Information, welcher FPGA angesprochen werden soll, wird über die Address-Next-Leitung `AN[15:0]` von der CPU festgelegt, indem man einen Lese- bzw. Schreibzugriff auf die Memory-gemappte Speicheradresse ausführt. In der Funktionseinheit `MEMCTRL` werden die Adressen verarbeitet. Das höchste Bit `AN15` entscheidet wie immer darüber, ob ein echter Speicherzugriff vorliegt (`AN15` = Null) oder ein Zugriff auf einen Peripheriebaustein erfolgen soll. Die Bits `AN14` bis `AN12` entscheiden, welcher Peripheriebaustein gemeint ist. Für das Businterface sind die Werte `001` (Lesen) und `011` (Schreiben) reserviert. Diese Information wird im `MEMCTRL` dekodiert und auf acht einzelne Signale `IOSEL[7:0]` umgesetzt, die die On-chip-Peripherie ansteuern. Für den FPGA-Bus wird das Signal `IOSEL1` als Enable-Signal für den

Lesezugriff verwendet, `IOSEL3` aktiviert den Schreibzugriff. Diese beiden Signale werden einerseits dazu verwendet, die Untereinheit `BUSINTERFACE` zu aktivieren, andererseits werden damit auch die `BUSENA`, `BUSWRT` bzw. `BUSOE`-Signale erzeugt.

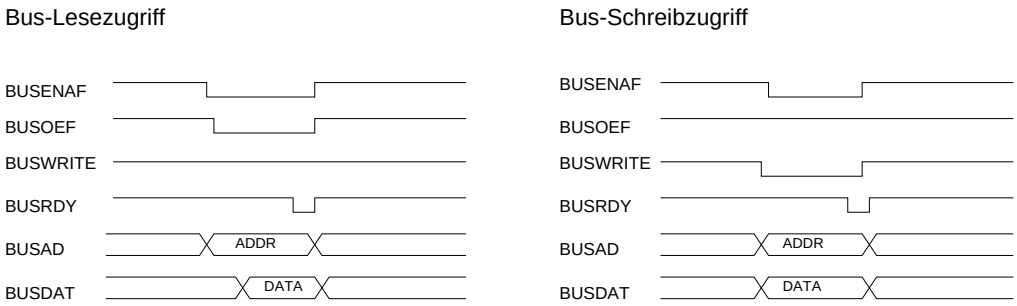



**Abbildung 7.5:** Busprotokoll des internen CATCH-Bus zur Kommunikation zwischen Control-FPGA und den anderen FPGAs

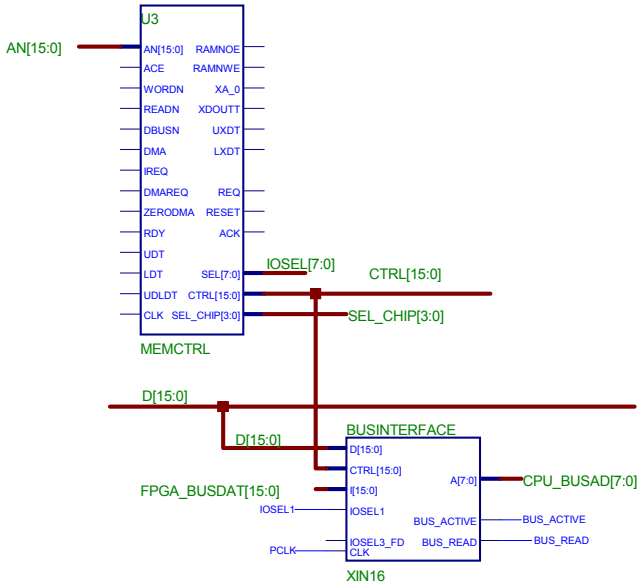



**Abbildung 7.6:** Implementierung des Businterfaces von der CPU aus

### 7.4.4 FPGA-Reset

Die Implementierung der Resets für die FPGAs ist sehr einfach. Wird der Peripheriebaustein `XRESET` durch das dafür vorgesehene Signal `IOSEL4` aktiviert, wird der Inhalt des Prozessorbusses auf den Bus `FPGA_R[15:0]` ausgegeben. Dieser geht über den Multiplexer `MUXRESET` in Abb. 7.3 direkt zu den entsprechenden IO-Pads des FPGA.

### 7.4.5 Display

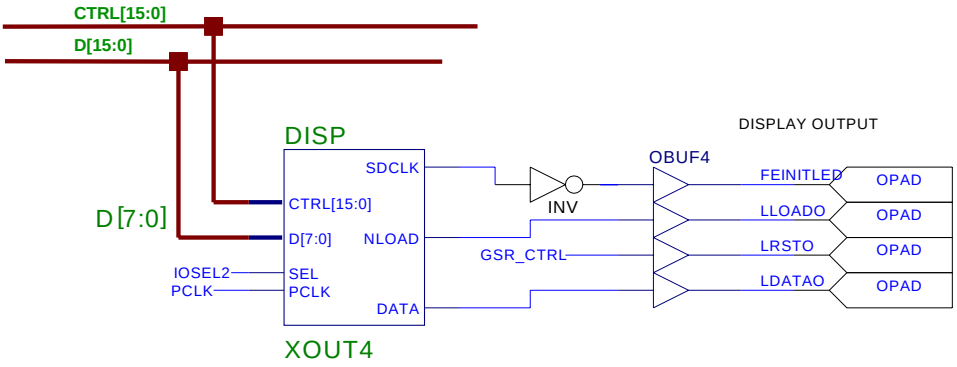



**Abbildung 7.7:** Schnittstelle vom Prozessor zum Display

Abbildung 7.7 zeigt, wie das Display implementiert wurde. Der Bus `D[7:0]` ist ein Teil des 16-bittigen Datenbus des Prozessors. Er enthält die acht Bit Daten, die die Information für das Display enthalten. Sie werden im Modul `DISP` von parallel nach seriell gewandelt und über die Leitung `LDATAO` ausgegeben. Dieses Modul wurde in VHDL implementiert und ist ein relativ simpler parallel-nach-seriell-Wandler. Der Bus `CTRL[15:0]` überträgt Steuerinformationen und das Signal `IOSEL2` dient als Write-Enable für das Display. Ein Hardware-Reset des Displays wird bei jedem Reset von XCONTROL ausgeführt (Signel `GSR_CTRL`). Nach jedem übertragenen Byte muss laut Datenblatt des Displays mindestens 600 ns lang gewartet werden. Bei einer Taktfrequenz von 10 MHz, mit der der Prozessor betrieben wird, sind das also 6 Clockzyklen. Das folgende Beispiel verdeutlicht, wie das Display komplett beschrieben wird.

```
int d[25]={0xc0, 0xa0,0x10,0x28,0x44,0x62,0x81, // clear, character 0
                 0xa1,0x1f,0x30,0x5e,0x70,0x90, //  char. 1
                 0xa2,0x0e,0x33,0x55,0x79,0x8e, //  char. 2
                 0xa3,0x01,0x22,0x44,0x68,0x90};//  char. 3
int *adr;

adr=(int *)0xa000; //Speicheradresse des Displays

    for(i=0;i<25; i++)
    {
      *adr=d[i];
      for(j=1;j<6;j++); // warten
    }
```

### 7.4.6 Serielles Interface

Da `xr16` nur ein 16-Bit-Prozessor ist, an das serielle Interface aber 24 Bit übertragen werden müssen, ist mehr als ein Schreibzugriff notwendig.

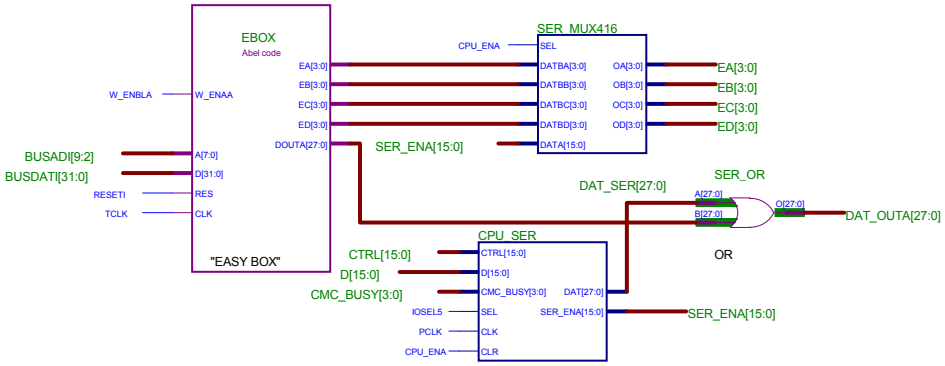



**Abbildung 7.8:** Implementierung des seriellen Interfaces. Die Parallel-nach-seriell-Wandler sind hier nicht abgebildet

Die serielle Schnittstelle besitzt deshalb drei verschiedene RAM-Adressen. Sie werden dem Peripheriebaustein über `CTRL[15:0]` mitgeteilt. Die erste Adresse ist für ein 8-bittiges internes Register für die ersten 8 Bit der 24 bit Daten, die zweite für das Register für die restlichen 16 Bit. Die 24 Bit Daten werden um je zwei Start- und Stopbits ergänzt und an `DAT_OUTA[27:0]` angelegt. Die Register sind im Schematics-Makro `CPU_SER` in Abb. 7.8 untergebracht. Die Ansteuerung des seriellen Interfaces vom VME-Bus aus befindet sich in `EBOX`.

Wird an die dritte Adresse ein 16-Bit-Wort geschrieben, so wird dies als Enable für die seriell-nach-parallel-Wandler verwendet, die die Daten an die sechzehn Ports des CATCH verschicken. Die Enable-Signale von der CPU und vom herkömmlichen seriellen Interface werden im Multiplexer `SER_MUX416` umgeschaltet und auf `EA[3:0]` bis `ED[3:0]` gelegt, die als Enable-Signale für die eigentlichen parallel-nach-seriell-Wandler dienen.

## 7.5 Design-Software

Als Design-Software wurde das `Foundation 3.1i`-Paket der Firma XilinX verwendet [36, 37]. Die Software übernimmt den kompletten Designfluß von der Erstellung eines Designs mit einer HDL oder dem Schematics-Editor, der Erzeugung von Netzlisten und deren Verarbeitung bis hinunter zum fertigen FPGA-Design, das sich in Hardware ausprobieren lässt (siehe auch [14]).

Der “Projektmanager” (Abb. 7.9) stellt eine grafische Oberfläche bereit, unter der sich die einzelnen Schritte ausführen lassen. Links sind die zum Projekt gehörigen Dateien dargestellt, rechts ein Flussdiagramm des Designablaufs mit den Implementierungsschritten. Die Teilprogramme, die für die Schritte verwendet werden, stammen oft von Drittanbietern. Einige stammen noch aus der 16-Bit (MS-DOS)-Ära, so dass die Designsoftware in der 16-Bit-Emulationsumgebung von Windows NT (WOWExec) laufen muss, was die Stabilität der Software nicht gerade erhöht.

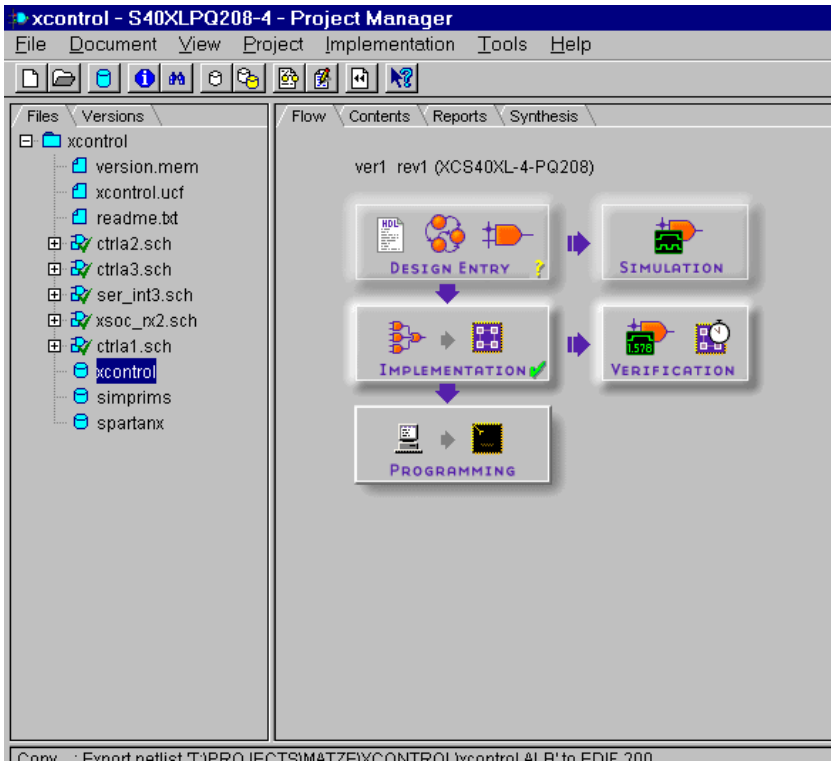



**Abbildung 7.9:** Unser treues Werkzeug: Der Xilinx Projektmanager

### 7.5.1 Entwurf und Implementierung

Die Erstellung der Designs kann in einer HDL wie VHDL [38] oder ABEL durchgeführt werden, wozu ein relativ komfortabler HDL-Editor im Foundation-Paket enthalten ist. Da aber sowohl die bisher benutzte Controller-Version als auch das XSOC-Projekt in grafischer Form, also als “Schematics”-Datei, vorlag, wurde hauptsächlich mit dem grafischen Editor, dem sogenannten “Schematics-Editor”, gearbeitet. Abbildung 7.10 zeigt einen Screenshot des Schematics-Editor.

Nach dem Erstellen des Designs gibt es die Möglichkeit, eine Simulation auf funktionaler Ebene durchzuführen (Punkt “Simulation in Abb. 7.9). Ist diese zur Zufriedenheit verlaufen, kann man die Übersetzung des Designs auf die FPGA-Ebene vornehmen (“Implementation”). Hierzu dient die Flow-Engine (Abb. 7.11).

Sie führt ihrerseits wieder verschiedene Teilschritte durch. Dazu gehören das Placement, d.h. die Verteilung der Logik auf die zur Verfügung stehenden Logikblöcke des FPGAs. Der zeitintensivste Schritt ist jedoch das Routing, d.h. die Verbindung der Logikblöcke untereinander über die Routing-Matrizen des FPGAs. Dies muss unter Berücksichtigung vorgegebener Randbedingungen (“Constraints”) geschehen. Dazu zählen die vorgegebene Pinbelegung des Chips und die gestellten Timinganforderungen. Nach Durchlaufen dieses Schrittes werden auch die Timing-Informationen für die spätere Simulation erstellt. Sind die Constraints erfüllt worden, (Abb. 7.12) wird eine Binärdatei er-

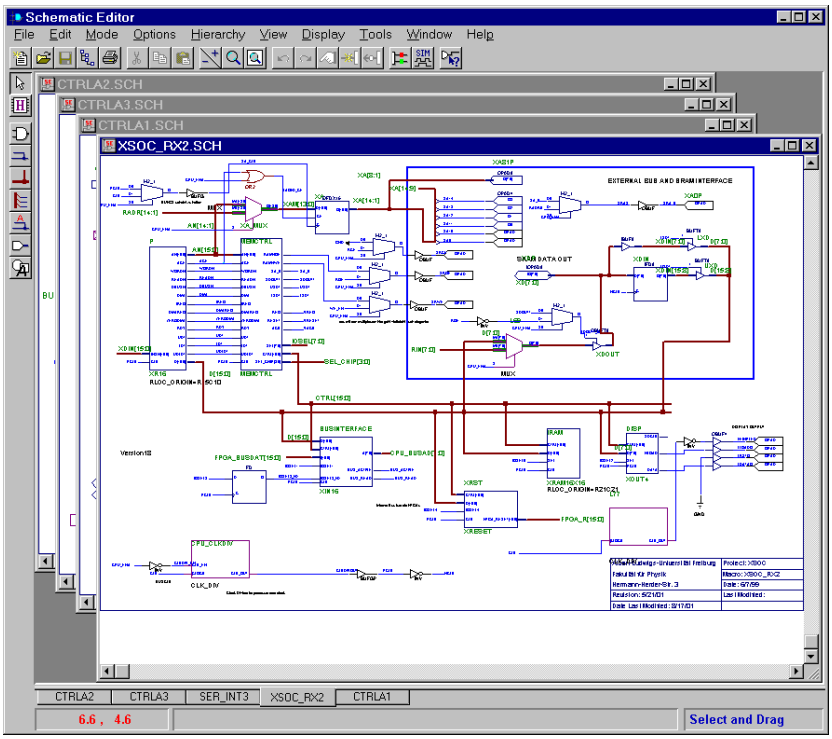



**Abbildung 7.10:** Der Schematics-Editor mit geöffnerem `XCONTROL`-Projekt

zeugt, die in den FPGA programmiert werden kann.

Mehrere dieser Binärdateien können mit dem “Prom-File-Formatter” zu einer `.hex`-Datei zusammengefasst werden, die zur Programmierung mehrerer FPGAs – beispielsweise auf dem CATCH – dienen kann.

Ist der Implementierungsschritt erfolgreich gewesen, kann man eine Timing-Simulation machen (Abschnitt 7.6.1)

Im sogenannten Floorplanner kann man das implementierte Design anschauen und die Platzierung einzelner Blöcke verändern. In (Abb. 7.13) dargestellt sind die belegten Logikblöcke des FPGAs, deren verschiedene Funktionen farblich gekennzeichnet ist. Über ein Menüsystem kann man die belegten Logikblöcke den Bausteinen im Schematics-Editor zuordnen. Stellt man Schwachstellen in der von der Flow-Engine automatisch erzeuten Platzierung fest, kann man diese auch manuell festlegen. Dies setzt aber ein sehr gutes Verständnis der Funktionsweise des FPGAs voraus.

### 7.5.2 Probleme

Eines der großen Probleme bei der Implementierung von `XCONTROL` mit der XilinX `Foundation`-Software war das Routing. Wie man in Bild 7.13 sehen kann, gibt es in `XCONTROL` zwei regelmässige Blöcke. Am unteren Rand ist die CPU selbst zu sehen. Die strukturierte Anordnung der Blöcke wird nicht automatisch von `Foundation` erzeugt, sondern wurde von Jan Gray mit den oben an-

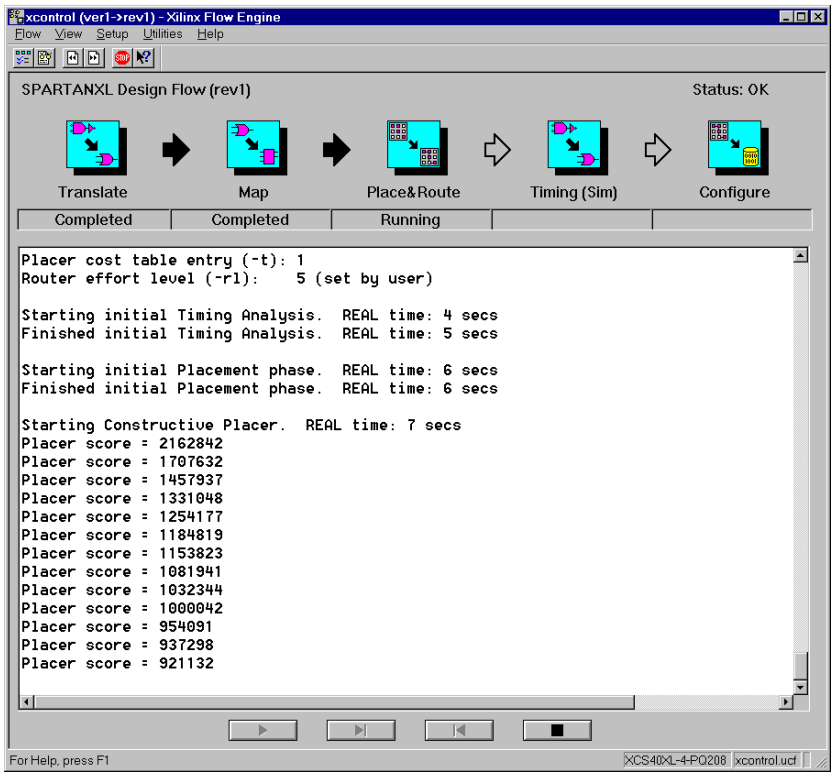



**Abbildung 7.11:** Placement-Schritt der XilinX Flow-Engine

gesprochenen `RLOC`-Constraints vorgegeben. Die zweite regelmäßige Struktur sind die Parallel-nach-seriell-Wandler der seriellen Schnittstelle. Sie liegen als fertig geroutete Bausteine (“cores”) in einer mitgelieferten Bibliothek vor. Durch diese handoptimierte Anordung der Funktionen auf dem FPGA lässt sich ein sehr gutes Timingverhalten erzielen. Der Nachteil dabei ist, dass die grossflächig vergebenen Gebiete den Routing-Algorithmus behindern. Dies spielt deshalb eine Rolle, weil die Input- und Output-Pads von `XCONTROL` bereits durch die existierende CATCH-Hardware vorgegeben waren. Beim Routing von `XCONTROL` traten teilweise massive Routingprobleme auf. Die Flow-Engine stoppte mit Fehlermeldungen, dass ein routen der Signale unmöglich sei, obwohl nur ca. 65 % der Chipfläche belegt waren. Dies ist vermutlich darauf zurückzuführen, dass die homogenen Teile des Designs (CPU, core, Seriell-Wandler) den Verbindungen der Logik zu den IO-Blocks im Wege standen. Dieses Problem konnte nur durch das Verschieben des CPU-Cores auf dem Chip mit Hilfe der `RLOC-ORIGIN`-Constraints umgangen werden.

## 7.6 Simulation

Um zu sehen, ob das implementierte Design auch funktioniert, wurden Timing-Simulationen mit der `Foundation`-Software am PC ausgeführt. Für die Timing-Simulation werden bei der Implementie-

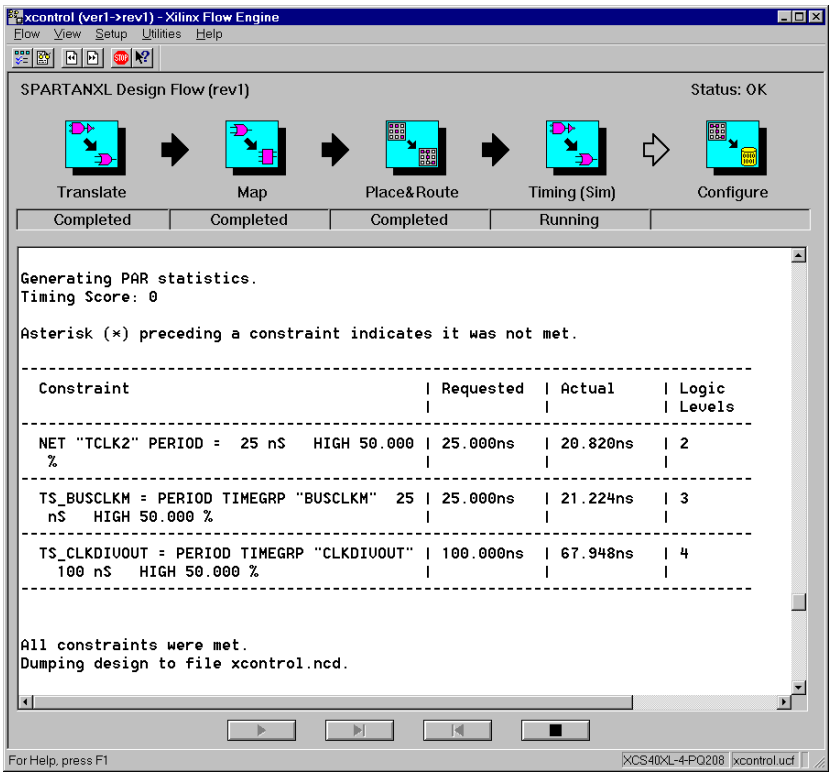



**Abbildung 7.12:** Zehn Minuten später: Geschafft, alle Timing-Constraints erfüllt

rung spezielle Netzlisten erzeugt, die Worst-case-Annahmen über die Schalt- und Signallaufzeiten im FPGA enthalten.

### 7.6.1 Timing-Simulation

In Abbildung 7.14 sind einige Takte des Mikroprozessors simuliert worden. Benutzt wurde der bei `Foundation` mitgelieferte Logik-Simulator der Firma Aldec. In der linken Spalte sind die Namen der beobachteten Signale aufgelistet, rechts ihre Wellenformen. `CLK` ist Clocksignal auf dem CATCH, `PCLK` ist der heruntergeteilte Takt für den Prozessor (10 MHz). Auf dem Bus `XD` liegen die Daten aus dem RAM an. Das Register `P/C/OP` ist die Decode-Stufe der Pipeline, während `P/C/EXOP` ist die Execute-Stufe der Pipeline ist. Der Bus `D[15:0]` ist der Prozessor-Datenbus.

### 7.6.2 Skripte

Die Simulation eines Mikroprozessors ist vergleichsweise aufwändig: Es müssen sehr viele Signale beobachtet werden und komplizierte Befehlsfolgen dem Prozessor zur Verarbeitung zur Verfügung gestellt werden. Ideal wäre die Simulation des kompletten Prozessors inklusive VME-Bus und RAM, aus dem die Instruktionen geholt und Daten geschrieben werden können. Dies wäre zum Beispiel

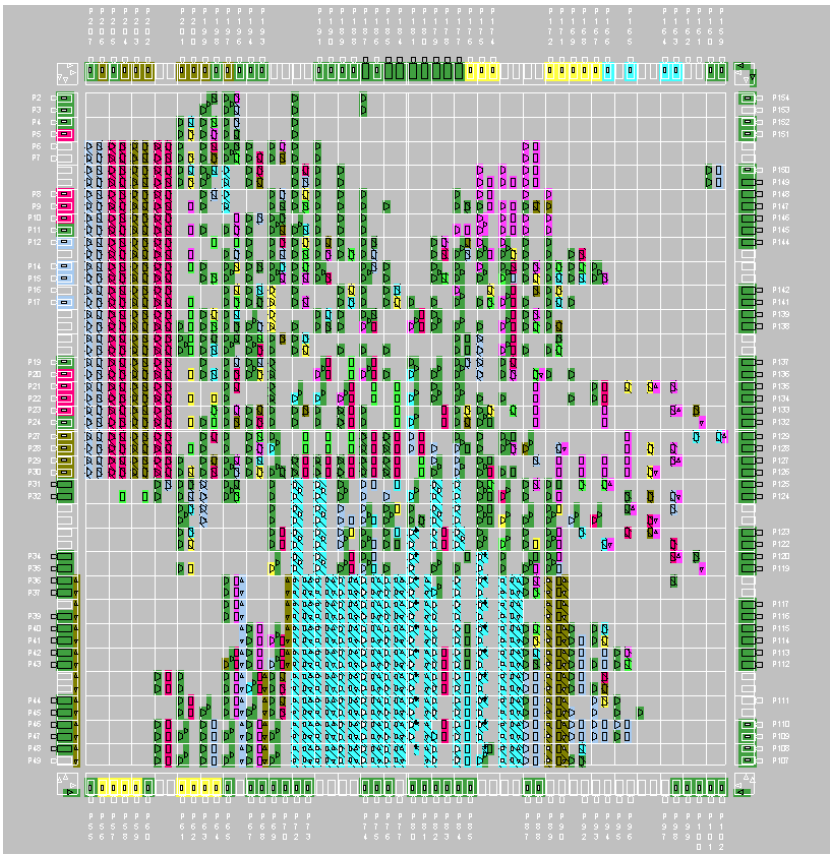


**Abbildung 7.13:** Das fertige Design im Floorplanner, unten erkennt man die regelmässige Struktur des `xr16`-Kerns, links oben die serielle Schnittstelle

mit einer VHDL-basierten Simulationssoftware möglich, mit der man Modelle für das RAM und den VME-Bus direkt in der Hardwarebeschreibungssprache VHDL schreiben und dann das Gesamtsystem aus CPU, RAM und Bus simulieren kann (sogenannter “Testbench”). Ein solches Simulationspaket ist beispielsweise von der Firma Mentor Graphics erhältlich.

Da die `Foundation`-Software keine Simulation mit Testbenches auf VHDL-Ebene unterstützt, musste die mitgelieferte Möglichkeit der Timing-Simutation benutzt werden, mit der nur das entworfene Design selbst simuliert werden kann. Zur Simulation kann man aus einer Liste mit allen Signalen, die verwendet wurden, diejenigen wählen, für die man sich interessiert. Die Signale kann man dann von Hand auf einen bestimmten Wert legen, dann einige Clockzyklen simulieren und das Ergebnis beobachten.

Dies ist jedoch viel zu aufwändig, um längere Simulationen durchzuführen. Der Aldec Logik-Simulator unterstützt jedoch eine Skriptsprache, mit der man den Simulationsvorgang automatisieren kann. Ein Beispiel für ein solches Simulationsskript ist in Anhang B.2 gegeben. Die Opcodes des zu simulierenden Programms werden aus der vom XSOC-Compiler erzeugten Binärdatei entnommen. Sie werden einzeln in den Bus `XD[7:0]` eingefüttert.

Dabei sind zwei Schritte je 16-Bit Opcode auszuführen, da das RAM nur 8-Bit breit ist. Bis man

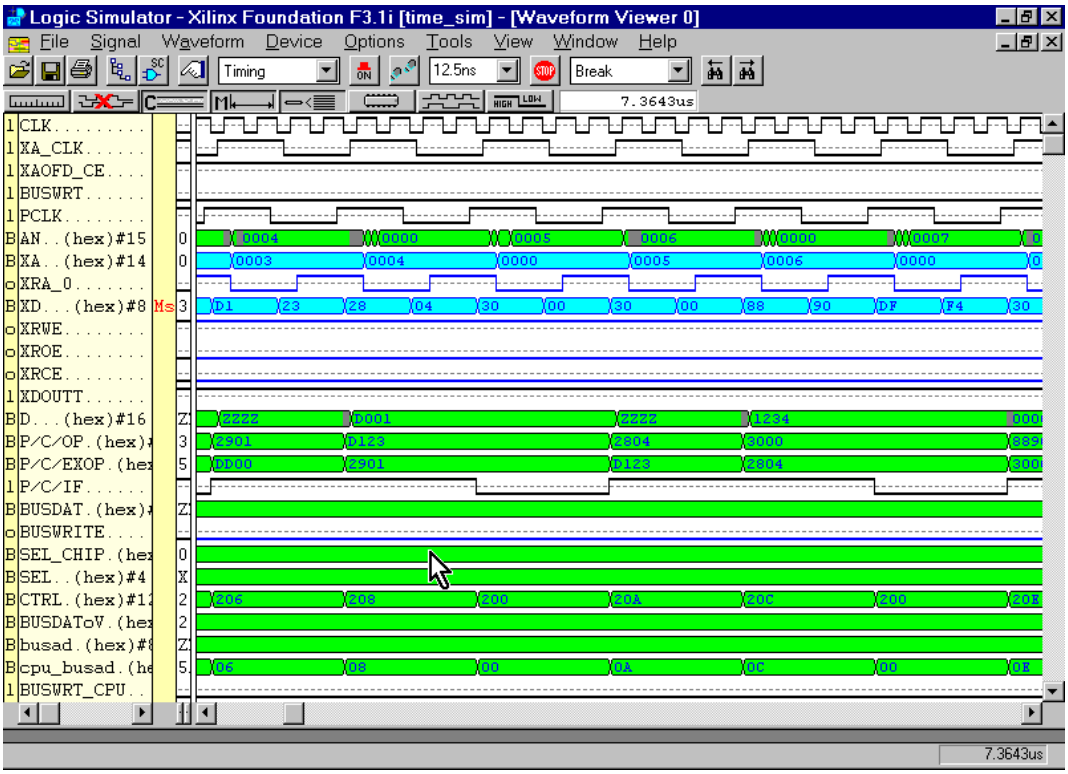



**Abbildung 7.14:** Timing-Simulation des `XCONTROL`-Projekts

zu der eigentlich interessierenden Stelle des Programms gelangt sind viele Opcodes zu simulieren, was die benutzten Skripts sehr lang und unübersichtlich macht.

# Kapitel 8

# Fazit

Ziel dieser Arbeit war es, einen Mikroprozessor in das CATCH Modul zu integrieren, der autonom Steuer- und Überwachungssoftware ausführen kann. Durch die Integration eines fertigen CPU-Kerns in das vorhandene Controller Design war es möglich, innerhalb kurzer Zeit ein System umzusetzen, das die gestellten Erwartungen erfüllt. Erneut zeigte sich der Vorteil des Einsatzes von rekonfigurierbarer Logik auf dem CATCH, da keine Änderungen an der Hardware nötig waren, um einen 16-Bit-RISC-Mikroprozessor zu integrieren.

Die größte Schwierigkeit bei der Implementierung des Projektes war die Integration des neuen CPU-Kerns unter Beibehaltug der bisherigen Funktionen des Control-FPGA. Dazu war zunächst eine Einarbeitung in das vorhandene Design notwendig, außerdem mussten einige relativ umständliche Lösungswege gewählt werden um die Abwärtskompatibilität zu gewährleisten. Die beschriebenen Routingprobleme sind zum Teil darauf zurückzuführen.

Recht gewöhnungsbedürftig war auch die Designsoftware von XilinX. Der Hardwareentwurf und die Simulation wären durch bessere Software sicherlich zu beschleunigen gewesen.

Insgesamt hat sich `XCONTROL` gut bewährt und wurde auch am CERN erfolgreich getestet. Durch die Verfügbarkeit eines angepassten Assemblers und C-Compilers ergeben sich flexible Programmiermöglichkeiten, so dass in Zukunft neue Funktionen implementiert werden können.

# Anhang A

# Anwendung von XCONTROL

## A.1 Programmierung des CATCH

Bevor der Prozessor benutzt werden kann, muss natürlich das CATCH programmiert werden. Dazu dient der Befehl `progcatch`. Als Design kann zum Beispiel `XCTRL_26.hex` verwendet werden[1]. Ein Beispiel für die Anwendung dieses Befehls ist:

```
progcatch /users/hoffmann/catch/XCTRL_26.hex a
```

Hierbei gibt der Parameter `a` am Ende des Befehlsaufrufs die Nummer des zu programmierenden CATCH an (hexadezimale Schreibweise). Diese Nummer kann man herausfinden, indem man auf dem VME-Rechner den Befehl `vme_check` ausführt, der alle Nummern der im Crate steckenden CATCH-Platinen samt der Revisionsnummer der gesteckten Mezzanine-Karten anzeigt.

## A.2 Ein Programm ins RAM schreiben

### A.2.1 Hex2ram

Das Programm `hex2ram` dient dazu, die mit `lcc-xr16` bzw. mit dem Assembler `xr16` erstellte Programme (binaries) ins RAM auf dem CATCH zu schreiben sowie den Speicher blockweise auslesen zu können[2].

**Ein Programm in den Speicher laden:** Die Syntax für das Laden eines auszuführenden Programmes lautet:

```
hex2ram catch_id -w filename.hex
```

Der erste Parameter (`catch_id` gibt die Seriennummer des zu programmierenden CATCH in hexadezimaler Schreibweise an. Der Parameter `-w` spezifiziert den Schreibzugriff, der letzte Parameter gibt das zu ladende File an.
Das Beispiel

[1] Dieses Design beinhaltet bereits die serielle Schnittstelle
[2] Dies hat nichts mit dem Block-Tranfer-Modus des VME-Bus zu tun!

```
hex2ram a -w /users/hoffmann/cpu/primsieb.hex
```

lädt das Binary `primsieb.hex` in das RAM auf CATCH Nummer 10.
Die Zieladdressen im RAM, an die das File geschrieben werden soll, sind im `.hex`-File selbst enthalten (Angaben zum Format s. Abschnitt A.4.2

**Blockweises Lesen:** Es ist auch möglich, den Inhalt des RAMs ab einer gewissen Speicherstelle anzeigen zu lassen.

```
hex2ram catch_id -r startaddresse bereich
```

Hierbei gibt der Parameter `-w` den Lesemodus an, der nächste Parameter gibt die Adresse der Speicherzelle an ab der der RAM-Inhalt ausgegeben werden soll. Der letzte Parameter gibt den zu lesenden Bereich an. Das Beispiel unten gibt also den Inhalt der Speicherstellen von `0x0600` bis `0x0620` aus.

Das Programm `hex2ram` erzeugt dann folgende Ausgabe:

```
vmefr02> ./hex2ram2b 3 -r 6000 20
Controller version: 27

Reading data
(6000)   : 0
(6002)   : 0
(6004)   : 0
(6006)   : 0
(6008)   : 0
(600a)   : 0
(600c)   : 0
(600e)   : 0
(6010)   : 0
(6012)   : 0
(6014)   : 0
(6016)   : 0
(6018)   : 0
(601a)   : 0
(601c)   : 0
(601e)   : 0
```

Zuerst wird die Versionsnummer des Control-FPGA-Designs gelesen, hier 27. Danach wird der RAM-Inhalt im spezifizierten Bereich angezeigt. Die linke Spalte gibt die RAM-Adresse an, die rechte den aktuellen Wert dieser Adresse.

## A.3 Initialisierung der Front-End-Karten

Das folgende Beispiel-Programm dient der Initialisierung von vier Front-End-Boards, die alle an der ersten HOTLink-Karte eines CATCH angeschlossen sind. Dazu werden 16 Datenworte verschickt. Die Worte werden aus dem RAM ab Adresse `0x6000` entnommen, wo sie vorher z.B. mit VME-Befehlen platziert wurden.

```
// Soll Daten aus dem Ram ab Adresse 0x6000 direkt
// an das serielle Interface schreiben

    unsigned short *ser_adr1;
    unsigned short *ser_adr2;
    unsigned short *ser_ena;
    unsigned short *ser_busy;
    unsigned short *ram_address;
    int i=1;
    unsigned short send_to=0x000f;      //an alle FE-Karten an CMC 1


void main() {
    ser_adr1=(unsigned short *)0xd001;  //Adressen des ser. Interfaces
    ser_adr2=(unsigned short *)0xd002;
    ser_ena=(unsigned short *)0xd004;
    ser_busy = (unsigned short *)0xd008;
    ram_address=(unsigned short *)0x6000;  //RAM- Adresse

   for(i=0;i<16;i++)
     {
    *ser_adr1=*ram_address;
    ram_address++;
    *ser_adr2=*ram_address;
    ram_address++;
    *ser_ena=send_to;
    while(*ser_busy != 0); //warten bis Busy-Flag 0
     }
   }
```

### A.3.1 F1toram

Zum Erzeugen der Konfigurationsdaten für die Freiburger Straw-Front-End-Karten oder die TDC-CMCs kann das Programm `f1conf` benutzt werden. Es erzeugt Files mit der Endung `.f1`. Diese können mit dem Programm `f1toram` in den RAM geschrieben werden. Für die Daten sind spezielle RAM-Bereiche vorgesehen. Ein auf der `XCONTROL`-CPU laufendes Programm, das dann Front-End-Karten initialisieren will, kann auf diese Daten dann zugreifen. Um entscheiden zu können, welche

Chips mit welchen Daten initialisiert werden sollen, dienen sechzehn Konfigurationsregister. Die Aufteilung des Speichers ist dann wie in Abbildung A.1.

Der Aufruf des Programms `f1toram` geschieht wie folgt:

```
> f1toram straws-1600.f1 4f 15
```

In diesem Beispiel wird das File `straws-1600.f1` in das CATCH mit der Nummer `4f` geladen, wobei das Front-End-Board an Port 15 initialisiert werden soll. Die Informationen welcher der acht TDC (oder alle auf einmal) die Informationen bekommt, steckt im `.f1`-File.

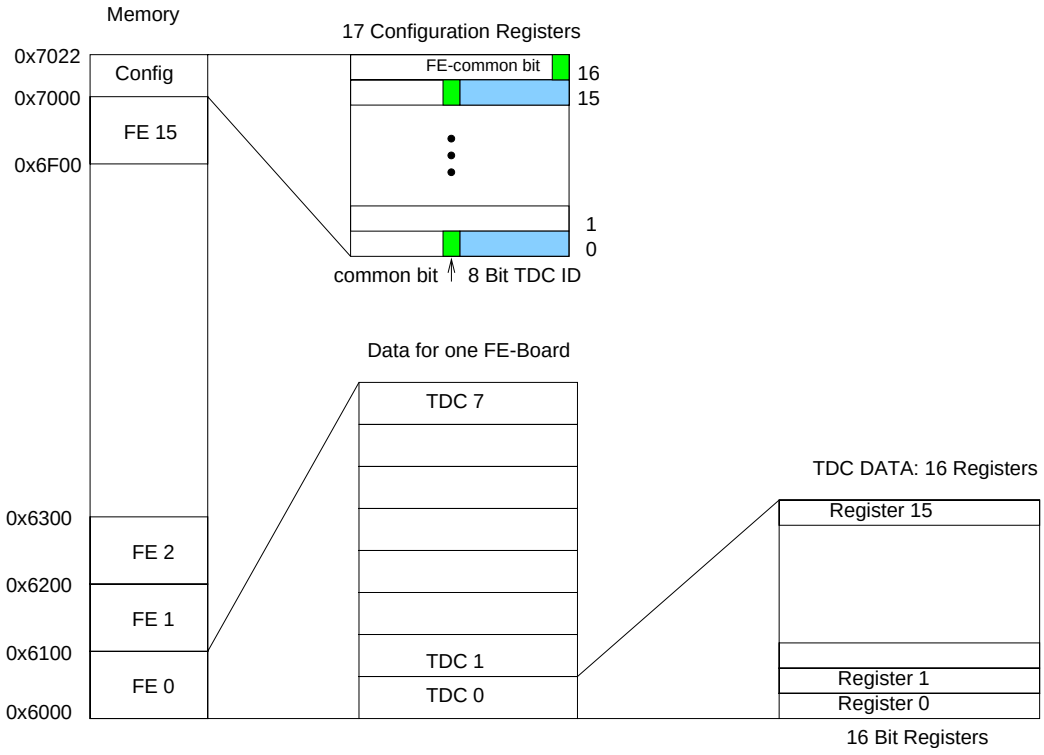



**Abbildung A.1:** Aufteilung des Adressraums für die Konfigurationsdaten

Weitere Informationen über `f1toram` sind im `XCONTROL`-User's-Manual [30] enthalten.

### A.3.2 `ramtoreg`

Das Programm `ramtoreg` läuft auf der `XONTROL`-CPU. Es liest die Daten aus den von `f1toram` wie in Abb. A.1 benutzten Speicheraddressen aus und schreibt sie an die Front-End-Karten. Zusätzlich wird die Source-ID auf dem Display angezeigt.

## A.4 Compiler

Das XSOC-Projekt bringt einen an das Instruction Set des xr16-Kerns angepassten C-Compiler und Assembler mit [42]. Beim C-Compiler handelt es sich um eine angepasste Version des `lcc` Compilers. Die Homepage des Projekts findet man hier [41].

### A.4.1 Installation

#### Windows

Nachdem man sich die Windows-Version (aktuell ist Version 4.1) von `lcc` besorgt hat, installiert man sie z.B. nach

```
C:\Programme\lcc\4.1
```

In einer Dos-Box setzt man die Umgebungsvariable `LCCDIR` auf das Verzeichnis, das die Binaries des `lcc` Compilers enthält:

```
set LCCDIR=\Progra~1\lcc\4.1\bin
```

Als nächstes kopiert man die im `XSOC.zip` im Unterverzeichnis `\lcc-xr16\bin` befindlichen Dateien `lcc-xr16.exe rcc-xr16.exe xr16.exe reset.s libxr16.s` nach `%LCCDIR%`

```
xcopy \xsoc\lcc-xr16\bin\*.* %LCCDIR%
```

#### Unix

Es gibt auch eine auf UNIX angepassten Portierung von LCC, allerdings mussten einige Veränderungen daran vorgenommen werden, um die Anpassungen an den `xr16` auch unter UNIX laufen lassen zu können. Vor allem der Assembler `xr16` musste angepasst werden. Compiler und Assembler stehen in einem `tar`-Archiv als Quellen zur Verfügung. Es existiert je eine angepasste Version für Linux und eine für Digital Unix (Alpha). Um das Programm `lcc-xr16` wie im nächsten Absatz beschrieben aufrufen zu können, muss man allerdings den korrekten Pfad zum Compiler setzen:

```
setenv LCCDIR ~/cpu/lcc/
```

### A.4.2 Compilieren

Um den Compiler benutzen zu können muss der aktuelle Pfad `%PATH%` das Verzeichnis `%LCCDIR%` enthalten.

Um nun ein C-Programm (z.B. primsieb.c) zu compilieren, muss man folgende Befehle eingeben:

```
lcc-xr16 -v -c primsieb.c

lcc-xr16 -v -o primsieb.hex -lst=primsieb.lst primsieb.o
```

Der erste Befehl erzeugt das Assembler-File primsieb.o, der zweite bindet die Bibliothek libxr16.s ein (Linker) und erzeugt ein ASCII-File `primsieb.hex`, das mit dem Tools `hex2ram` in das RAM des CATCH geladen werden kann.

**Format der `.hex`-Files**

Die vom Assembler `xr16` erzeugten `.hex`-Dateien sind ASCII Dateien. Sie sehen folgendermassen aus:

```
- 10 0000 D7 FF 2D 0E C0 01 A0 00 00 00 00 00 00 00 00 00
- 10 0010 2D DA 8A D0 8B D2 8C D4 2A 01 2C 01 2B 01 B0 02
- 10 0020 0B AC 0A C0 0C B0 DD 8F 20 B0 B8 F9 02 A0 5A D0
- 08 0030 5B D2 5C D4 2D D6 A0 F0
```

Jede Zeile wird durch einen Bindestrich " - " eingeleitet, danach folgt die Anzahl der Bytes in der Zeile in hexadezimaler Schreibweise. Die nächste, 16-bittige Zahl ist die Adresse, an die die Daten im RAM geschrieben werden sollen, und schliesslich folgen die eigentlichen Daten.

## A.5 Die Runtime-Library `libxr16`

Für jedes Programm bindet der Compiler, wenn er wie in Abschnitt A.4.2 aufgerufen wird, die Runtime-Library `libxr16` ein. Sie enthält einige wichtige Grundfunktionen für den Programmstart, sowie komplexe Funktionen wie Integer-Multiplikation (oder auch Division), die vom Compiler nicht direkt in Maschinenbefehle umgesetzt werden können. Die im Augenblick in `libxr16` enthaltenen Funktionen werden nun kurz erläutert.

### A.5.1 `zeromem`

Die Funktion `zeromem` löscht den gesamten Hauptspeicher bis zur Adresse `0x7FFF`, der Wert der mit der Funktion `_tos` (top of stack) zurückgegeben wird. Will man einen Teil des Hauptspeichers für Setup-Daten reservieren (siehe Abschnitt 6.7.3) muss die Funktion modifiziert werden, so dass nur bis zur Adresse `0x6000` ein Reset durchgeführt wird.

```
/* Reset memory to consistent known state up to 0x6000.*/
void _zeromem() {
        Word* p = &_end;
        Word* end = _tos();

        for ( ; p < 0x6000; ++p)
                *p = 0;
}
```

### A.5.2 `mulu2`

Da `xr16` keine Multiplikationseinheit besitzt, müssen Multiplikationen in Software ausgeführt werden. Die Funktion mulu2 führt eine einfache 16-Bit Integer-Multiplikation ohne Berücksichtigung eines eventuellen Overflow durch. Sie wird vom Compiler verlinkt, sobald eine Multiplikation des Typs unsigned Integer durchgeführt wird, die nicht durch einfache Shift- oder Additionsbefehle ersetzt werden kann.

Verwendet man in einem C-Programm Multiplikation mit Integern (mit Vorzeichen) so versucht der Compiler dies auf eine Funktion `_muli2` umzusetzen, die in der aktuellen Version der Runtime-Library (noch) nicht existiert und bricht mit einer entsprechenden Fehlermeldung ab.

### A.5.3 Listing

```
/* libxr16.c -- minimal xr16 runtime library
 *
 * Copyright (C) 1999, 2000, Gray Research LLC.  All rights reserved.
 * The contents of this file are subject to the XSOC License Agreement;
 * you may not use this file except in compliance with this Agreement.
 * See the LICENSE file.*/

typedef unsigned Word;

extern int main();
extern Word     _end;

void _zeromem();
Word* _tos();

/* Processor reset: zero memory and call main. */
int _reset() {
        _zeromem();

        main();

        /* dynamic halt */
        for (;;)
                ;
}

/* Reset memory to consistent known state.*/
void _zeromem() {
        Word* p = &_end;
        Word* end = _tos();

        for ( ; p < end; ++p)
                *p = 0;
}

/* Interrupts have not been tested.*/
int _interrupt() {
        return 0;
```

```
}

/* Too clever way to return address of approximately the top of stack;
 * ignore compiler warning on returning address of argument. */
Word* _tos(Word arg) {
        return &arg;
}

/* Multiply unsigned times unsigned.*/
unsigned mulu2(unsigned a, unsigned b) {
        unsigned w = 0;

        for ( ; a; a >>= 1) {
                if (a&1) w += b;
                b <<= 1;
        }
        return w;
}
/* This must be the last symbol in the last module "linked" into the load
 * image.*/
Word _end;
```

Vor Verwendung von `libxr16` muss das File `libxr16.c` compiliert werden

```
lcc-xr16 -v -S libxr16.c
```

Dies liefert unter Umständen zwei Compiler-Warnings wegen des bei der Top-Of-Stack-Abfrage verwendeten Tricks. Sie können getrost ignoriert werden. Das erzeugte Assembler-File `libxr16.s` muss dann noch in das `$LCCDIR`-Verzeichnis kopiert werden, um es benutzen zu können.

### A.5.4 `divi2` und `modi2`

Für Division und Modulo-Rechnung existiert in der ursprünglichen Version der `libxr16` kein Code. Der Assembler erwartet die Targets `divi2` und `modi2`, welche extra implementiert werden müssen. Da ich die Funktionen für ein Demo-Programm benötigte, habe ich sehr sehr einfache (=ineffiziente) Algorithmen verwendet. Andererseits sind sie speicherplatzsparend und bei kleinen Zahlen (wie im Demoprogramm) kommt es auf die Laufzeit sowieso nicht an. Trotzdem sollte die `libxr16` in Zukunft verbessert werden.

```
unsigned modi2(unsigned a, unsigned b) {
        unsigned w=0;
        w=a;
        while(w>=b)
           w=w-b;
        return w;
```

```
}

unsigned divi2(unsigned a, unsigned b) {
  unsigned w=0;
  while(a>=b)
  {
    a=a-b;
    w++;
  }
  return w;
}
```

# Anhang B

# Einige technische Einzelheiten

## B.1 Technisches zum Display

Das CATCH verfügt in der aktuellen Revision über ein Matrix-Display des Typs SCDV5540 (rot) bzw. SCDV5543 (grün) von Infineon (früher Osram) [32]. Es besteht aus vier Zeichen aus je $5 \times 5$ Pixeln, siehe dazu Abbildung B.1. Auf dem Display lässt sich jedes Pixel einzeln ansprechen, so dass im Prinzip beliebige Zeichen darstellbar sind. Die Ansteuerung des Displays erfolgt seriell, so dass XCONTROL zuerst einmal eine Parallel-nach-seriell-Wandlung vornehmen muss. Um das Display komplett zu beschreiben, müssen vier Zeichen übertragen werden. Jedes Zeichen besteht aus sechs Bytes, einer *character address* und fünf Bytes für die fünf Zeilen des Buchstabens (Tabelle B.1. Insgesamt müssen also 24 Byte an das Display übertragen werden, wie in Abbildung B.2 gezeigt ist.

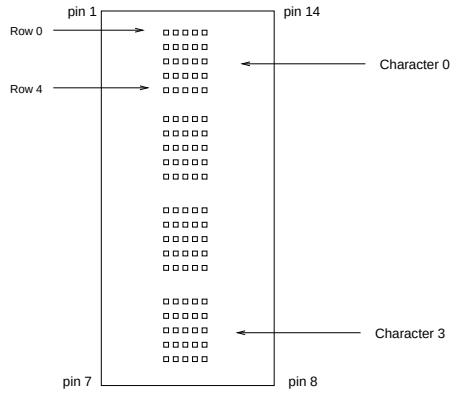



**Abbildung B.1:** Vier-Zeichen Matrix Display auf dem CATCH

Die Character-Address für jedes Zeichen wird wie in Tabelle B.1 nummeriert.

Da jede Zeile eigentlich nur fünf Pixel hat, werden die restlichen drei Bit als Opcodes verwendet. Eine Zeile (Row) sieht also wie folgt aus:

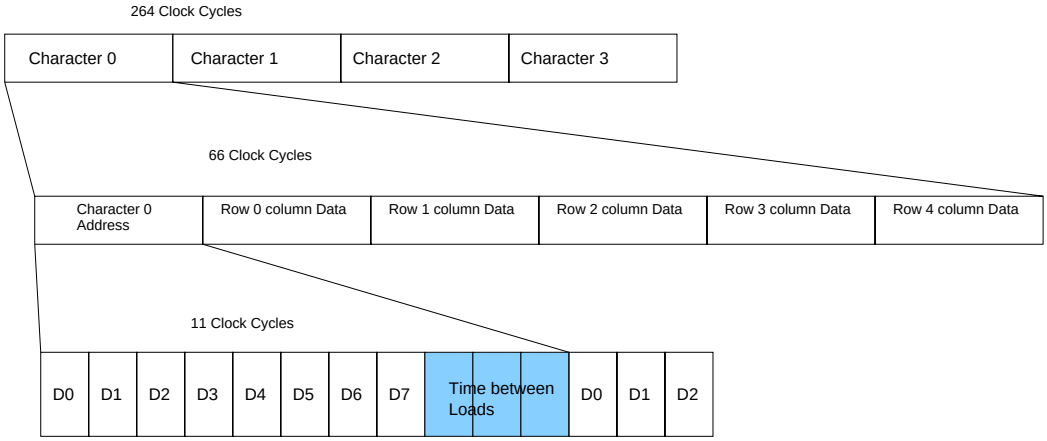



**Abbildung B.2:** Serielles Datenformat für das Display

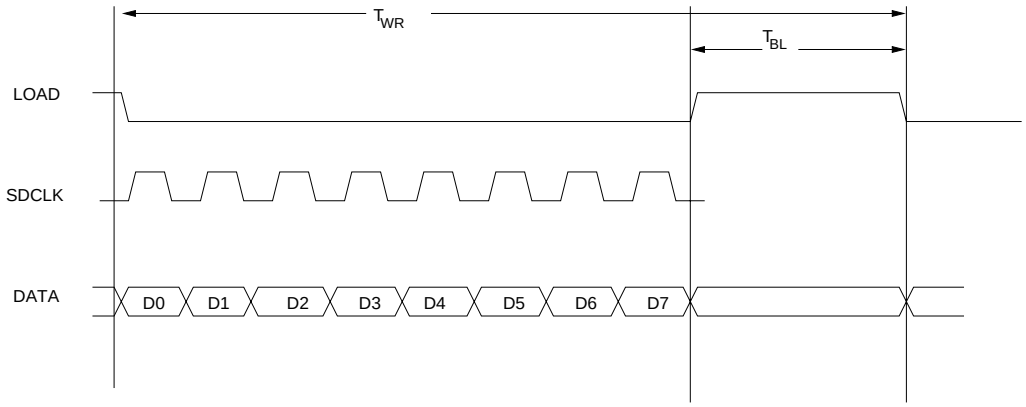



**Abbildung B.3:** Timing Diagramm für das Display

| Character Address |
|---|
| Row 0 data |
| Row 1 data |
| Row 2 data |
| Row 3 data |
| Row 4 data |

**Tabelle B.1:** Die Information eines Zeichens auf dem Display besteht aus einem Byte Character-Address und fünf Byte Daten für die fünf Zeilen

| Character No. | Character Address |
|---|---|
| 0 | A0 |
| 1 | A1 |
| 2 | A2 |
| 3 | A3 |

**Tabelle B.2:** Nummerierung der Character-Address

| bit 7-5 | bit 4-0 |
|---|---|
| Opcode | Data |

Die Opcodes für die Daten dienen dazu, die Nummer der Zeile zu kennzeichnen. Die erste Zeile hat den Opcode `000`, die zweite den Opcode `001` usw. Ein Beispiel für einen kompletten Buchstaben (hier “D”) lautet also folgendermaßen:

| | **Opcode** | **Data** | **hex** |
|---|---|---|---|
| Row 0 | `0 0 0` | `1 1 1 1 0` | 1E |
| Row 1 | `0 0 1` | `1 0 0 0 1` | 31 |
| Row 2 | `0 1 0` | `1 0 0 0 1` | 51 |
| Row 3 | `0 1 1` | `1 0 0 0 1` | 71 |
| Row 4 | `1 0 0` | `1 1 1 1 0` | 9E |

Die gängigsten Zeichen sind im Datenblatt des Displays tabelliert [32].

Ein weiterer wichtiger Befehl an das Display ist `C0`, was ein Soft-Reset des Displays veranlasst.

## B.2 Listing für die Timingsimulation

```
|-------------------------------------------------------------------------
| XCONTROL Cpu-Projekt
|
| Matthias Hoffmann 08.07.2001
|
| Dieses Demo fuehrt eine Schleife aus, die eine Variable (register 12)
| um eins erhoeht und an die Ram Addresse 0x0600 schreibt
| mit cpu_ena=0, erzwingt CPU Modus
|
| Initial settings
-------------------------------------------------------------------------

delete_signals
set_mode timing
restart
stepsize 10 ns


watch clk
|watch XA_CLK
|watch XAOFD_CE
|watch buswrt
watch pclk
vector AN AN[15:1] XRCE
vector XA XA[14:1] XRA_0 | getrennt fuer Adress und LSB
watch P/C/IF
|watch XRA_0
vector XD XD[7:0]
watch XRWE
watch XROE| XRCE

|watch softreset
vector D D[15:0]
vector P/C/OP P/C/OP[3:0] P/C/RD[3:0] P/C/RA[3:0] P/C/RB[3:0]
vector P/C/NIR P/C/NIR[15:0]
vector P/C/EXIR P/C/EXIR[15:0]
vector P/C/EXOP P/C/EXOP[3:0] P/C/EXRD[3:0] P/BRDISP[7:0]
watch xdoutt
vector P/RNA P/RNA[3:0]
vector P/D/A P/D/A[15:0]
vector P/RNB P/RNB[3:0]
vector P/D/B P/D/B[15:0]
vector P/RNN[
watch READN
watch P/BRANCH

watch P/C/EXANNUL
watch XDOUTT

| Clocks Definieren

stepsize 12.5ns

CLOCK BUSCLKM 1 0

vector CTRL CTRL[15:0]
```

```
|-----------------------------------------
| INIT


assign cpu_ena 0 | CPU Modus erzwingen

assign CENA 1
assign cwrite 1
assign coe 1
assign BUSRDY 1
assign buswrt 0
assign XD 2d\h
assign xdin 0000\h  | S t a r t u p
assign softres 0
cycle 2

assign softres 1
release xdin


cycle 1            | verzoegerung

|           0     1     2     3     4     5     6     7
| - 10 0000 2D DC 8B D0 8C D2 2C 01 D0 60 2B 00 B0 03 2C C1
|           8     9     A     B     C     D     E     F     10
| - 10 0010 09 C0 30 00 89 B0 B0 FA 5B D0 5C D2 2D D4 A0 F0
|
| ----------------- Programm ausfuehren


after 12 ns do assign XD 30\h
cycle 2


after 12 ns do assign XD 30\h
cycle 2
after 12 ns do assign XD 00\h
cycle 2    | IF='1'
after 12 ns do assign XD 30\h
cycle 2
after 12 ns do assign XD 00\h
cycle 2

after 12 ns do assign XD 8b\h         | Register retten
cycle 2
after 12 ns do assign XD d0\h
cycle 2
after 12 ns do assign XD 8c\h
cycle 2
after 12 ns do assign XD d2\h
cycle 2
after 12 ns do assign XD 30\h
cycle 2
after 12 ns do assign XD 00\h
cycle 2
after 12 ns do assign XD 2c\h | IF='1'
cycle 2
after 12 ns do assign XD 01\h
cycle 2
```

```
release XD
cycle 12


after 12 ns do assign XD d0\h
cycle 2
after 12 ns do assign XD 60\h
cycle 2
after 12 ns do assign XD 30\h | IF = 1
cycle 2
after 12 ns do assign XD 00\h
cycle 2

release XD
cycle 12


|           0     1     2     3     4     5     6     7
| - 10 0000 2D DC 8B D0 8C D2 2C 01 D0 60 2B 00 B0 03 2C C1
|           8     9     A     B     C     D     E     F     10
| - 10 0010 09 C0 30 00 89 B0 B0 FA 5B D0 5C D2 2D D4 A0 F0



after 12 ns do assign XD 2b\h
cycle 2
after 12 ns do assign XD 00\h
cycle 2
after 12 ns do assign XD 30\h | IF = 1
cycle 2
after 12 ns do assign XD 00\h
cycle 2
after 12 ns do assign XD b0\h
cycle 2
after 12 ns do assign XD 03\h
cycle 2
after 12 ns do assign XD 2c\h             |Hier beginnt das eigentliche Programm
cycle 2
after 12 ns do assign XD c1\h
cycle 2
after 12 ns do assign XD 30\h | IF = 1
cycle 2
after 12 ns do assign XD 00\h
cycle 2


after 12 ns do assign XD 09\h
cycle 2
after 12 ns do assign XD c0\h
cycle 2
after 12 ns do assign XD B0\h
cycle 2
after 12 ns do assign XD fa\h
cycle 2
after 12 ns do assign XD 30\h | IF='1'
cycle 2
after 12 ns do assign XD 00\h
```

```
cycle 2



after 12 ns do assign XD 5b\h
cycle 2
after 12 ns do assign XD d0\h
cycle 2
after 12 ns do assign XD 5c\h  | 0E
cycle 2
after 12 ns do assign XD d2\h
cycle 2
after 12 ns do assign XD 30\h | IF='1'
cycle 2
after 12 ns do assign XD 00\h
cycle 2



after 12 ns do assign XD 2c\h
cycle 2
after 12 ns do assign XD c1\h
cycle 2
after 12 ns do assign XD 09\h  |
cycle 2
after 12 ns do assign XD c0\h
cycle 2
after 12 ns do assign XD 30\h
cycle 2
after 12 ns do assign XD 00\h
cycle 2


after 12 ns do assign XD 30\h | IF='1'
cycle 2
after 12 ns do assign XD 00\h
cycle 2
after 12 ns do assign XD 89\h
cycle 2
after 12 ns do assign XD b0\h
cycle 2
after 12 ns do assign XD 30\h | IF='1'
cycle 2
after 12 ns do assign XD 00\h
cycle 2


after 12 ns do assign XD b0\h
cycle 2
after 12 ns do assign XD fa\h
cycle 2
after 12 ns do assign XD 5b\h
cycle 2
after 12 ns do assign XD d0\h
cycle 2
after 12 ns do assign XD 30\h | IF='1'
cycle 2
after 12 ns do assign XD 00\h
cycle 2
```

```
after 12 ns do release XD
cycle 12


after 12 ns do assign XD 5c\h  | 0E
cycle 2
after 12 ns do assign XD d2\h
cycle 2
after 12 ns do assign XD 30\h | IF='1'
cycle 2
after 12 ns do assign XD 00\h
cycle 2



| ---------------------------------- ab hier von vorne
|
exit
```

# Anhang C

# Schaltplan

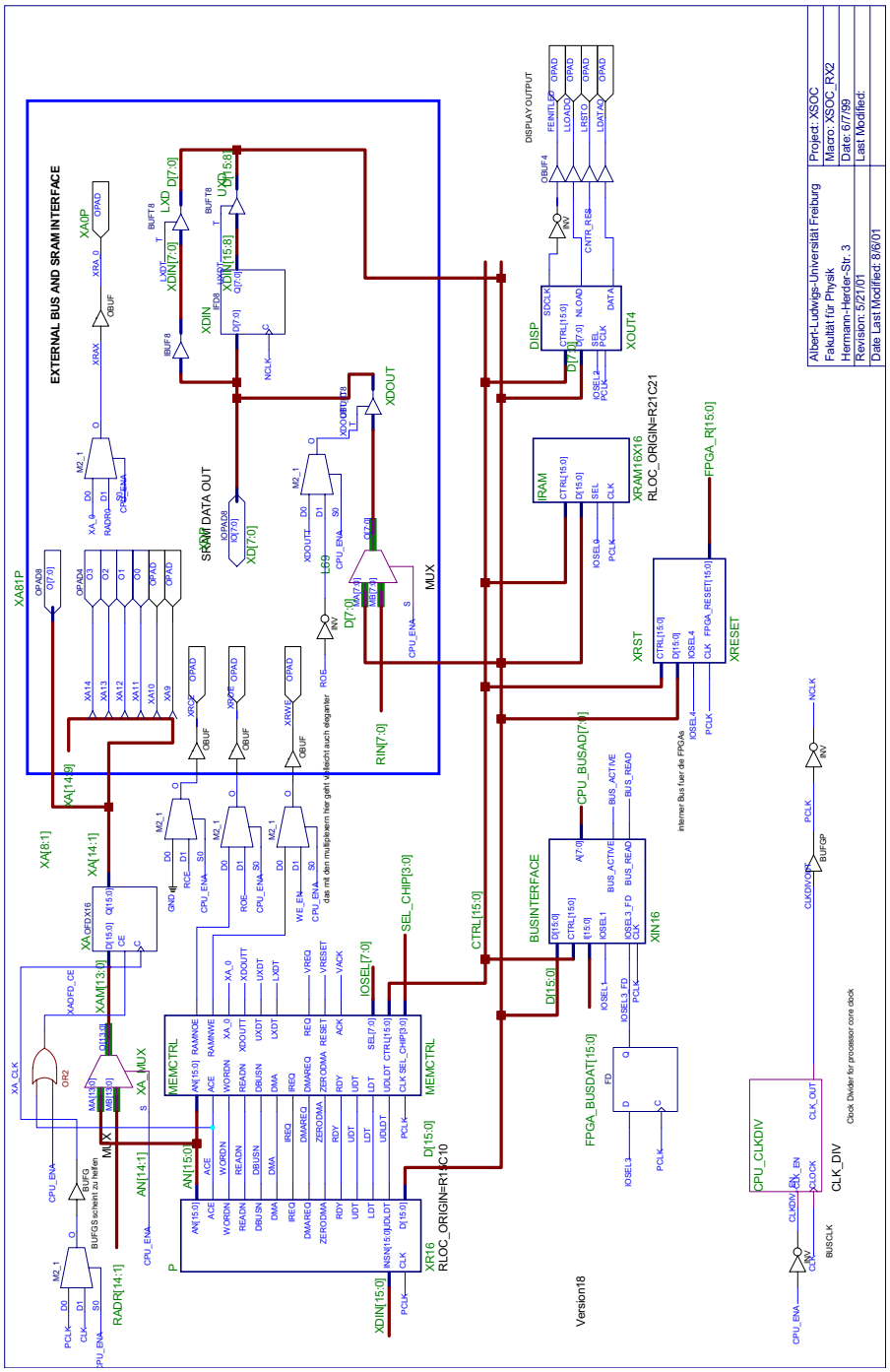



**Abbildung C.1:** Schematics Editor: Prozessor, Memory Controller und on-chip Peripheriebausteine

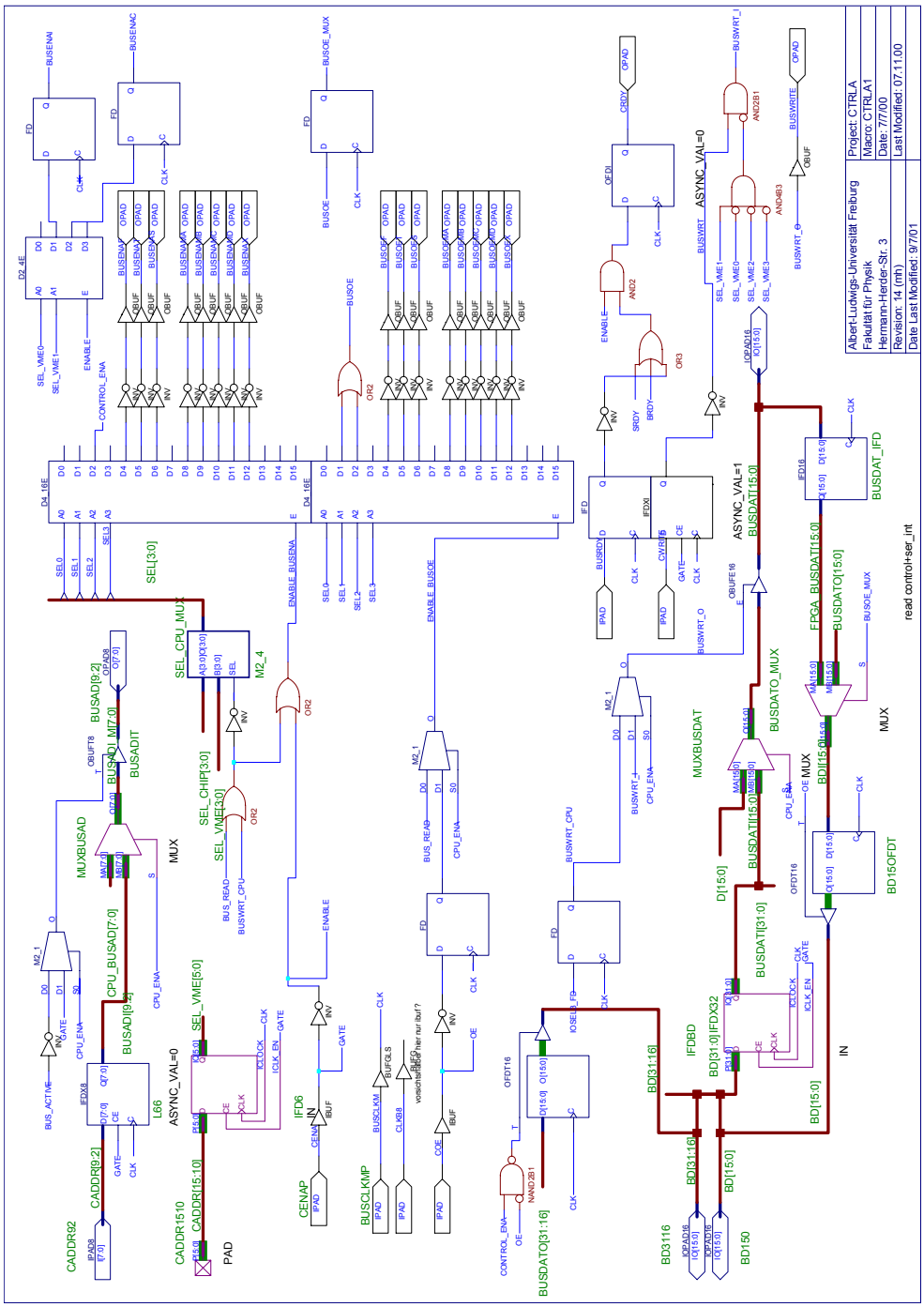



**Abbildung C.2:** Schematics Editor: Das Businterface von XCONTROL

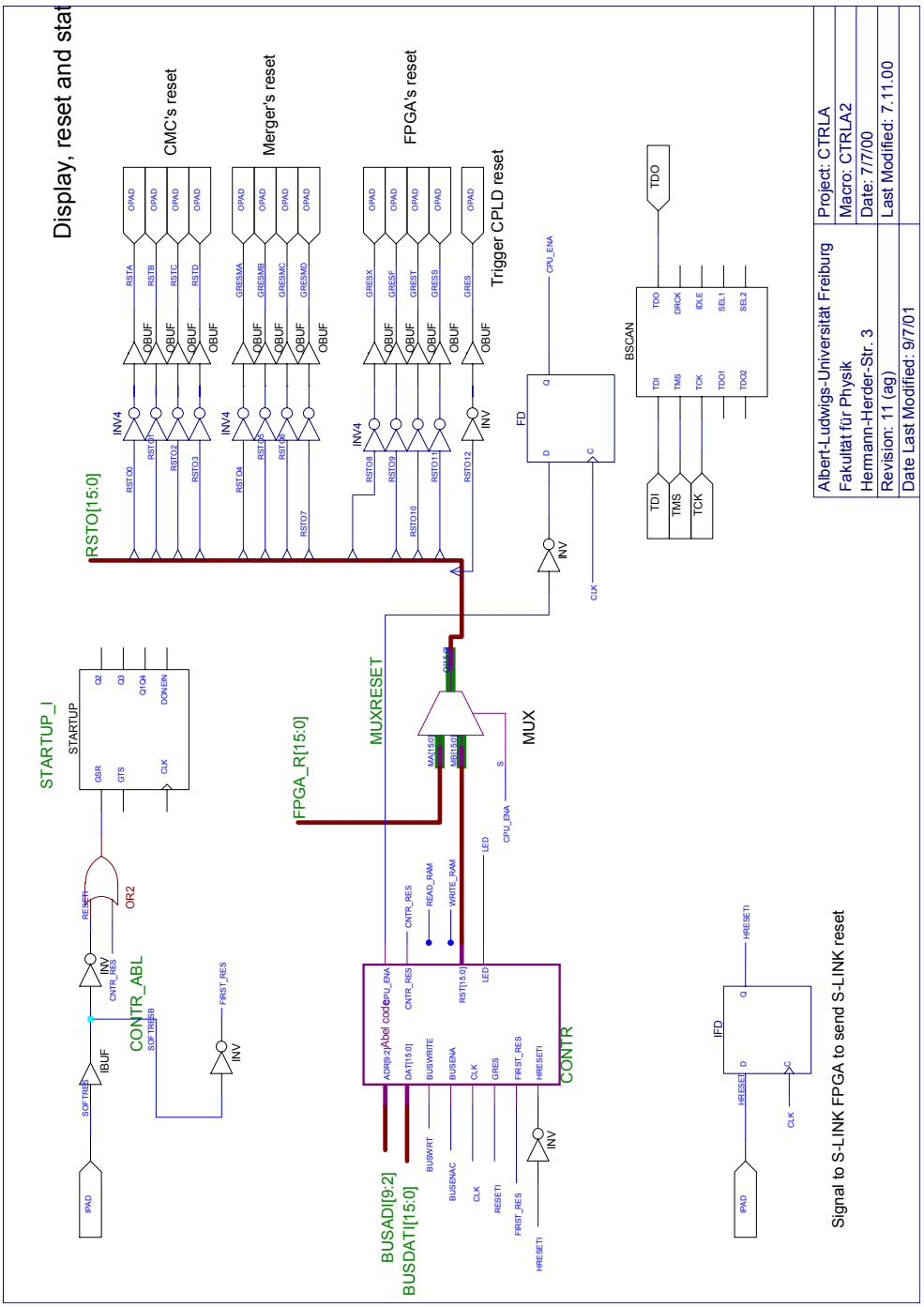



**Abbildung C.3:** Schematics Editor: Reset-Logik, Boundary Scan und Startup

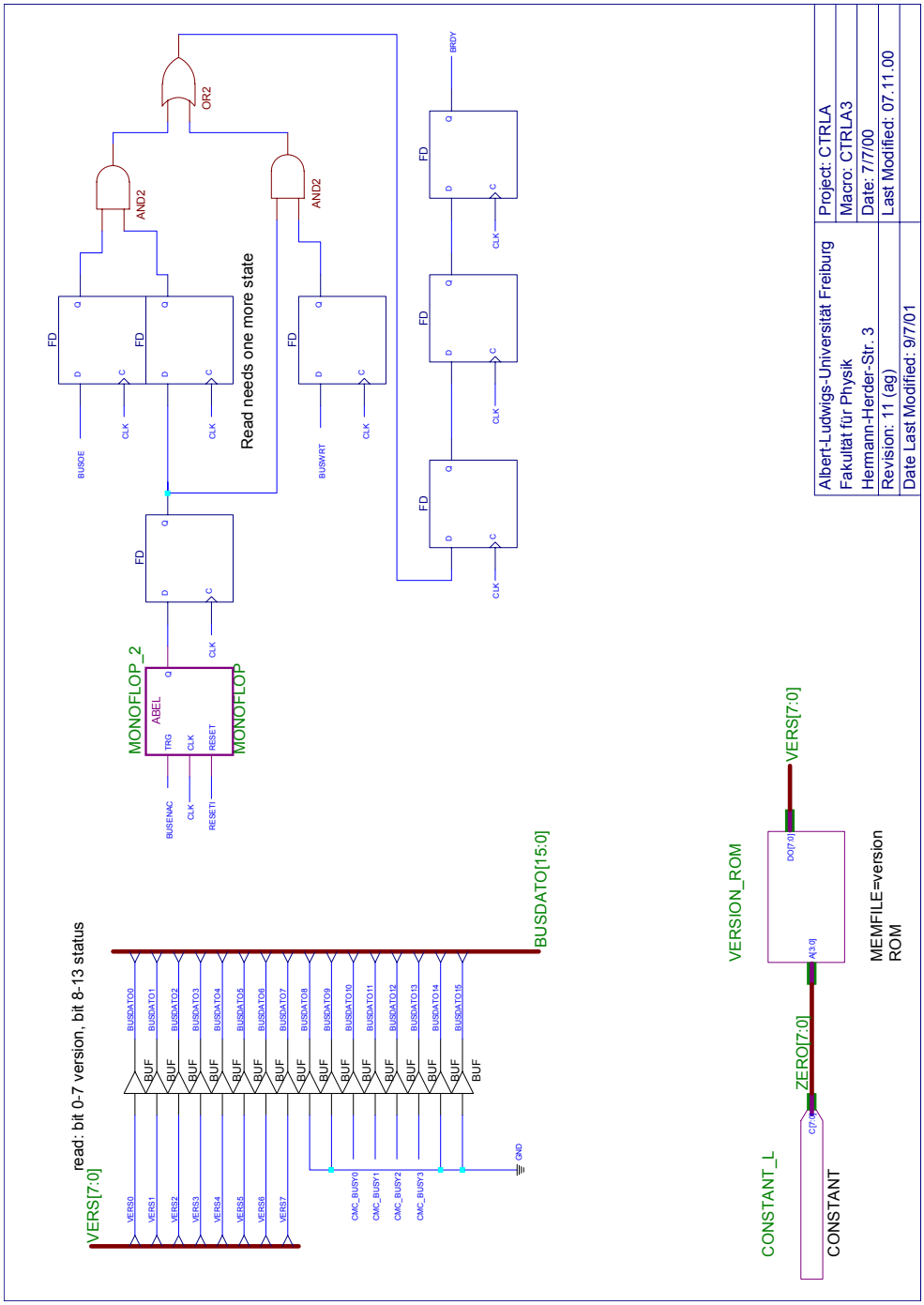



**Abbildung C.4:** Schematics Editor: ROM für die Versionsnummer, Flip-Flops für das Busready-Signal

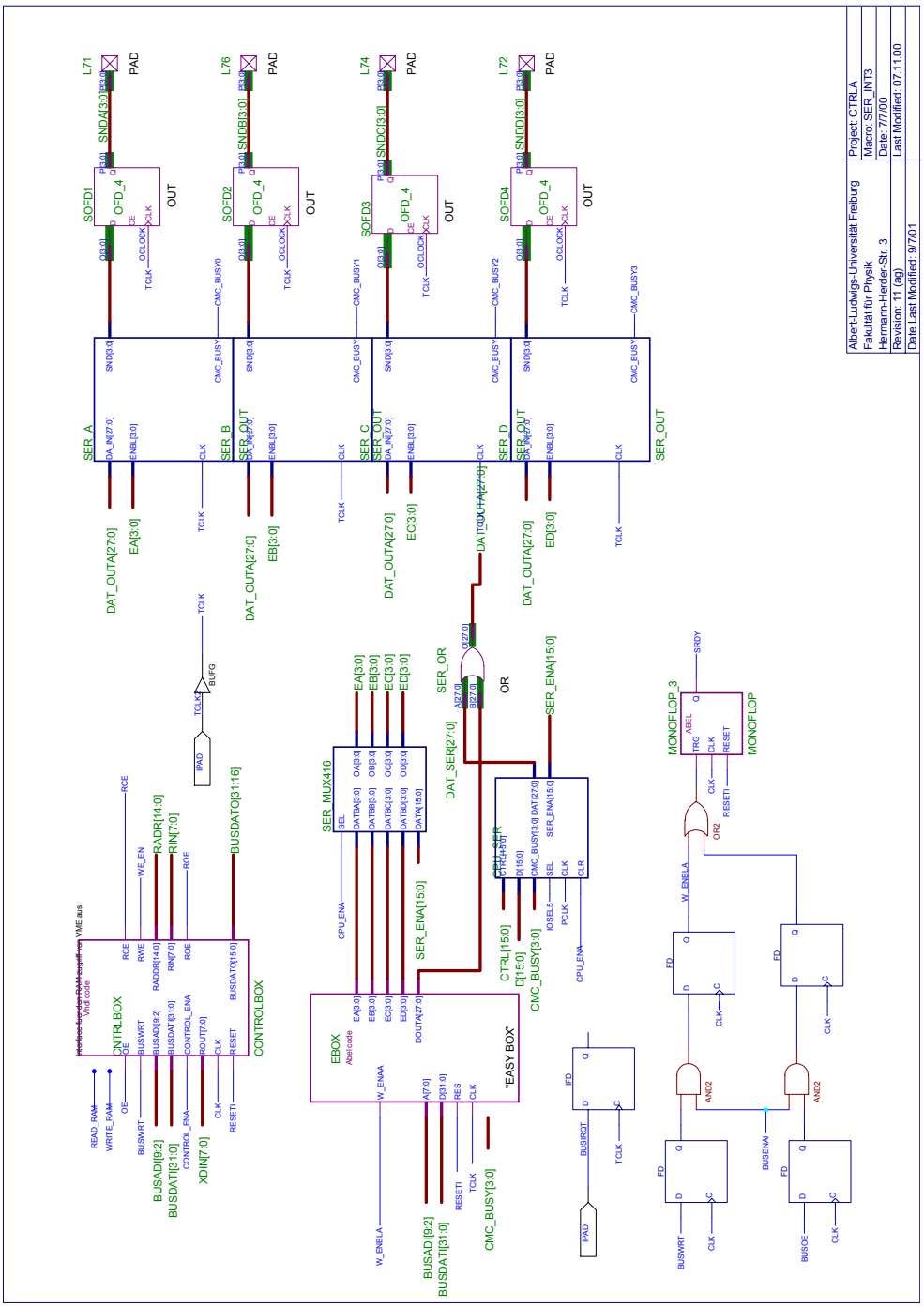



**Abbildung C.5:** Schematics Editor: Serielles Interface (Core-Generator-Blöcke), Logik für das serielle Interface von VME aus (Abel-Code), von der CPU aus (Register und Multiplexer) sowie VHDL-Code zur Ansteuerung des RAM auf dem CATCH (Lese/Schreibzugriffe)

# Anhang D

# Glossar

**9U** Größenangabe für VME-Module, 367 x 400 mm.

**ABEL** *Advanced Boolean Expression Language*, Programmiersprache zur Beschreibung von Hardware.

**ADC** *Analog-to-Digital-Converter*, wandelt Pulsformen in digitale Signale um.

**ASIC** *Application-Specific Integrated Circuit*, ein statischer integrierter Schaltkreis.

**Bestückungsseite** Die Seite der Platine, auf der sich die (meisten) Bauteile befinden ("Oberseite").

**Bit-File** Das File zum Programmieren eines FPGAs.

**BMS** *Beam Momentum Station*, Detektor der durch Ortsmessungen an dafür vorgesehenen Magneten die Impulse der Strahlteilchen vor dem Target bestimmt.

**BOB** *Begin Of Burst*, Signal, das den Beginn eines Spills ankündigt.

**Burst** Synonym für Spill.

**CATCH** *COMPASS Accumulate, Transfer & Control Hardware*, Herzstück der COMPASS-Auslese-Elektronik

**CERN** *Conseil Européen pour la Recherche Nucléaire*, Europäisches Forschungs-Zentrum für Teilchenphysik in Genf.

**CLB** *Complex Logic Block*, Baustein eines FPGAs.

**CLK** Abkürzung für Clock .

**Clock** Periodisches, rechteckiges Signal für sequenzielle Logik, erzeugt von einem Oszillator.

**CMC** *Common Mezzanine Card*, Aufsteckkarte für das CATCH-Modul.

**COMPASS** *COmmon Muon and Proton Apparatus for Structure and Spectroscopy*, Experiment am CERN, für das CATCH und `XCONTROL` enwickelt wurden.

**CPLD** *Complex Programmable Logic Device*, Programmierbarer Logikbaustein auf Basis von UND- und ODER-Gattern.

**CPU** *Central Processing Unit*, Prozessor, oft auch MPU (Microprocessor Unit) genannt.

**DATE** *Data Acquisition Test Environment*, Datennahme-Programm des ALICE-Experiments.

**EDIF** Format für die Netzliste.

**EOB** *End Of Burst*, Signal, das das Ende eines Spills ankündigt.

**$\mathcal{F}$1-TDC** *$\mathcal{F}$1-Time-To-Digital-Converter*, wandelt Zeitdifferenzen in digitale Daten um.

**$\mathcal{F}$1-TDC-CMC** Aufsteckkarte für das CATCH mit vier $\mathcal{F}$1-TDCs.

**FIFO** *First In—First Out*-Speicher.

**Flip-Flop** Logische Speicherzelle (1 bit).

**FPGA** *Field Programmable Gate Array*, programmierbarer Logik-Baustein.

**Front-End-Karte** Elektronische Schnittstelle zu den Detektoren. Dort werden die Daten digitalisiert.

**Gigabit-Ethernet** Standard für lokale Netzwerke (LANs). Seit Juni 1998 als IEEE 802.3z.

**Glitch** kurze Spannungsspitze auf einer Daten- oder Clockleitung, die einen undefinierten logischen Zustand verursachen kann.

**Header** Leitet eine Datensequenz ein.

**HDL** *Hardware Description Language*, Oberbegriff für Programmiersprachen zur Beschreibung von Hardware.

**HOTFibre-CMC** Aufsteckkarte für das CATCH zur Datenübertragung via Glasfaser mit HOTLink-Protokoll.

**HOTLink** Protokoll zur Datenübertragung via Twisted-Pair-Kabel oder Glasfaser.

**HOTLink-CMC** Aufsteckkarte für das CATCH zur Datenübertragung via Twisted-Pair-Kabel mit HOTLink-Protokoll.

**ID** Englische Kurzbezeichnung für Identifikationsnummer

**IO** Oberbegriff für Ein- und Ausgänge.

**IOB** *Input-/Output-Buffer* Schnittstelle des FPGAs zwischen interner Logik und Pins.

**JTAG** *Joint Test Action Group*, Standard-Schnittstelle für Boundary-Scan.

**Lötseite** Die Seite der Platine, auf der sich die Lötstellen diskreter Bauteile befinden (sozusagen die Unterseite der Platine).

**LUT** *Look-Up Table*, kombinatorisches Element im FPGA.

**LVDS** *Low Voltage Differential Signal*, Differenzieller Signalstandard.

**LVTTL** *Low Voltage Transistor Transistor Logic*, Signalstandard.

**LVPECL** Differenzieller Signalstandard.

**M2-Beamline** Strahlführung zwischen SPS und COMPASS-Experiment.

**MSB** *Most Significant Bit*, Bit mit höchstem Stellenwert.

**PCI** *Peripheral Component Interconnect*, Schnittstelle zum Verbinden der Peripheriegeräte eines PCs.

**PROM** *Programmable Read-Only Memory*, Speicher zum Programmieren des FPGA. Behält seinen Speicherinhalt auch ohne Spannung.

**RAM** *Random Access Memory*, Arbeitsspeicher.

**ROB** *Read-Out Buffer*, Zwischenspeicher der COMPASS-Auslese-Elektronik.

**Routing** Das Verbinden von Logikelementen durch Schaltung von Leitungen im FPGA.

**Scaler** Englisches Synonym für Zähler.

**Scaler-CMC** Zähler-Aufsteckkarte.

**SciFi** Abkürzung für Detektoren aus szintillierenden Fasern.

**S-LINK** Am CERN entwickeltes Protokoll zur optischen Datenübertragung.

**SMC** *Spin Muon Collaboration*, Experiment am CERN.

**SpartanXL** Für das CATCH verwendeter FPGA der Firma Xilinx.

**Spill** Zeitraum in dem der Teilchenstrahl auf das COMPASS-Target trifft.

**SPS** *Super Proton Synchrotron accelerator*, Beschleuniger am CERN.

**Spy-Buffer** Daten-FIFO auf dem CATCH, das über die VME-Schnittstelle ausgelesen werden kann.

**SRAM** *Static RAM*. Im Gegensatz zum dynamischen RAM (DRAM) behalten die Speicherzellen des SRAM ihren Wert bei, so lange das Bauteil mit Strom versorgt wird; ein regelmässiger Refresh ist nicht nötig. Dafür sind i.A. sechs Transistoren für ein Bit notwendig (DRAM benötigt nur einen Transistor und einen Kondensator).

**Target** Anordnung auf die der Teilchenstrahl trifft.

**TCS** *Trigger-Control-System*, Trigger-Kontroll-System.

**TDC** siehe $\mathcal{F}$1-TDC.

**Trailer** Beendet eine Datensequenz.

**Tristate-Buffer** Puffer mit einem dritten Zustand (hochohmig, „floating“).

**Twisted-Pair-Kabel** Kabel, bei dem jeweils zwei Leitungen verdrillt sind.

**UND-Gatter** fundamentales, logisches Gatter, verknüpft seine Eingangssignale durch die Operation „UND“.

**VGA** *Video Graphics Array* Grafikstandard für PCs, eingeführt von IBM im April 1987.

**VHDL** *Very High Speed Integrated Circuit (VHSIC) Hardware Description Language*, eine moderne Hardwarebeschreibungssprache.

**VME-Bus** *Versabus Modified for Eurocard* - ein von Motorola abstammendes Busprotokoll, das bei Industrierechnern, etwa für Telekommunikation, aber auch häufig in der Hochenergiephysik, eingesetzt wird.

**VME-Crate** VME-Überrahmen. In diesen werden die Module und der VME-Rechner eingeschoben.

**VME-Rechner** Computer, der in ein Crate eingebaut ist und über einen VME-Bus-verfügt.

**VME-Modul** Platine für einen VME-Crate.

# Abbildungsverzeichnis


2.1 Streuung longitudinal polarisierter Myonen an longitudinal polarisierten Protonen . . 5
2.2 Die Funktion $g_1^p$ aufgetragen gegen $x$. Die Daten wurden bei $Q_0^2 = 2\,\mathrm{GeV}^2$ (links) und bei $Q_0^2 = 10\,\mathrm{GeV}^2$ (rechts) entwickelt (Quelle: [6]). . . . . 6
2.3 Feynmangraph der Photon-Gluon-Fusion . . . . . 7
2.4 Mögliche Parametrisierungen der polarisierten Gluonverteilungsfunktion $\Delta G$. Links sind $\eta G(\eta)$ (oben) und drei Parametrisierungen von $\eta\Delta G(\eta)$ dargestellt. Rechts sind drei Parametrisierungen von $\Delta G(\eta)/G(\eta)$ abgebildet. Die Notation A, B und C bezieht sich auf Modelle von Gehrmann und Stirling [8]. . . . . 7
2.5 Zerfall des $D^0$-Mesons . . . . . 9
2.6 Aufbau des COMPASS-Detektors in der endgültigen Ausbaustufe (oben) und zu Anfang des Experiments . . . . . 10

3.1 Schematischer Aufbau der Datenerfassung bei COMPASS (aus [20]) . . . . . 14
3.2 Der COMPASS-Detektor . . . . . 14
3.3 Skizze einer Front-End-Karte für Straws. Vier der acht $\mathcal{F}1$- und ASD8b-Chips befinden sich auf der Lötseite der Platine und sind hier nicht sichtbar. Die Digital-Analogwandler und der Opltokoppler befinden sich ebenfalls auf der Lötseite und sind hell dargestellt . . . . . 16
3.4 Datenübertragung vom Detektor via HOTLink . . . . . 16
3.5 Datenübertragung vom Detektor an die TDC-CMCs . . . . . 19
3.6 Struktur eines SPS-Spills . . . . . 19
3.7 Aufbau des Trigger-Systems (aus [20]) . . . . . 21

4.1 Schematischer Aufbau eines CATCH Moduls . . . . . 24
4.2 Vereinfachtes Schaltbild eines CATCH-Moduls . . . . . 25
4.3 Schnittstellen des Control FPGAs . . . . . 30
4.4 Bisherige Steuerung der CATCH-Funktionen . . . . . 31
4.5 XCONTROL soll CATCH autonom steuern können . . . . . 32

5.1 Die fünf klassischen Komponenten eines Computers: Input, Output, Memory, Data-Path und Control . . . . . 34

5.2 Der Kern des XSOC-Projekts: der `xr16`-Prozessor. Die Control-Unit (links) erhält Instruktionen, die vom Memory Controller aus dem RAM geholt wurden über `INSN[15:0]`. Sie werden dort dekodiert und in Steuersignale umgesetzt (Mitte). Der Data-Path (rechts) enthält Register und ALU. Daten können mit `D[15:0]` in die Register geladen oder von dort auf den Bus geschrieben werden. Über den Bus `AN[15:0]` teilt der Data-Path dem Memory-Controller Adressen mit, von denen Daten gelesen bzw. an die Daten geschrieben werden sollen. . . . . 40
5.3 Die ALU des `xr16`-Prozessors . . . . . 41
5.4 Register und ALU des `xr16` . . . . . 43

6.1 Das im CATCH integrierte Display (Foto: Infineon) . . . . . 48
6.2 Serielles Datenformat zur Initialisierung des $\mathcal{F}$1-TDC-Chip . . . . . 52
6.3 Aufteilung des Adressraums für XCONTROL . . . . . 54

7.1 Die CLBs eines XCS40XL sind in einer Matrix angeordnet, die von IO-Blocks umgeben ist, welche zur Ein- und Ausgabe von Signalen auf den Pads des FPGAs dienen (Aus [35]) . . . . . 56
7.2 Ein CLB des XCS40XL besteht aus zwei Look-Up-Tables mit je vier Eingängen (G-LUT, F-LUT), einem mit drei Eingängen (H-LUT), Multiplexern sowie zwei Flip-Flops (Aus [35]) . . . . . 56
7.3 Das ABEL-Makro `CONTR` übernimmt die Dekodierung der VME-Befehle für das Reset der FPGAs und die Umschaltung zwischen CPU- und Legacy-Modus . . . . . 59
7.4 Das VHDL-Makro `CONTROLBOX` implementiert die Schnittstelle, mit der man über den VME-Bus auf das RAM zugreifen kann. . . . . 60
7.5 Busprotokoll des internen CATCH-Bus zur Kommunikation zwischen Control-FPGA und den anderen FPGAs . . . . . 61
7.6 Implementierung des Businterfaces von der CPU aus . . . . . 61
7.7 Schnittstelle vom Prozessor zum Display . . . . . 62
7.8 Implementierung des seriellen Interfaces. Die Parallel-nach-seriell-Wandler sind hier nicht abgebildet . . . . . 63
7.9 Unser treues Werkzeug: Der Xilinx Projektmanager . . . . . 64
7.10 Der Schematics-Editor mit geöffnerem `XCONTROL`-Projekt . . . . . 65
7.11 Placement-Schritt der XilinX Flow-Engine . . . . . 66
7.12 Zehn Minuten später: Geschafft, alle Timing-Constraints erfüllt . . . . . 67
7.13 Das fertige Design im Floorplanner, unten erkennt man die regelmässige Struktur des `xr16`-Kerns, links oben die serielle Schnittstelle . . . . . 68
7.14 Timing-Simulation des `XCONTROL`-Projekts . . . . . 69

A.1 Aufteilung des Adressraums für die Konfigurationsdaten . . . . . 76

B.1 Vier-Zeichen Matrix Display auf dem CATCH . . . . . 83
B.2 Serielles Datenformat für das Display . . . . . 84
B.3 Timing Diagramm für das Display . . . . . 84

C.1 Schematics Editor: Prozessor, Memory Controller und on-chip Peripheriebausteine . 92

C.2 Schematics Editor: Das Businterface von XCONTROL . . . . . 93
C.3 Schematics Editor: Reset-Logik, Boundary Scan und Startup . . . . . 94
C.4 Schematics Editor: ROM für die Versionsnummer, Flip-Flops für das Busready-Signal 95
C.5 Schematics Editor: Serielles Interface (Core-Generator-Blöcke), Logik für das serielle Interface von VME aus (Abel-Code), von der CPU aus (Register und Multiplexer) sowie VHDL-Code zur Ansteuerung des RAM auf dem CATCH (Lese/Schreibzugriffe) 96

# Tabellenverzeichnis


2.1 Kinematische Variablen bei der tiefinelastischen Streuung . . . . . 4

3.1 Aufteilung der benötigten CATCH-Module und CMC-Karten auf die Komponenten des Detektors . . . . . 18

5.1 Die im `xr16`-Prozessor verfügbaren Register . . . . . 35
5.2 Der xr16 Mikroprozessorkern benötigt nur 43 Befehle um eine Integer Teilmenge von C zu implementieren . . . . . 39
5.3 `xr16` besitzt eine einfache Pipeline mit drei Stufen . . . . . 42
5.4 `xr16` kann Load-Befehle gepipelined ausführen . . . . . 44

6.1 Register des Formatter-FPGAs und ihre Funktion. . . . . 51
6.2 Resets an die FPGAs und CMCs werden über die Speicheradresse `0xC000` gegeben 51

B.1 Die Information eines Zeichens auf dem Display besteht aus einem Byte Character-Address und fünf Byte Daten für die fünf Zeilen . . . . . 84
B.2 Nummerierung der Character-Address . . . . . 85

# Literaturverzeichnis


[1] COMPASS Collaboration, *Common Muon and Proton Apparatus for Structure and Spectroscopy*, CERN/SPLC 96-14, 1996.

[2] The Crystal Barrel Collaboration, C. Amsler *et al.*, *Phys. Lett.* B **342**, 433 (1995)

[3] The European Muon Collaboration, J. Ashman et al., *Phys. Lett* B **206**, 364 (1988)

[4] Gerhard K. Mallot, *The Spin Structure Of The Nucleon from the SMC Experiment*, Habilitationsschrift, Mainz 1996

[5] F. Halzen, A.D. Martin, *Quarks and Leptons*, John Wiley & Sons, 1984

[6] The HERMES Collaboration, Measurement of the proton spin structure function $g_1^p$ with a pure hydrogen target, Physics Letters B **442** (1998) 484-492

[7] The HERMES Collaboration, Flavor decomposition of the polarized quark distributions in the nucleon from inclusive and semi-inclusive deep-inelastic scattering, Physics Letters B **464** (1999) 123-134

[8] T. Gehrmann and W.J. Stirling, *Z. Phys.* C 65, 461 (1994).

[9] D.Adams *et al.* [Spin Muon Collaboration], *The polarized double cell target of the SMC*, Nucl. Instrum. Meth. A **437** 23 (1999).

[10] The Muon Trigger Group, *Muon Trigger Documentation*, January 20, 2000

[11] G. Braun et al., *An Eight Channel Time-to-Digital Converter Chip for High Rate Experiments*, Contributed to 5th Workshop on Electronics for the LHC Experiments, Snowmass, Colorado; CERN/LHCC/99-33 und hep-ex/9911009.

[12] Hewlett-Packard, HCPL-2400, HCPL-2430: 20 MBd High CMR Logic Gate Optocouplers

[13] CYPRESS Semiconductor Corporation, *CY7B923/33 HOTLink Transmitter/Receiver Data Sheet.*

[14] Matthias Hoffmann, *Aufbau einer Signalquelle zum Test von schnellen Datenaufnahmesystemen*, Staatsexamensarbeit, Freiburg Dezember 2000

[15] IEEE P1386/Draft 2.0 04-APR-1995 *Draft Standard for a Common Mezzanine Card Family: CMC*

[16] M. Niebuhr, *Entwickling eines totzeitfrei auslesbaren 250 MHz Zählers*, Diplomarbeit, Freiburg, 2000.

[17] F.H. Heinsius, K. Königsmann, M. Schierloh, T. Schmidt, H. Schmitt, J.Urban, *Draft Standard for CATCH Mezzanine Cards: CATCH/OPEN, G. Braun, H. Fischer, J. Franz, A. Grünemaier*, Compass Note 2001-7

[18] O. Boyle et al., *The S-LINK Interface Specification*, ECP Division, CERN

[19] The CERN ALICE DAQ group, *ALICE DATE User's Guide*, November 1998

[20] Igor Konorov, Lars Schmitt, Boris Grube, *COMPASS TCS Documentation*, COMPASS Note 2001-9

[21] H. Fischer, J. Franz, A. Grünemaier, F.H. Heinsius, M. Hoffmann, F. Karstens, W. Kastaun, K. Königsmann, M. Niebuhr, M. Parr, R. Risken, T. Schmidt, H. Schmitt, A. Schweimler, J. Urban, M. von Hodenberg, I. Konorov, L. Schmitt, *The COMPASS Online Data Format, Version 2*, COMPASS-Note 2001-8.

[22] M. Schierloh, *Einsatz programmierbarer Logikbausteine in der COMPASS-Ausleseelektronik*, Diplomarbeit, Freiburg, Dezember 1998.

[23] L. Hennig, *Integration einer Kontroll und Ausleseeinheit in das COMPASS-Experiment*, Diplomarbeit, Freiburg, August 2000

[24] Jan Gray, *Building a RISC System in an FPGA*, Circuit Cellar Magazine, issues 116, 117 and 118

[25] `www.opencores.org`

[26] The Free Model Foundry, `http://www.vhdl.org/vi/fmf/`

[27] The F-CPU-Project, `http://f-cpu.tux.org/`

[28] David A. Patterson and John L. Hennessy, *Computer Organization & Design*, Second Edition, Morgan Kaufmann, 1998

[29] Jan Gray, *The xr16 specification*, distributed with the XSOC project files

[30] Matthias Hoffmann, *XCONTROL users's manual*, Freiburg, September 2001

[31] H.Fischer et al. *CATCH User Manual, (preliminary)* , private Mitteilung

[32] Infineon `SCDV554x` Datenblatt, erhältlich unter `http://www.infineon.com/cmc_upload/0/000/012`

[33] ITRS *International Technological Roadmap for Semiconductors, 1999 Edition*, 1999 `http://public.itrs.net/`

[34] Xilinx, Inc., San Jose, USA (`http://www.xilinx.com`).

[35] Xilinx, Inc., *The programmable Logic Data Book 2000*, San Jose 2000

[36] Xilinx, Inc. *Development System Reference Guide 3.1i*, San Jose, 2001
`http://support.xilinx.com/support/sw_manuals/3_1i/download/dev_ref.zip`

[37] Xilinx, Inc. *Foundation Series 2.1i User Guide*, San Jose, 1999
`http://support.xilinx.com/support/sw_manuals/3_1i/download/fsuguide.zip`

[38] D. Pellerin, D. Taylor, *VHDL Made Easy!*, Prentice Hall, 1997

[39] W. J. Dally, J. W. Poulton, *Digital Systems Engineering*, Cambridge University Press, 1998

[40] Falk Karstens, *persönliche Mitteilung*

[41] Homepage des Lcc-Projekts: `http://www.cs.princeton.edu/software/lcc`

[42] Jan Gray, *Getting started with the XSOC project*, distributed with the XSOC project files

Ich erkläre, daß ich die vorliegende Arbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe, und daß alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angabe der Quellen als Entlehnungen kenntlich gemacht worden sind.

Matthias Hoffmann, Oktober 2001