universitätbonn

Entwicklung eines ungetakteten 64-Kanal-Meantimers und einer Koinzidenzschaltung auf einem FPGA

Diplomarbeit in Physik
angefertigt am
Physikalischen Institut

vorgelegt der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

von

John Bieling

Ich versichere, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate kenntlich gemacht habe.

Bonn, den 15.November 2010

Referent:	PD Dr. Jörg Pretz
Korreferent:	Prof. Dr. Kai-Thomas Brinkmann

Inhaltsverzeichnis

1. Einleitung

Ende des 19. Jahrhunderts war sich die wissenschaftliche Gemeinde sicher, bald alle wichtigen Prinzipien der Natur entschlüsselt zu haben. Doch gerade im Verlauf des 20. Jahrhunderts veränderte sich das Verständnis der physikalischen Zusammenhänge durch die Quantentheorie grundlegend und die Kenntnisse über den Aufbau der Materie wurden vor allem mit Hilfe von Experimenten an Teilchenbeschleunigern enorm erweitert.

Heute bezeichnet man das Elektron zusammen mit dem Proton und dem Neutron zwar weiterhin als Grundbausteine der Materie, doch seit den Experimenten von J.I. Friedman, H.W. Kendall und R.E. Taylor am SLAC[1] Ende der 1960er Jahre [34] gibt es eindeutige Hinweise darauf, dass Proton und Neutron punktförmige Substrukturen aufweisen. Diese werden als Partonen bezeichnet; die geladenen werden mit den Quarks und die elektrisch neutralen mit den Gluonen identifiziert.

Der Spin des Protons ist $\frac{1}{2}\hbar$[†]. Die anfängliche Annahme, dass sich dieser zu 100% aus den Spinanteilen der Quarks ($\Delta\Sigma = 1$, s. Gleichung *1.1*) zusammensetzt, konnte 1988 bei den EMC[3]-Experimenten jedoch nicht bestätigt werden. Entgegen aller Erwartungen wurden damals sehr viel kleinere, sogar mit Null verträgliche Anteile gemessen ($\Delta\Sigma = 0.12 \pm 0.17$ [3]). Diese Entdeckung war so unerwartet, dass eine Zeit lang sogar von einer Spin-Krise gesprochen wurde. Inzwischen schreibt man neben den Quarks auch den Gluonen (ΔG) und deren Bahndrehimpulsen (L_q und L_g) einen Anteil am Gesamtspin zu:

$$\frac{1}{2} = \frac{1}{2}\Delta\Sigma + L_q + \Delta G + L_g \,. \tag{1.1}$$

Seither untersucht man - wie derzeit auch mit dem COMPASS[4]-Experiment am CERN[5] - die Spinstruktur des Nukleons, um die verschiedenen Anteile zu bestimmen.

[1] Stanford Linear Accelerator Center
[†] Im weiteren Verlauf dieser Arbeit wird $\hbar = c = 1$ gesetzt.
[3] European Muon Collaboration
[4] COmmon Muon Proton Apparatus for Structure and Spectroscopy
[5] Conseil Européen pour la Recherche Nucléaire (Europäisches Kernforschungszentrum)

Für die Untersuchung dieser Strukturen innerhalb des Protons ($< 10^{15}$m) bietet sich die tiefinelastische Streuung von hochenergetischen Myonen an einem festen Target an, da das Auflösungsvermögen eines Streuexperiments maßgeblich durch die De-Broglie-Wellenlänge $\lambda = \frac{h}{p}$ des verwendeten Teilchenstrahls bestimmt wird[6]. Allerdings treten bei solchen Experimenten für gewöhnlich hohe Detektionsraten mit einem sehr großen Untergrundanteil auf. Selbst moderne Datenerfassungssysteme stoßen da an ihre Grenzen, sowohl bzgl. der Datenmenge als auch der Schreibrate. Daher versucht man den Untergrund *vorher* zu erkennen und gar nicht erst zu speichern. Eine einfache Möglichkeit besteht in einer groben Spuranalyse: Nur wenn das Myon wirklich am Target gestreut wurde, werden die damit verbundenen Detektorereignisse für die spätere genauere Analyse gespeichert. Diese Echtzeitprüfung (innerhalb von 500ns) wird durch das COMPASS-Trigger-System durchgeführt.

Im Rahmen dieser Diplomarbeit wird eine Komponente für dieses Trigger-System entwickelt. Diese Komponente wertet die Signale von zwei neuen Hodoskop[7]-Detektoren des COMPASS-Experiments aus, um die Pfade von durch das Experiment hindurchfliegenden Myonen zu analysieren. Sie bietet dazu für jedes der beiden Hodoskope 32 unabhängige sog. Meantimerschaltungen zur Bestimmung der Myon-Durchflugzeitpunkte. Weiterhin enthält sie eine komplexe Koinzidenzschaltung, mit der die Orts- und Zeitinformationen beider Hodoskope in Relation gesetzt und dadurch die Pfade der Myonen bestimmt werden können. Zusammen mit den beiden neuen Hodoskopen bilden diese Schaltungen den LAS[8]-Trigger, mit dem auch Streuereignisse unter großen Winkeln registriert werden können. Diese kinematische Region wurde bis jetzt noch nicht durch einen Hodoskop-Trigger des Trigger-Systems abgedeckt.

Als Projektvorgabe wird gefordert, dass die benötigten elektrischen Schaltungen nicht auf einer Platine entwickelt, sondern auf einem FPGA[9] programmiert werden. Mit dem zur Verfügung stehenden FPGA (Virtex 5 von Xilinx) kann mit einer getakteten Schaltung maximal eine zeitliche Auflösung von ca. 1ns erreicht werden, dies genügt jedoch nicht den Anforderungen des Experiments. Um eine höhere Auflösung zu erreichen, werden in dieser Diplomarbeit alle 64 Meantimer und die Koinzidenzschaltung als ungetaktete Schaltungen entwickelt. Es gibt dazu bislang keine dokumentierten Versuche, sodass hier neue Konzepte und Techniken entwickelt werden müssen, um für alle 64 Meantimer ein möglichst identisches Timing-Verhalten zu erreichen.

In den nächsten beiden Kapiteln werden einige physikalische und technische Grundlagen erläutert. Anschließend erfolgt im 4. Kapitel eine Beschreibung des COMPASS-Experiments und seiner Komponenten. Im 5. Kapitel wird eine Einführung in die FPGA-Technologie gegeben und der in dieser Arbeit eingesetzte Virtex 5 vorgestellt. Anschließend werden im 6. Kapitel ausführlich die Entwicklung der 64 parallelen Meantimer und die Konstruktion der Koinzidenzschaltung erläutert sowie einige Testmessungen diskutiert. Im darauf folgenden Kapitel wird die Installation des fertigen Systems am COMPASS-Experiment beschrieben. Die Diplomarbeit schließt im 8. Kapitel mit einer Zusammenfassung und einem Ausblick ab.

[6] Dies gilt z.B. analog für das Auflösungsvermögen eines Mikroskops, das durch die Wellenlänge des verwendeten Lichts begrenzt ist.

[7] gr. hodos: Ort, Detektor zur Ortsbestimmung von ionisierenden Teilchen.

[8] Large Angle Spectrometer

[9] Field Programmable Gate Array, regelmäßige Anordnung von programmierbaren Logikbausteinen.

2. Physikalische Grundlagen

2.1 Die Tiefinelastische Streuung

Die tiefinelastische Streuung (DIS[1]) beschreibt die Streuung eines Leptons an einem Nukleon bei sehr hohen Strahlenergien. Das einlaufende Lepton wechselwirkt dabei über den Austausch eines virtuellen Photons γ^* mit einem Quark des Nukleons und wird unter einem Winkel θ gestreut. Ist der Energieübertrag groß genug, bricht das Nukleon auf und das Quark, mit dem das Lepton wechselwirkte, wird herausgelöst (s. Abbildung *2.1*). Aufgrund der Confinement-Theorie[2] fragmentiert es in einen hadronischen Endzustand X.

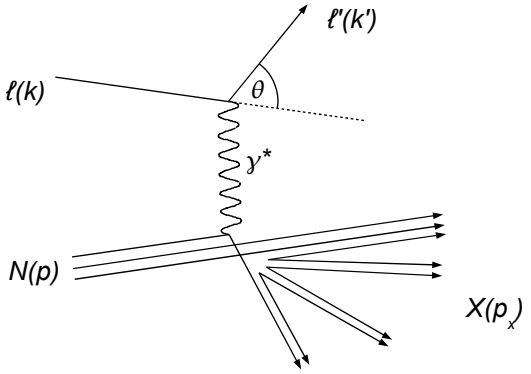


Abbildung 2.1: Tiefinelastische Streuung eines Leptons ℓ an einem Nukleon N mit hadronischen Endzustand X.

[1] Deep Inelastic Scattering

[2] Die Confinement-Theorie beschreibt die Tatsache, dass bis jetzt einzelne Quarks nicht beobachtet werden konnten. Die Quantenchromodynamik schreibt jedem Quark eine Farbe (r, g, b) und jedem Antiquark eine Antifarbe $(\bar{r}, \bar{g}, \bar{b})$ zu. Diese Farben lassen sich durch Kombination neutralisieren, entweder als ein Farb-Antifarb-Paar oder als Mischung aller drei Farben bzw. Antifarben. Quarks sollen nun nicht einzeln, sondern nur in einem farbneutralen Verbund existieren können. Sie treten daher mindestens Paarweise auf. Alternativ lässt sich dieser Sachverhalt auch über das Wechelwirkungspotential der starken Wechelwirkung beschreiben. Im Gegensatz zur Coulombkraft nimmt die Farbkraft mit dem Abstand zu. Wird ein einzelnes Quark aus einem farbneutralen Verbund herausgelöst, steigt die dafür benötigte Energie immer weiter an, bis sie groß genug ist, um ein neues $q\bar{q}$-Paar zu erzeugen und sich dadurch wieder zwei farbneutrale Objekte bilden.

Wird nur das gestreute Lepton nachgewiesen, nennt man die Messung inklusiv. Wird zusätzlich noch mindestens eines der Hadronen aus dem Endzustand nachgewiesen, bezeichnet man die Messung als semi-inklusiv. Bei einer exklusiven Messung werden alle Hadronen des Endzustands nachgewiesen.

Im Unterschied zur inelastischen Streuung, bei der sich z.B. bei den Nukleonresonanzen ebenfalls neue Hadronen im Endzustand beobachten lassen, entstehen diese bei der DIS direkt durch den Streuvorgang und nicht erst durch Abregung des angeregten Targetnukleons. Der Hadronisierungsprozess der DIS ist in Abbildung *2.1* dargestellt, dabei gelten folgende Bezeichnungen:

$$\boldsymbol{k} = \begin{pmatrix} E \\ \vec{k} \end{pmatrix}, \qquad \text{Viererimpuls des einlaufenden Leptons} \tag{2.1}$$

$$\boldsymbol{k'} = \begin{pmatrix} E' \\ \vec{k'} \end{pmatrix}, \qquad \text{Viererimpuls des auslaufenden Leptons} \tag{2.2}$$

$$\boldsymbol{p} = \begin{pmatrix} M \\ \vec{p} \end{pmatrix}, \qquad \text{Viererimpuls des Targetnukleons mit Masse } M \tag{2.3}$$

$$\boldsymbol{q} = \boldsymbol{k} - \boldsymbol{k'}. \qquad \text{Viererimpuls des Photons, Vierimpulsübertrag} \tag{2.4}$$

Es folgen die Definitionen von einigen lorentzinvarianten kinematischen Größen der tiefinelastischen Streuung:

$$\nu = \frac{\boldsymbol{p}\cdot\boldsymbol{q}}{M} \tag{2.5}$$

$$\overset{TRF}{=} \frac{1}{M}\begin{pmatrix} M \\ \vec{0} \end{pmatrix}\cdot\begin{pmatrix} E-E' \\ \vec{k}-\vec{k'} \end{pmatrix} = E - E', \tag{2.6}$$

$$y = \frac{\boldsymbol{p}\cdot\boldsymbol{q}}{\boldsymbol{p}\cdot\boldsymbol{k}} \tag{2.7}$$

$$\overset{TRF}{=} \frac{M\cdot\nu}{M\cdot E} = \frac{\nu}{E}, \tag{2.8}$$

$$q^2 = (\boldsymbol{k}-\boldsymbol{k'})^2 = \nu^2 - \left(\vec{k}-\vec{k'}\right)^2 \tag{2.9}$$

$$\overset{TRF}{=} 2m_l^2 - 2\left(EE' - |\vec{k}||\vec{k'}|\cos\theta\right) \tag{2.10}$$

$$\approx -4EE'\sin^2\frac{\theta}{2} = -4E(E-\nu)\sin^2\frac{\theta}{2} \tag{2.11}$$

$$\approx -4E(E-Ey)\sin^2\frac{\theta}{2} = -4E^2(1-y)\sin^2\frac{\theta}{2}. \tag{2.12}$$

Im sog. Target Rest Frame (TRF)[3] kann demnach ν als absoluter Energieverlust und y als relativer Energieverlust des gestreuten Leptons interpretiert werden. Nach Gleichung *2.12* gilt, dass $q^2 < 0$, daher definiert man

$$Q^2 = -q^2 \tag{2.13}$$

[3] Laborsystem eines ruhenden Targets.

um nicht mit negativen Größen arbeiten zu müssen [30]. Die für das Auflösungsvermögen relevante De-Broglie-Wellenlänge des virtuellen Photons lässt sich durch Q^2 ausdrücken, es gilt:

$$\lambda = \frac{1}{\sqrt{Q^2}}\,. \tag{2.14}$$

Eine weitere lorentzinvariante kinematische Größe der tiefinelastischen Streuung ist die invariante Masse des hadronischen Endzustands:

$$\begin{aligned} W^2 &= (\boldsymbol{p}+\boldsymbol{q})^2 && (2.15)\\ &= p^2 + 2\boldsymbol{p}\cdot\boldsymbol{q} + q^2 && (2.16)\\ &= M^2 + 2M\nu - Q^2\,. && \text{nach } \textit{2.5} \text{ und } \textit{2.13} \quad (2.17) \end{aligned}$$

Bei der elastischen Streuung gilt $W^2 = M^2$, für den Impulsübertrag bei der elastischen und bei der inelastischen Streuung gilt dann nach Gleichung *2.17*:

$$\begin{aligned} Q^2 &= 2M\nu\,, && \text{elastisch} \quad (2.18)\\ Q^2 &= M^2 + 2M\nu - W^2\,. && \text{inelastisch} \quad (2.19) \end{aligned}$$

Vergleicht man die beiden, sieht man sofort, dass es bei der inelastischen Streuung einen zusätzlichen freien Parameter gibt. Der Impulsübertrag Q^2 ist nicht mehr direkt proportional zum Energieübertrag ν, sondern hängt auch von der Anregungsenergie des Targets bzw. von der invarianten Masse des hadronischen Endzustands ab. Alternativ zu (Q^2, ν) können auch (θ, E') als die beiden freien Parameter gewählt werden (s. Gleichung *2.11*).

Diese Abhängigkeit zeigt sich auch im Wirkungsquerschnitt der tiefinelastischen Streuung, bzw. in den beiden Strukturfunktionen W_1 und W_2:

$$\frac{\mathrm{d}^2\sigma}{\mathrm{d}\Omega \mathrm{d}E'} = \left(\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega}\right)_{\text{Mott}} \left[W_2(Q^2,\nu) + 2W_1(Q^2,\nu)\tan^2\frac{\theta}{2}\right]\,. \tag{2.20}$$

2.2 Die Bjorken'sche Skalenvariable

Die Bjorken'sche Skalenvariable ist wie folgt definiert:

$$x_B = \frac{Q^2}{2\boldsymbol{p}\cdot\boldsymbol{q}} = \frac{Q^2}{2M\nu}\,. \tag{2.21}$$

Weiterhin werden die Strukturfunktionen der tiefinelastischen Streuung meistens durch dimensionslose Strukturfunktionen ersetzt:

$$\begin{aligned} F_1(x_B, Q^2) &= MW_1(Q^2,\nu)\,, && \text{magnetische Strukturfunktion} \quad (2.22)\\ F_2(x_B, Q^2) &= \nu W_2(Q^2,\nu)\,. && \text{elektrische Strukturfunktion} \quad (2.23) \end{aligned}$$

In den ersten Messungen zeigten diese Strukturfunktionen eine relative Q^2-Invarianz (ein sog. Skalenverhalten), wodurch - analog zum Formfaktor des Elektrons - auf punktförmige Streuzentren geschlossen werden konnte. [34]

Trägt man die Strukturfunktionen gegen die Bjorken'sche Skalenvariable auf, ist x_B ein Maß für die Elastizität der Streuung. Dies ergibt sich, wenn man den elastischen Grenzfall betrachtet; es gilt dann wieder $Q^2 = 2M\nu$ (also $x_B = 1$). Die inelastische Streuung wird dann durch $0 < x_B < 1$ charakterisiert. Dies ist sehr anschaulich in Abbildung *2.2* zu erkennen.

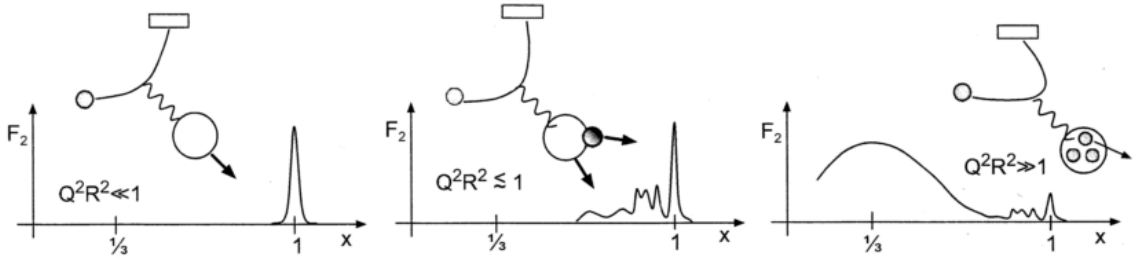


Abbildung 2.2: Strukturfunktionen bei unterschiedlichen Strahlenergien. Angegeben ist jeweils das Verhältnis zum Nukleonradius R. [30]

Bei festem Streuwinkel sind für drei verschiedene Energien die ermittelten Strukturfunktionen gegen x_B aufgetragen. Im ersten Bild ist aufgrund der geringen Energie nur der elastische Peak ausgeprägt. Bei der inelastischen Streuung im mittleren Bild liegt die Wellenlänge der Streuteilchen im Bereich des Nukleon-Radius und es sind bereits die Δ-Resonanzen erkennbar. Bei der tiefinelastische Streuung schließlich können die Bestandteile des Nukleons aufgelöst werden.

Nach dem Quark-Parton-Modell sind diese Bestandteile punktförmig und werden als Partonen bezeichnet. Die ungeladenen Spin $\frac{1}{2}$ Partonen werden mit den Quarks und die geladenen Spin 1 Partonen mit den Gluonen identifiziert. Die Partonen können sich quasi-frei bewegen und die tiefinelastische Streuung am Nukleon kann als elastische Streuung an einem einzelnen Parton aufgefasst werden. Den Impuls dieses Partons kann man dann als einen Bruchteil vom Gesamtimpuls des Nukleons ausdrücken:

$$\boldsymbol{p}_p = \xi \boldsymbol{p}\,. \qquad \text{Impulsanteil eines Partons am Gesamtimpuls} \quad (2.24)$$

Aus der invarianten Masse W^2 einer solchen Lepton-Parton-Streuung folgt:

$$p'^2_p = \left(\boldsymbol{p}_p + \boldsymbol{q}\right)^2\,, \quad (2.25)$$

$$m_p^2 = m_p^2 + 2 \cdot \boldsymbol{p}_p \cdot \boldsymbol{q} + q^2\,, \quad (2.26)$$

$$0 = 2 \cdot \xi \boldsymbol{p} \cdot \boldsymbol{q} - Q^2\,, \quad (2.27)$$

$$\xi = \frac{Q^2}{2\boldsymbol{p} \cdot \boldsymbol{q}} \equiv x_B\,. \quad (2.28)$$

Die physikalische Bedeutung der Bjorken'schen Skalenvariable ist demnach der Impulsanteil des gestreuten Partons am Gesamtimpuls des Nukleons. Diese Bedeutung hilft beim Verständnis der sog. Skalenbrechung: Bei höheren Strahlenergien ist die Strukturfunktion nicht mehr unabhängig von Q^2, sondern steigt zu kleinem x_B mit Q^2 an (s. Abbildung *2.3*).

Mit größerer Auflösung findet man also mehr Partonen mit geringem Impulsanteil. Diese Erweiterung des Quark-Parton-Modells wird inzwischen sehr gut durch die Quantenchromodynamik beschrieben und mit Gluon-Abstrahlungen und kurzlebigen $q\bar{q}$-Paaren (sog. Seequarks) erklärt, die erst bei sehr hohen Q^2 aufgelöst werden können.

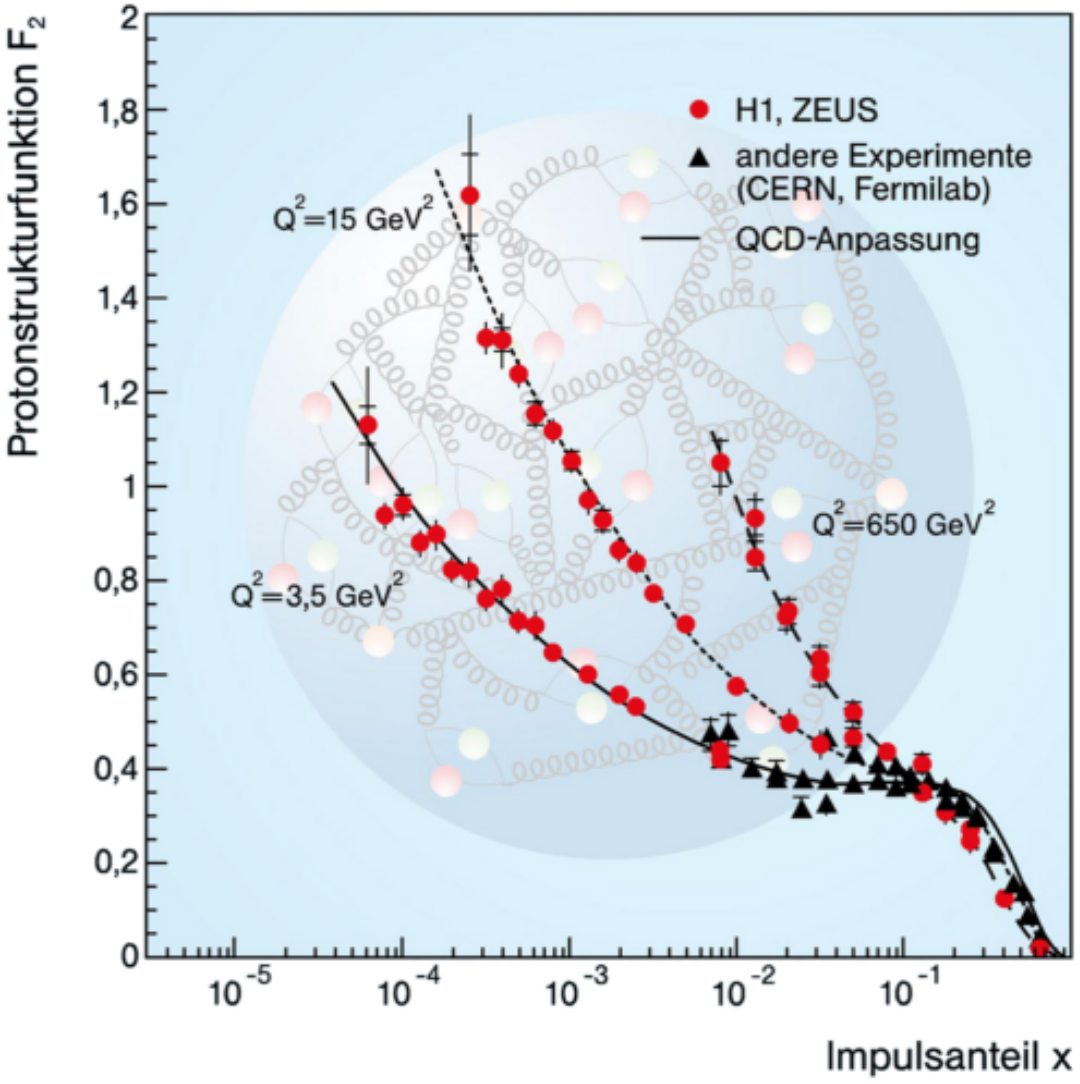


Abbildung 2.3: Skalenbrechung bei hohem Q^2. Die Strukturfunktion ist nicht mehr unabhängig von Q^2. [11]

3. Technische Grundlagen

Im vorhergehenden Kapitel wurde deutlich, dass ein am Target gestreutes Strahlmyon ein Indikator für eine physikalisch interessante Target-Reaktion ist. Deshalb sollten die Daten der Detektoren des COMPASS-Spektrometers ausgelesen und gespeichert werden, sobald ein solches Myon detektiert wird. Für diese Aufgabe gibt es im COMPASS-Experiment neben den hochauflösenden Spurdetektoren auch mehrere sog. Myonen-Trigger, die zwar eine deutlich geringere Auflösung besitzen, dafür aber nahezu in Echtzeit durch ein Signal anzeigen können, dass ein am Target gestreutes Myon registriert wurde.

In diesem Kapitel werden alle Komponenten beschrieben, die für das technische Verständnis eines solchen Triggers benötigt werden. Zunächst wird der Szintillationsdetektor (im Folgenden als Hodoskop bezeichnet) vorgestellt, mit dessen Hilfe ionisierende Teilchen lokalisiert werden können. Anschließend wird das Konzept des Meantimers eingeführt, mit dem sich auch der Zeitpunkt bestimmen lässt, zu dem die Teilchenposition festgestellt wurde. Danach wird die Koinzidenzschaltung erklärt, die es ermöglicht, anhand der Orts- und Zeitinformationen von *zwei* Hodoskopen den Pfad eines durch sie hindurchfliegenden Teilchens zu prüfen[1].

Im Rahmen der vorliegenden Diplomarbeit wurden die für ein neues Trigger-System benötigten Meantimerschaltungen und die Koinzidenzschaltung auf einem FPGA programmiert. Die externen Schnittstellen und die notwendigen Versorgungsspannungen für diesen FPGA werden durch das sog. GANDALF[2]-Board bereitgestellt, das im letzten Abschnitt dieses Kapitels vorgestellt wird. Der FPGA selbst wird im Detail in Kapitel *5* vorgestellt.

[1] Das Funktionsprinzip des Trigger-Systems inklusive des neuen LAS-Triggers wird eingehend in Kapitel *4.4* beschrieben.

[2] Generic Advanced Numerical Device for Analytic and Logic Functions

3.1 Das Hodoskop

Als Hodoskop wird ein Verbund aus mehreren Szintillatorstreifen bezeichnet. In diesen Streifen erzeugt ein hindurchfliegendes ionisierendes Teilchen (z.B. ein Myon) aufgrund des Szintillationseffekts einen Lichtblitz, der dann über Photomultiplier an den Stirnseiten in ein messbares Detektorsignal umgewandelt wird. Auf diese Weise ist es möglich, die y-Koordinate des Myons zu bestimmen. Wie in Abbildung *3.1* zu erkennen ist, hängt die Genauigkeit der Ortsbestimmung von der Geometrie der verwendeten Streifen ab.

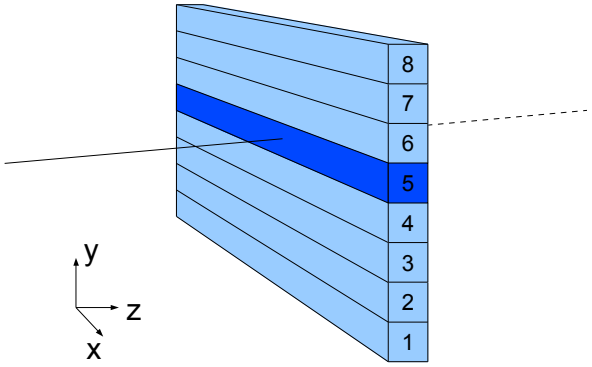


Abbildung 3.1: Die vereinfachte Darstellung des Hodoskops zeigt die Abhängigkeit des Auflösungsvermögens von der Geometrie der einzelnen Streifen.

Das analoge Ausgangssignal der Photomultiplier wird meist für die weitere Verarbeitung mit Hilfe sog. Diskriminatoren digitalisiert (z.B. um die Ereignisse zu zählen).

3.1.1 Szintillationseffekt

Einige Materialien besitzen die Eigenschaft, dass ihre Moleküle durch ionisierende Teilchen auf ein höheres Energieniveau gehoben werden und diese Anregungsenergie anschließend in Form von Licht im UV-Bereich wieder abgeben. Dieser Effekt wird als Szinillation[3] bezeichnet.

Da UV-Licht nur eine sehr begrenzte Ausbreitungsreichweite besitzt, wird dem Szintillatormaterial noch ein zweites fluoreszierendes Material beigemischt, um die Wellenlänge in den sichtbaren Bereich zu verschieben. Die für diese Diplomarbeit relevanten Szintillationsdetektoren verwenden den Plastikzintillator BC408. Er basiert auf Polyvinyltoluen und ist leicht formbar, sodass die Streifen sehr einfach hergestellt werden können.

[3] lat. scintillare: funkeln, flackern

3.1.2 Photomultiplier

Ein Photomultiplier besteht aus einer Vakuumröhre, an deren Vorderseite sich eine Photokathode befindet. Ein einlaufendes Photon schlägt aufgrund des Photoeffekts aus dieser Kathode ein Elektronen heraus. Dieses Photoelektron wird durch ein elektrisches Feld in Richtung einer Elektrode beschleunigt. Durch den Aufprall werden dort eine Vielzahl von Sekundärelektronen herausgelöst, die dann wiederum zur nächsten Elektrode hin beschleunigt werden. Dieser Prozess wiederholt sich, bis die Elektronen schließlich die Anode erreichen. Der schematische Aufbau eines Photomultipliers ist in Abbildung *3.2* dargestellt.

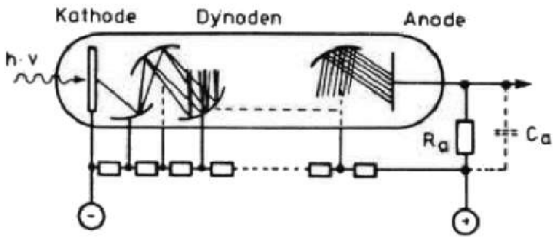


Abbildung 3.2: Schematischer Aufbau eines Photomultipliers.

Die benötigten, relativ zueinander ansteigenden Potentiale an diesen sog. Dynoden werden mit einem Spannungsteiler eingestellt. Je nach eingestellter Beschleunigungsspannung ist das Ausgangssignal proportional zur Anzahl der eingestreuten Photonen und damit proportional zur Intensität des Lichts.

3.1.3 Diskriminatoren

Diskriminatoren sind Schwellwertschalter, d.h. sie liefern ein genormtes Ausgangssignal, sobald das Eingangssignal oberhalb einer einstellbaren Schwelle liegt.

Eine einfache Variante ist der sog. Vorderflanken Diskriminator (VFD). Wie in Abbildung *3.3* zu erkennen ist, hängt hier das Ausgangssignal von der Höhe des Eingangssignals ab: Wenn Signale unterschiedlicher Höhe ihr Maximum in der gleichen Zeit erreichen, überschreiten Signale mit einer großen Amplitude die Schwelle früher als Signale mit einer kleineren Amplitude. Durch diesen time-walk wird die Zeitinformation des Eingangssignals verfälscht.

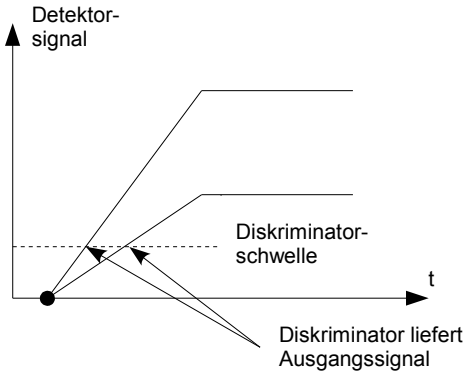


Abbildung 3.3: Funktionsprinzip eines Vorderflanken Diskriminators (VFD) für zwei Signale mit unterschiedlicher Amplitude.

Bei dem sog. Constant Fraction Diskriminator (CFD) wird das Eingangssignal zunächst dupliziert. Das eine Signal wird um eine feste Zeit verzögert, das andere wird gedämpft und invertiert. Wie durch Abbildung *3.4* ersichtlich ist, ergibt sich durch die Addition der beiden so erzeugten Signale ein als Diskriminatorschwelle nutzbarer Nulldurchgang, der weitgehend unabhängig von der Amplitude des ursprünglichen Eingangssignals ist.

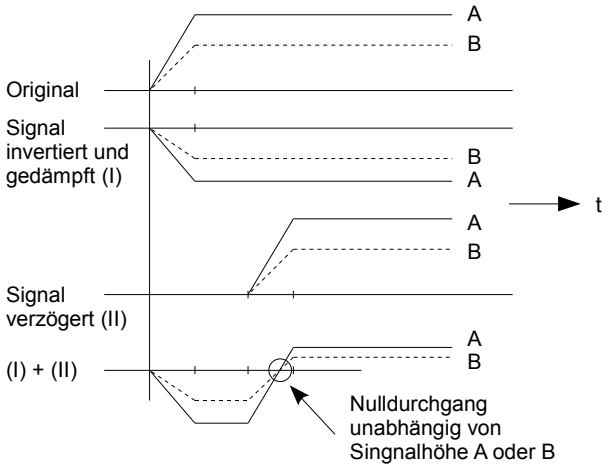


Abbildung 3.4: Funktionsprinzip eines Constant Fraction Diskriminators (CFD) für zwei Signale mit unterschiedlicher Amplitude.

3.2 Meantimer

Bei einseitig ausgelesenen Szintillatorstreifen werden Ereignisse an verschiedenen Positionen innerhalb des Streifens aufgrund der endlichen Ausbreitungsgeschwindigkeit der Photonen ($\approx$ 10cm/ns im Szintillatormaterial) nicht zeitstabil registriert (s. Abbildung *3.5*). Der Ansatz der einseitigen Auslese ist daher nur für kurze Streifen ausreichend, sofern der entstehende Zeitfehler keinen Einfluss auf die Auslese der anderen Detektoren hat.

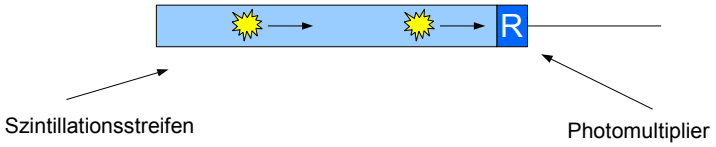


Abbildung 3.5: Zwei gleichzeitige Ereignisse in einem einseitig ausgelesenen Szintillatorstreifen werden nicht gleichzeitig registriert.

Bei längeren Streifen genügt das jedoch nicht mehr. Sie werden dann beidseitig ausgelesen und aus den Zeitinformationen beider Photomultiplier-Signale (t_L und t_R) der eigentliche Ereigniszeitpunkt t^0 rekonstruiert. Wenn die unterschiedlichen Laufzeiten innerhalb des Szintillators vom Auftreffpunkt bis hin zum linken und rechten Photomultiplier mit Δ_{SL} bzw. Δ_{SR} bezeichnet werden (s. Abbildung *3.7*), dann lässt sich t^0 wie folgt berechnen:

$$t_L + t_R = t^0 + \Delta_{SL} + t^0 + \Delta_{SR} \tag{3.1}$$

$$= 2t^0 + \Delta_{SL} + \Delta_{SR}\,, \tag{3.2}$$

$$
\begin{aligned}
t^0 &= \frac{t_L + t_R}{2} - \frac{\Delta_{SL} + \Delta_{SR}}{2} && (3.3)\\
&= \frac{t_L + t_R}{2} - \frac{\Delta_S}{2} && (3.4)\\
&= \frac{t_L + t_R}{2} - \frac{l}{2v} && (3.5)\\
&= \frac{t_L + t_R}{2} - \frac{l \cdot n}{2c}\,. && (3.6)
\end{aligned}
$$

Die Summe $\Delta_S = \Delta_{SL} + \Delta_{SR}$ ist konstant und wird nur durch die Länge l des Streifens und den Brechungsindex n des Szintillators bestimmt.

Demnach kann aus den Zeitinformationen des linken und des rechten Photomultipliers der eigentliche Ereigniszeitpunkt t^0 innerhalb des Szintillatorstreifens berechnet werden. Für diese Aufgabe wurden bereits verschiedene elektronische Schaltungen entwickelt, die als Meantimer bezeichnet werden. Sie können aufgrund ihrer Natur jedoch nur ein zu t^0 konstant verschobenes Ausgangssignal t^* liefern; dieser Offset kann in der nachgeschalteten Elektronik aber sehr leicht ausgeglichen werden. Nutzt man zwei Meantimer, um die t^0's von zwei Szintillatorstreifen miteinander zu vergleichen, wird dieser Offset bedeutungslos. Zwei solche Meantimerschaltungen werden im Folgenden kurz vorgestellt.

3.2.1 Kondensator - Schaltung

Bei dieser Schaltung nach [5, 36] wird mit jedem der beiden Detektorsignale, die den Meantimer zu den Zeitpunkten t_L und t_R erreichen, der Aufladevorgang eines Kondensators ausgelöst. Sobald die Summe der Kondensatorspannungen die Schwelle U_{th} erreicht, wird ein Ausgangssignal erzeugt. Die ist der gesuchte Zeitpunkt t^*. Die Schaltung ist so konzipiert, dass der Aufladestrom konstant gehalten wird und die Spannung gemäß

$$U_L = \frac{I}{C} \int_{t_L}^{t^*} dt = \frac{I}{C} \left(t^* - t_L\right) \tag{3.7}$$

linear ansteigt (s. Abbildung *3.6*). Für die Summe der Kondensatorspannungen gilt:

$$U_{th} = U_L + U_R = \frac{I}{C} \left(2 \cdot t^* - t_L - t_R\right)\,, \tag{3.8}$$

wodurch sich nach Gleichung *3.6* für t^* der geforderte konstante Offset zum eigentlichen Ereigniszeitpunkt t^0 ergibt:

$$t^* = \frac{C \cdot U_{th}}{2 \cdot I} + \frac{t_L + t_R}{2} = t^0 + \underbrace{\frac{C \cdot U_{th}}{2 \cdot I} + \frac{l \cdot n}{2 \cdot c}}_{\text{konstant}}\,. \tag{3.9}$$

Da die Kondensatoren zunächst wieder entladen werden müssen, ergibt sich eine von den Kenndaten der Kondensatoren abhängige Totzeit.

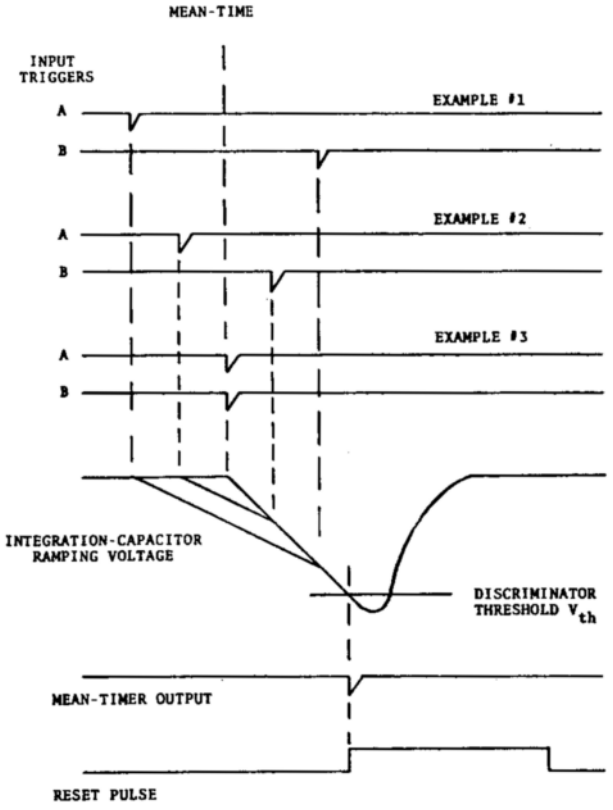


Abbildung 3.6: Verlauf der Kondensatorspannungen bei drei Ereignissen mit unterschiedlichen Eingangssignaldifferenzen. [5]

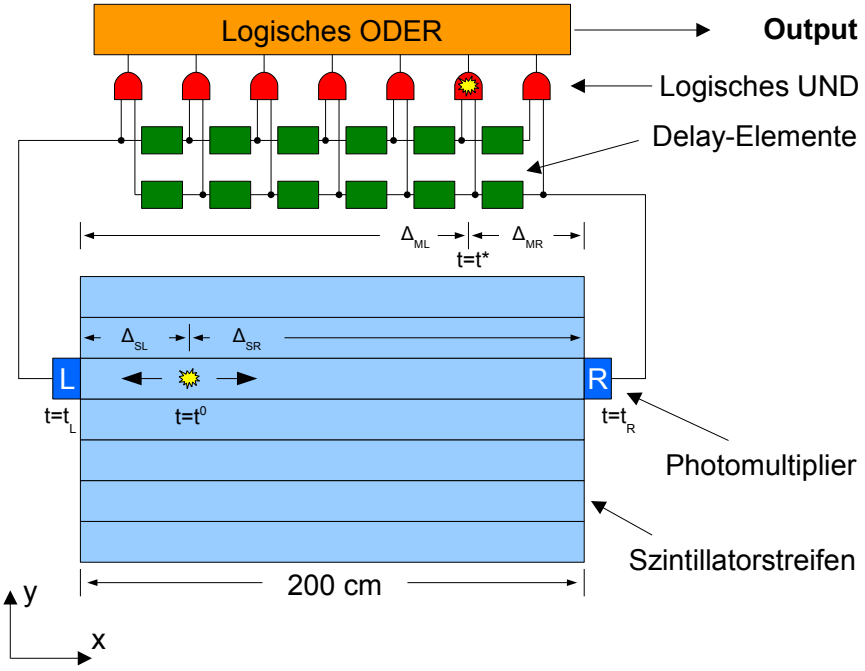


Abbildung 3.7: Funktionsprinzip eines TDL-Meantimers nach Faust und Larsen. [14]

3.2.2 Tapped Delay Line - Schaltung

Eine Tapped Delay Line (TDL, s. Abbildung *3.8*) erzeugt ausgehend von einem Startsignal periodische Sekundärsignale. Nach jedem der identischen Delay-Elemente können die um eine konstante Zeitspanne Δt verzögerten Signale abgegriffen werden.

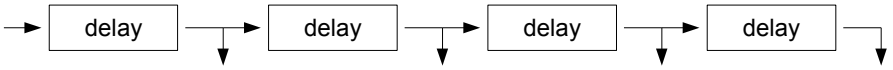


Abbildung 3.8: Schematische Darstellung einer TDL.

Werden die Detektorsignale der beiden Photomultiplier wie in Abbildung *3.7* in gegenläufige TDLs gegeben, kann über ein logisches UND der gegenüberliegenden Abgreifpunkte der Zeitpunkt t^* bestimmt werden, zu dem sich die Signale in den TDLs treffen. Die Differenz zwischen dem Ereigniszeitpunkt t^0 und t^* ist für das linke und das rechte Signal konstruktionsbedingt gleich.

Für die Laufzeiten innerhalb der TDLs gilt analog zu den Überlegungen in Kapitel *3.2*:

$$\Delta_{ML} + \Delta_{MR} = \Delta_M = \text{konstant}\,. \tag{3.10}$$

Für die Relation zwischen dem Ereigniszeitpunkt t^0 und t^* gilt daher:

$$\begin{aligned}
t^* - t^0 &= \Delta_{SR} + \Delta_{MR} && (3.11)\\
&= \Delta_S - \Delta_{SL} + \Delta_M - \Delta_{ML} && (3.12)\\
&= \Delta_S + \Delta_M - (\Delta_{SL} + \Delta_{ML}) && (3.13)\\
&= \Delta_S + \Delta_M - (\Delta_{SR} + \Delta_{MR}) && (3.14)\\
&= \Delta_S + \Delta_M - \left(t^* - t^0\right) && (3.15)\\
&= \underbrace{\frac{1}{2}(\Delta_S + \Delta_M)}_{\text{konstant}}\,. && (3.16)
\end{aligned}$$

Das eigentliche Ausgangssignal des Meantimers hat - bedingt durch die Laufzeiten von den einzelnen UND-Gattern zum globalen ODER-Gatter - eine zusätzliche Verzögerung. Ist diese Verzögerung für alle UND-Gatter gleich, hat das Ausgangssignal relativ zum eigentlichen Ereigniszeitpunkt t^0 den geforderten konstanten Offset.

Im Gegensatz zu der in Kapitel *3.2.1* beschriebenen Kondensator-Schaltung ist das Ausgangssignal eines TDL-Meantimers gerastert. Die Auflösung beträgt $\frac{1}{2}\Delta t$. Außerdem ist zu beachten, dass bei zwei Ereignissen mit einem Abstand $T < \Delta_M$ zu dem eigentlichen Meantimer-Signal einige zusätzliche Fehlsignale erzeugt werden, da sich mehrere Impulse in den TDLs befinden. Die Totzeit eines vorgeschalteten Diskriminators sollte also größer als Δ_M sein. Eine Totzeit im eigentlichen Sinne hat der TDL-Meantimer nicht.

3.3 Koinzidenzschaltung

Unter einer Koinzidenzschaltung versteht man die Prüfung mehrerer digitaler Signale auf zeitliche Gleichheit. Die einfachste Bauform ist daher ein logisches UND-Gatter, das ein Ausgangssignal liefert, wenn alle angelegten Prüfsignale logisch 1 sind. Für den LAS-Trigger wird eine Koinzidenzschaltung benötigt, bei der nicht nur einzelne Signale, sondern zwei Gruppen von Signalen auf Koinzidenz geprüft werden. Diese Prüfung ist genau dann positiv, sobald mindestens ein Signal aus der ersten Gruppe zeitgleich mit einem Signal aus der zweiten Gruppe in die Koinzidenzschaltung einläuft. In Abbildung *3.9* ist eine solche Schaltung für zwei Gruppen mit jeweils drei Signalen gegeben. Es ergeben sich in diesem Beispiel neun unabhängige 2-Signal-Koinzidenzprüfungen.

Abbildung *3.9* zeigt aber noch ein weiteres wesentliches Funktionsmerkmal der benötigten Koinzidenzschaltung: Über zusätzliche Steuersignale an jedem UND-Gatter lassen sich die Koinzidenzen individuell unterdrücken. Ist das Steuersignal logisch 0, bleibt das Ausgangssignal unabhängig von der eigentlichen Koinzidenzprüfung ebenfalls logisch 0 und die entsprechende Kombination zwischen den beiden Gruppen ist unterdrückt. Die Steuersignale werden dabei über eine sog. Koinzidenzmatrix konfiguriert.

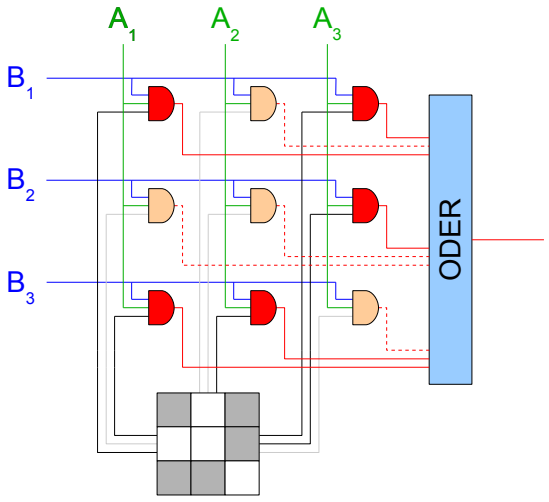


Abbildung 3.9: Vereinfachte Darstellung einer 3×3 Koinzidenzschaltung. Die grauen Felder in der Koinzidenzmatrix markieren die erlaubten Kombinationen, alle anderen Koinzidenzen werden aufgrund des fehlenden Steuersignals unterdrückt.

3.4 GANDALF-Board

Das GANDALF-Board wurde ursprünglich entwickelt, um die Signale eines Rückstoß-Proton-Detektors (RPD) auszulesen und zu verarbeiten (wurde 2008 und 2009 bei COMPASS eingesetzt). Die Arbeitsgruppe Königsmann der Freiburger Universität hatte sich 2008 für eine modulare und flexibel einsetzbare Lösung entschieden und die Signalanalyse auf einen frei programmierbaren FPGA ausgelagert. Das GANDALF-Board kann daher auch für die Aufgabenstellung dieser Diplomarbeit verwendet werden: Die Meantimer- und Koinzidenzschaltungen werden auf dem FPGA umgesetzt und das GANDALF-Board stellt alle für das COMPASS-Experiment benötigten Schnittstellen zur Verfügung.

Das GANDALF-Board (FPF 330) ist ein VME64x/VXS[4] Modul ($233 \times 160\text{mm}^2$, Bauhöhe 6U) und gemäß der VME64x-Spezifikation ANSI/VITA 1.1-1997 gefertigt [33]. VME64x ist eine schnellere Erweiterung des weit verbreiteten VME-Standards [28], der die Kommunikation zwischen den Modulen eines Crates sowie mit dem Crate-PC ermöglicht. Dieser PC kann über diverse Netzwerkprotokolle angesteuert werden, sodass das GANDALF-Board fernkonfigurierbar ist.

Es besitzt zwei Steckplätze für I/O-Karten, die für diese Diplomarbeit mit zwei 64-Kanal-LVDS-Input-Karten bestückt sind, sodass insgesamt 128-Eingangssignale verarbeitet werden können. Weiterhin bietet jede dieser Input-Karten einen NIM-Eingang und 2 NIM-Ausgänge (s. Abbildung *3.10*).

Das CPLD-Interface fungiert als Schnittstelle zwischen dem FPGA und dem VME-Bus. Es wird in dieser Diplomarbeit genutzt, um die wichtigsten Einstellungen der Meantimer und der Koinzidenzschaltung über ein Web-Interface vornehmen zu können. Diese Konfigurationsdaten werden jedoch nur im Speicher des FPGAs (8Mbit Block-RAM) selbst abgelegt. Das ebenfalls zur Verfügung stehende Memory-Interface, mit dem 4GB DDR2-RAM und 144MB QDRII-RAM ansprechbar sind, wird in dieser Diplomarbeit nicht genutzt.

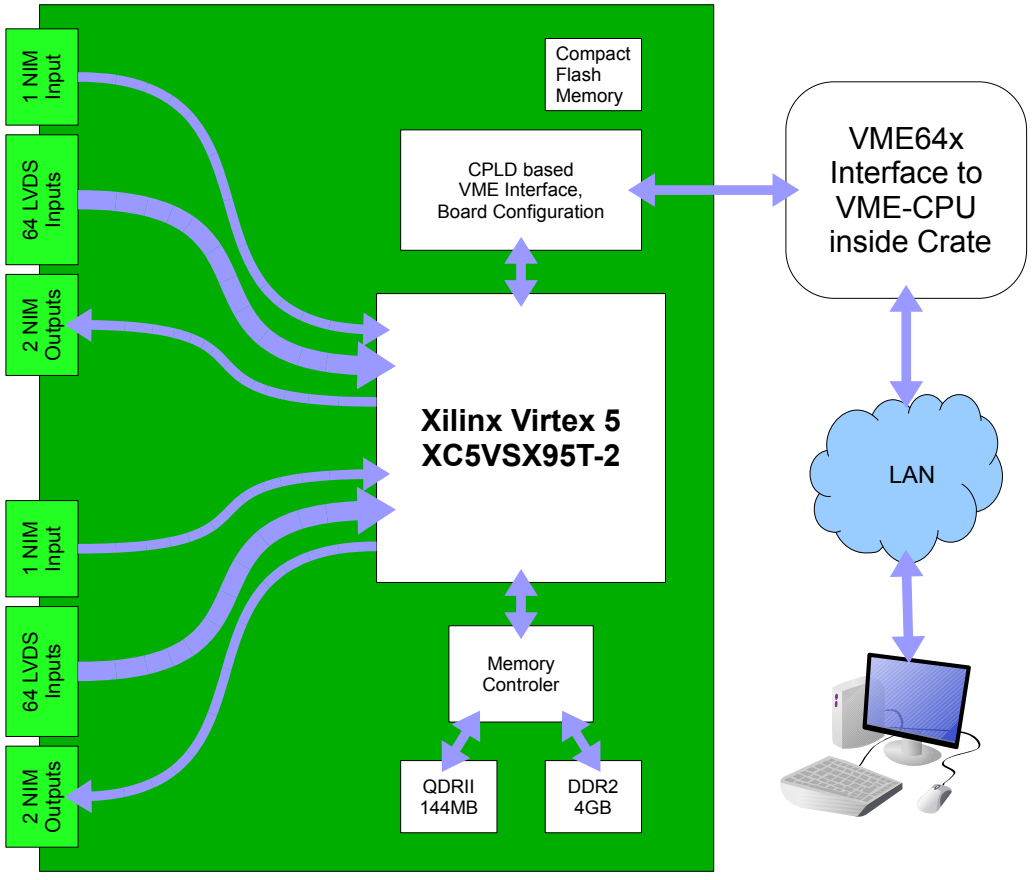


Abbildung 3.10: Schematische Darstellung des GANDALF-Boards. [33]

[4] Aufgrund der hohen Anzahl technischer Akronyme auf dieser Seite wurde auf erklärende Fußnoten verzichtet. Alle Abkürzungen sind im Abkürzungsverzeichnis zu finden.

4. Das COMPASS-Experiment

Das COMPASS-Experiment ist ein Fixed-Target-Experiment und befindet sich in der North Area des CERN in Prévessin (s. Abbildung *4.1*). Zentrales Ziel wie schon bei den Vorgänger-Experimenten EMC und SMC ist die Untersuchung der Nukleonspinstruktur. Dazu wird ein 160GeV Myonstrahl auf ein fest installiertes NH_3 Target gelenkt. Aus den rekonstruierten Spuren der detektierten Myonen und Hadronen können die gesuchten Impuls- und Spinverteilungen für die Partonen bestimmt werden. Während der Strahlzeit 2010 liegt ein besonderer Schwerpunkt auf der Messung der transversalen Spinstrukturfunktionen.

Ein weiterer Forschungsschwerpunkt neben dem Myonprogramm ist das Hadronprogramm. Dabei nutzt man einen Pionstrahl und beschäftigt sich mit der Quantenchromodynamik im niederenergetischen Bereich. Für diese Diplomarbeit ist das Hadronprogramm jedoch nicht relevant und es wird daher nicht weiter betrachtet.

Im Folgenden werden die drei Hauptkomponenten des COMPASS-Experiments vorgestellt: Der Bereich der Strahlführung (Myonproduktion und Vermessung des Strahls), das Target und das zweistufige Spektrometer mit seinen Detektoren. Abschließend wird das Trigger-System eingeführt, das interessante physikalische Ereignisse von unwichtigem Untergrund unterscheidet und die Speicherung der Detektorsignale auslöst.

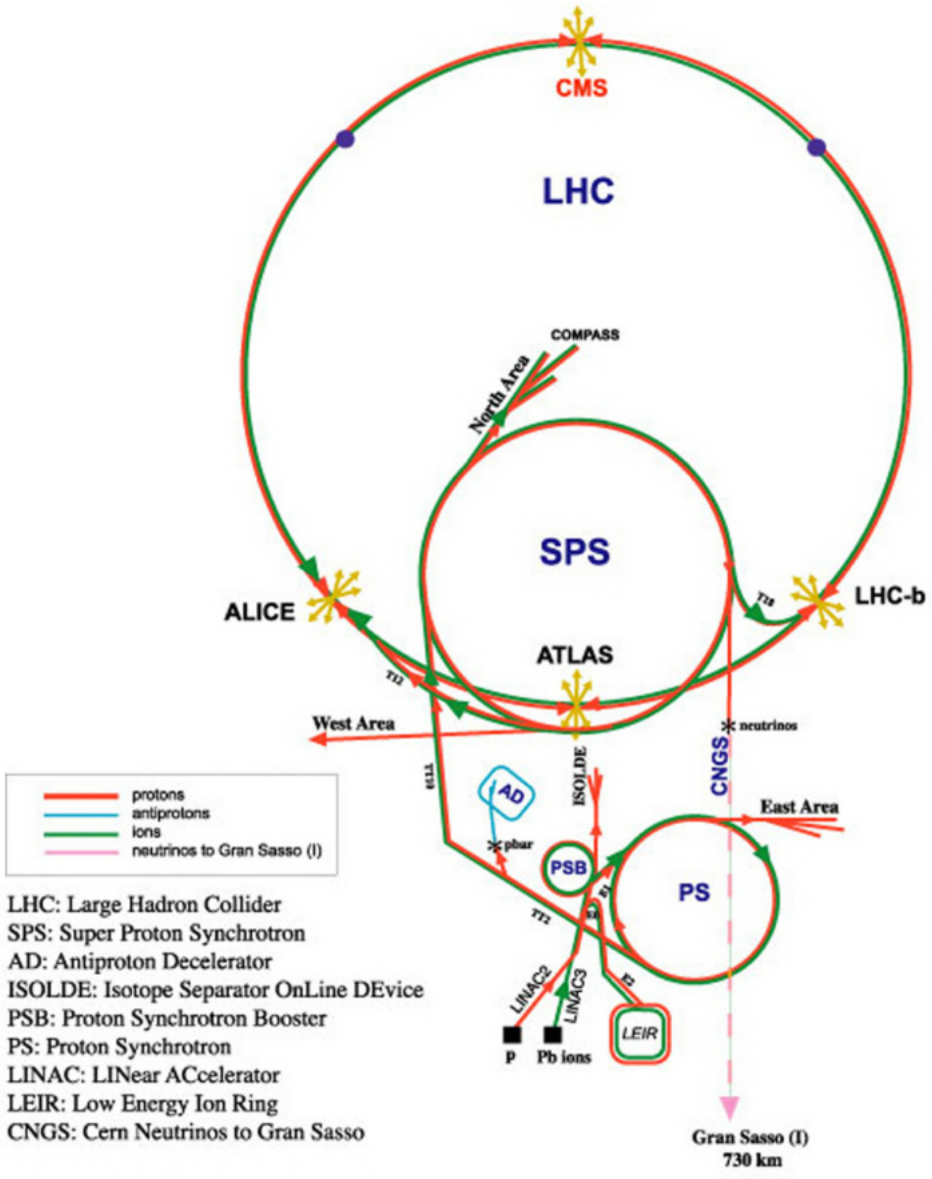


Abbildung 4.1: Die verschiedenen Beschleuniger am CERN. Die Darstellung ist nicht maßstabsgetreu.

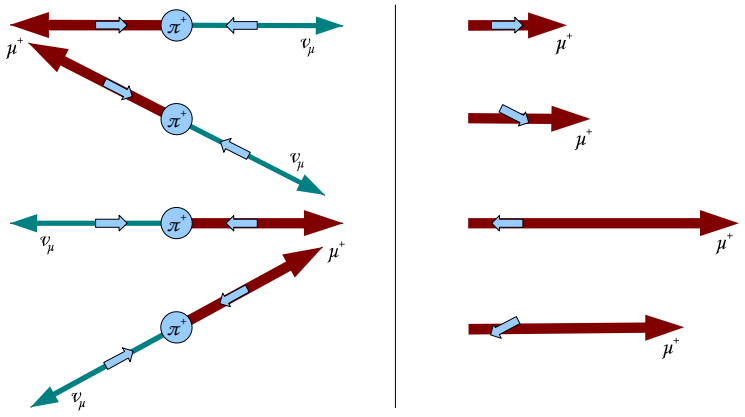


Abbildung 4.2: Im Ruhesystem des Pionzerfalls besitzen sowohl das Myon-Neutrino als auch das Myon eine antiparallele Spinkonfiguration (linke Abbildung). Der Myonstrahl im Laborsystem besitzt hingegen keine ausgezeichnete Polarisation mehr (rechte Abbildung).

4.1 Strahlführungsbereich

4.1.1 Der polarisierte Myonstrahl

Für die North Area wird in unregelmäßigen[1] Abständen aus dem SPS[2] ein 450GeV Protonen-Strahl extrahiert und in die M2 Beamline geleitet. Eine solche gleichmäßige Extraktion dauert 9.6s an und wird als Spill bezeichnet. In der Beamline wird der Protonenstrahl dann auf das aus Beryllium bestehende T6-Produktionstarget geleitet. Durch den Beschuss entstehen dort neben Kaonen auch Pionen, von denen ca. 99% [13] über die schwache Wechselwirkung gemäß

$$\pi^+ \rightarrow \mu^+ + \nu_\mu$$

zu Myonen zerfallen. Hinter dem Produktionstarget werden durch mehrere Absorber alle noch vorhandenen Hadronen aus dem Strahl entfernt, die verbleibende Verschmutzung mit Hadronen beträgt dann noch ca. 1%. Über mehrere Umlenkmagnete entlang der Beamline (blaue Dreiecke in Abbildung *4.3*) können die produzierten Myonen impulsselektiert werden (zwischen 40GeV und 200GeV). Über die Dicke des Beryllium-Targets, dass sich auf Werte zwischen 40mm und 500mm einstellen lässt, kann die Intensität des Myonstrahls bestimmt werden. Die maximal erreichbare Intensität beträgt während der Strahlzeit 2010 ca. $3.5 \cdot 10^8$ Myonen pro Spill (mit dem 500mm-Target).

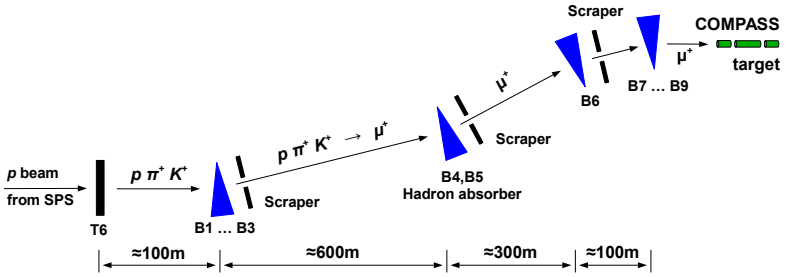


Abbildung 4.3: Verlauf der Beamline M2 vom SPS (unterirdisch) bis zum COMPASS-Target in Halle 888 (oberirdisch). [18]

Aufgrund der Impulserhaltung fliegen die durch den Pionenzerfall entstehenden Leptonen (Myon und Myon-Neutrino, beide Spin $\frac{1}{2}$) im Ruhesystem in entgegengesetzter Richtung auseinander. Aus der maximalen Paritätsverletzung folgt für die nahezu masselosen Neutrinos eine antiparallele Spinkonfiguration (Linkshändigkeit). Da die Spins von Myon und Myon-Neutrino sich zu Null addieren müssen - das Pion hat einen Spin 0 - sind im Ruhesystem auch die Spins der Myonen zu fast 100% antiparallel polarisiert.

Im Laborsystem dagegen ergibt sich für den Teilchenstrahl zunächst keine ausgezeichnete Polarisation (s. Abbildung *4.2*). Erst durch eine Impulsselektion in der Beamline können Myonen mit gleicher Polarisation herausgefiltert werden: Durch die Selektion der langsamen 40GeV-Myonen ergibt sich eine parallele Strahlpolarisation, durch die Selektion der schnellen 160GeV-Myonen erhält man die hauptsächlich genutzte antiparallele Strahl-Polarisation (ca. 80% polarisiert).

[1] Der SPS-SuperCycle passt sich dyn. den Anforderungen von LHC und CNGS an. Die Abstände der Protonen-Extraktion aus dem SPS für die M2-Beamline variieren zwischen 34s und 60s.

[2] Super Proton Synchrotron

4.1.2 Die Beam Momentum Station

Die Impulsselektion der Spektrometer in der M2-Beamline streut um ca. 5% [12], für die Datenanalyse muss jedoch der exakte Impuls bekannt sein. Die Beam Momentum Station (BMS) besteht aus je drei Hodoskopen vor und nach einem der letzten Ablenkmagneten der Beamline[3]. Über eine Rekonstruktion der Ablenkwinkel wird der Impuls jedes einfliegenden Myons mit einer Genauigkeit von 0.3% [27] bestimmt.

4.2 Das polarisierte Target

Mit dem Myonprogramm werden die Asymmetrien der Wirkungsquerschnitte bestimmt, die aufgrund entgegengesetzter Targetpolarisationen entstehen. Dies wird sowohl für die longitudinalen Targetpolarisationen ($\Rightarrow$ und $\Leftarrow$) als auch für die transversalen Targetpolarisationen ($\Uparrow$ und $\Downarrow$) durchgeführt:

$$A_{\text{longit.}} = \frac{\mathrm{d}\sigma^{\leftarrow\Rightarrow} - \mathrm{d}\sigma^{\leftarrow\Leftarrow}}{\mathrm{d}\sigma^{\leftarrow\Rightarrow} + \mathrm{d}\sigma^{\leftarrow\Leftarrow}}\,, \qquad A_{\text{transv.}} = \frac{\mathrm{d}\sigma^{\leftarrow\Uparrow} - \mathrm{d}\sigma^{\leftarrow\Downarrow}}{\mathrm{d}\sigma^{\leftarrow\Uparrow} + \mathrm{d}\sigma^{\leftarrow\Downarrow}}\,.$$

Der kleine Pfeil ($\leftarrow$) steht für die antiparallele Strahlpolarisation aus dem Pionenzerfall innerhalb der M2 Beamline. Um z.B. Messungen für die beiden transversalen Targetpolarisationen bei identischem Myonenfluss durchführen zu können, werden drei hintereinander angeordnete Targetzellen mit abwechselnden Polarisationen verwendet ($\Uparrow\Downarrow\Uparrow$ oder $\Downarrow\Uparrow\Downarrow$). Diese Targetzellen haben einen Durchmesser von 4cm, die äußeren beiden sind 30cm lang und die mittlere ist 60cm lang. Um apparative Asymmetrien zu kompensieren, welche die physikalisch interessanten Asymmetrien überlagern könnten, wird die Spinpolarisation aller drei Targetzellen in regelmäßigen Abständen umgeklappt.

Um einem Ereignis eine Targetpolarisationen zuordnen zu können, muss die Targetzelle bestimmt werden, an der die Wechselwirkung erfolgte. Dazu wird mit Hilfe der Spurdetektoren die Trajektorie des gestreuten Myons oder der primär produzierten Hadronen bestimmt.

Nach dem Curieschen Gesetz (μ ist das magnetische Moment der Targetteilchen und k_B ist die Boltzmankonstante)

$$P = \tanh\left(\frac{\mu B}{k_B T}\right)$$

werden für eine möglichst hohe Polarisation P eine sehr tiefe Temperatur und ein starkes Magnetfeld benötigt [32]. Das Target ist daher von einem 2.5T starken supraleitenden Magneten und einem sog. Dilution Refrigerator umgeben, der das Target auf 50mK herunterkühlt. Der Magnet kann zunächst nur eine longitudinale Polarisation für die Elektronen erwirken, die dann über die DNP[4]-Methode auf die Protonen übertragen wird. Für Messungen mit transversaler Targetpolarisation wird ein zweiter, senkrecht zum Strahl ausgerichteter 0.5T starker Magnet eingesetzt, der die Spins aufrichtet.

Für das Myonprogramm 2010 werden mit NH_3 gefüllte Targetzellen verwendet, mit denen ein Polarisationsgrad von 80% erreicht werden kann.

[3] Magnet B6, s. Abbildung *4.3*

[4] Dynamic Nuclear Polarisation, diese Methode ist ausführlich in [2] beschrieben.

4.3 Das Spektrometer

Abbildung *4.4* zeigt den Aufbau des aus zwei Stufen bestehenden COMPASS-Spektrometers. In beiden Stufen gibt es mehrere Detektoren, die zur Spurrekonstruktion eingesetzt werden, sowie ein elektromagnetisches und ein hadronisches Kalorimeter, die zur Bestimmung der Teilchenenergien aber auch zur Teilchenidentifikation verwendet werden.

Die erste Stufe kann gestreute Myonen und produzierte Hadronen nachweisen, die den ersten Spektrometermagneten (SM1) unter einem Winkel von ±180mrad verlassen. Die Absorber, Kalorimeter und Hodoskope der ersten Spektrometerstufe haben jedoch im zentralen Strahlbereich ein Loch, sodass Teilchen, die den SM1 unter einem Winkel von ±30mrad verlassen zusätzlich in die zweite Spektrometerstufe gelangen können.

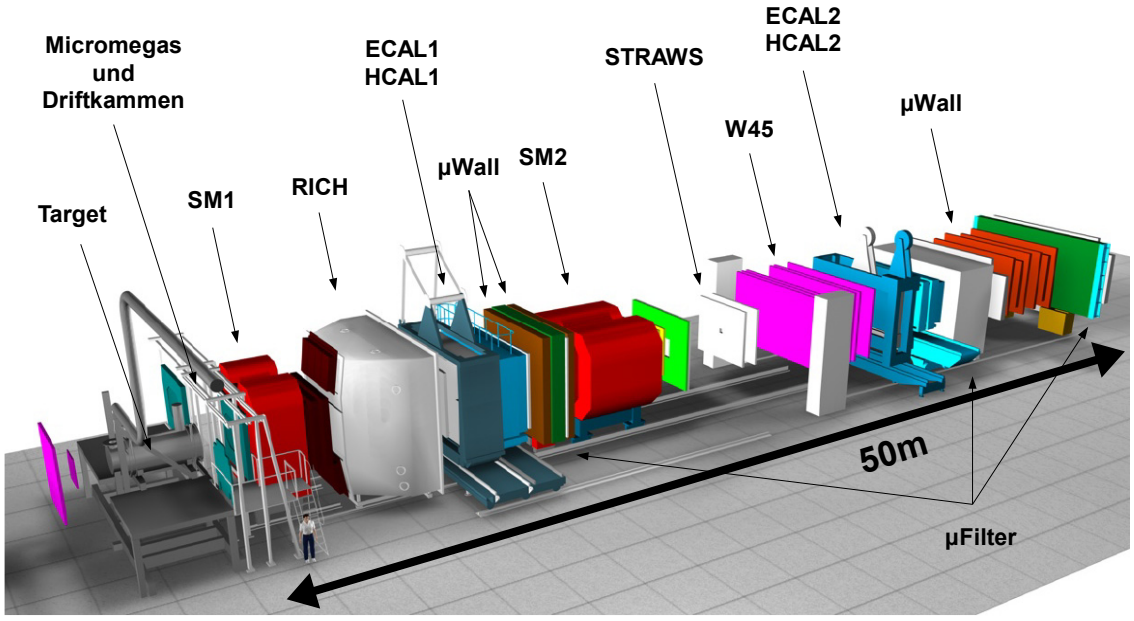


Abbildung 4.4: Der COMPASS-Aufbau zur Strahlzeit 2010. Nicht eingetragen sind die GEMs, MWPCs, einige Driftkammern und die SciFis, die sich über die gesamte Länge des Spektometers verteilen. Die ebenfalls fehlenden Trigger-Hodoskope sind gesondert in Abbildung *4.9* aufgeführt. [29]

4.3.1 Spurrekonstruktion

Mit Hilfe der Spurrekonstruktionen vor und nach den Spektrometermagneten kann der eigentliche Streu- bzw. Produktionswinkel und bei geladenen Teilchen über die zusätzliche Ablenkung auch der Impuls ermittelt werden. Über die Bestimmung der vollständigen Trajektorie kann außerdem die getroffene Targetzelle bestimmt werden.

Die Spurdetektoren werden in drei Gruppen gegliedert: Die very small area trackers (VSAT) decken einen sehr kleinen Bereich unmittelbar in der Nähe der Strahlachse ab. Aufgrund der hohen Rate müssen sie eine geringe Totzeit und eine hohe Orts- und Zeitauflösung besitzen, um die Ereignisse bestimmten Spuren zuordnen zu können. Beim COMPASS-Experiment kommen dafür Siliziumstreifendetektoren und szintillierende Fasern (SciFi) zum Einsatz. In einem Abstand von 2.5cm bis 40cm von der Strahlachse befinden sich die small area trackers (SAT), das sind hochauflösende Gasionisationsdetektoren (GEMs und Micromegas). Die large area trackers (LAT)

schließlich liegen im Randbereich und bestehen aus Vieldrahtproportionalkammern (MWPCs[5]) und Driftkammern (W45 und STRAWs).

4.3.2 Kalorimeter

Mit den elektromagnetischen Kalorimetern (ECAL1&2) können Photonen ab einer Energie von 100MeV nachgewiesen werden. In den Bleiglasmodulen ($n \approx 1.7$) der ECALs wird das einfallende Photon über Paarbildung in ein e^+e^--Paar umgewandelt. Dieses Paar verliert seine Energie zunächst über Bremsstrahlung, aus der wieder sekundäre e^+e^--Paare entstehen. Der so entstehende elektromagnetische Schauer breitet sich weiter aus, bis die Elektronen ihre Energie nicht mehr bevorzugt über Bremsstrahlung sondern über Ionisation abgeben (ab 10MeV); es erfolgt dann keine weitere Paarbildung mehr und die Elektronen werden schließlich gestoppt. Die eigentliche Bestimmung der Energie erfolgt mit Photomultiplieren, die das Čerenkov-Licht der Schauerelektronen registrieren: Die Anzahl der Čerenkov Photonen ist proportional zur Energie des Schauers, also zur Energie des ursprünglichen Photons. Das primäre e^+e^--Paar hat nach ca. 10 Strahlungslängen (x_0) gemäß

$$\frac{E}{E_0} = \exp\left(\frac{-10x_0}{x_0}\right) < 0.0001$$

bereits über 99.99% seiner Energie verloren. Um zu gewährleisten, dass auch alle sekundär erzeugten e^+e^--Paare innerhalb des ECALs gestoppt werden, beträgt die Dicke des ECALs ca. 16 Strahlungslängen.

Die hadronischen Kalorimeter (HCAL1&2) sind sog. Samplingkalorimeter, sie bestehen alternierend aus Eisenplatten und Plastikszintillatoren. Ein durchfliegendes Hadron löst in der Eisenschicht durch Kernreaktionen einen hadronischen Schauer aus. Die elektromagnetischen Komponenten dieses Schauers werden dann in der nächsten Szintillatorschicht nachgewiesen. Die hadronischen Komponenten können in der nächsten Eisenschicht weitere hadronische Schauer erzeugen. Die Dicke von HCAL1 beträgt ca. 5 nukleare Wechselwirkungslängen (λ_I), die von HCAL2 ungefähr $7\lambda_I$. Die Schauerprozesse können aber bereits in den davor stehenden ECALs einsetzen, sodass insgesamt genügend Kernreaktionen erfolgen können und so nahezu 100% der Teilchenenergie in den HCALs deponiert wird.

4.3.3 Teilchenidentifikation

In der ersten Spektrometerstufe befindet sich ein ringabbildener Čerenkov-Detektor (RICH[6]), mit dem Pionen, Kaonen und Protonen identifiziert werden können (s. Abbildung *4.5*). Der Detektor ist mit dem Radiatorgas C_4F_{10} gefüllt, das einen - für Gase - sehr hohen Brechungsindex besitzt ($n = 1,0015$). Wenn die im Target durch tiefinelastische Streuung produzierten Hadronen den Detektor durchqueren und dabei schneller als die Lichtgeschwindigkeit in dem Radiatorgas fliegen, strahlen sie Čerenkov-Licht ab, das auf Kreise in der Fokalebene abgebildet und dort von Photomultiplieren nachgewiesen wird. Anhand der Radien der Kreise lassen sich die Teilchen identifizieren.

[5] multi-wire proportional chambers
[6] Ring Imaging CHerenkov Detector

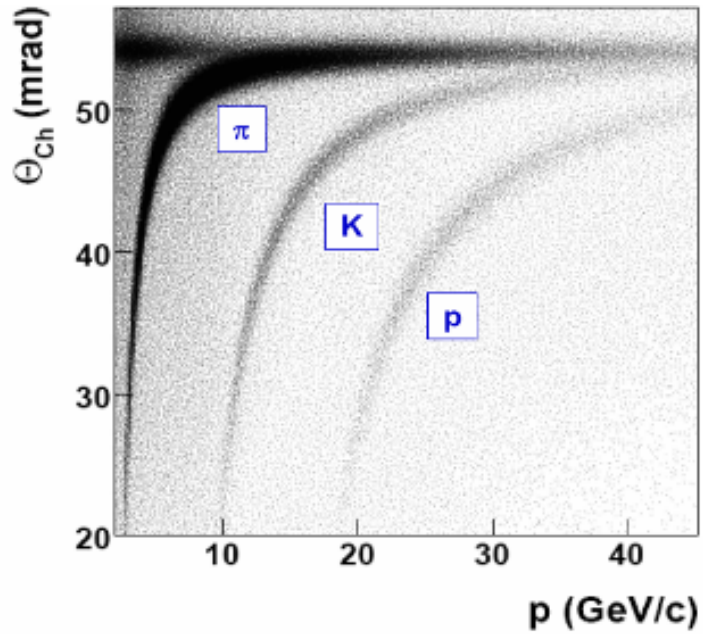


Abbildung 4.5: Identifizierung von Pionen, Kaonen und Protonen mit Hilfe des RICH-Detektors. [8]

Um die gestreuten Myonen zu identifizieren, nutzt man deren geringe Wechselwirkungswahrscheinlichkeit: Sie können einen schweren und dicken Absorber (μFilter) fast ungehindert durchdringen, während Hadronen total absorbiert werden. Der erste 60cm dicke Eisen-Absorber steht hinter dem HCAL1 und wird von zwei aus Driftröhren bestehenden μWalls umgeben. Wird ein Teilchen in beiden μWalls detektiert, kann es als Myon identifiziert werden. Ein weiterer μFilter, ein 2.4m dicker Beton-Absorber, steht hinter dem HCAL2. Ihm folgt eine μWall mit mehreren Driftkammern.

4.4 Das Trigger-System

Die Aufgabe des Trigger-Systems[7] besteht darin, die für die Analyse interessanten Ereignisse anhand einer oder mehrerer Echtzeit-Prüfungen zu erkennen und über ein Trigger-Signal die Speicherung der Detektorsignale auszulösen. Diese Selektion ist notwendig, da die Teilchenrate bei tiefinelastischen Streuexperimenten zu hoch ist, um tatsächlich alle erzeugten Detektorsignale zu speichern. Die Datenverarbeitung (DAQ[8]) wäre sowohl bzgl. der Datenmenge als auch der Datenrate überfordert. Zusätzlich ist ein erheblicher Teil der Ereignisse physikalisch uninteressanter Untergrund, der mit dem Trigger-System unterdrückt werden kann.

Jedes Trigger-System lässt sich durch seine purity und seine efficiency charakterisieren. Die purity ist der Anteil der tatsächlich brauchbaren Ereignisse an allen getriggerten Ereignissen. Bei einer purity von 100% würde bei keinem einzigen unerwünschten Ereignis ein Triggersignal ausgelöst werden. Die efficiency ist der Anteil der getriggerten und brauchbaren Ereignisse an den tatsächlich existierenden brauchbaren Ereignissen. Bei einer efficiency von 100% würde kein einziges brauchbares Ereignis verpasst werden. Eine hohe efficiency ist daher wichtiger als eine hohe purity, solange die Datenrate bzw. das Datenvolumen von der DAQ noch bewältigt werden kann.

[7] engl. trigger: auslösen
[8] Data Acquisition

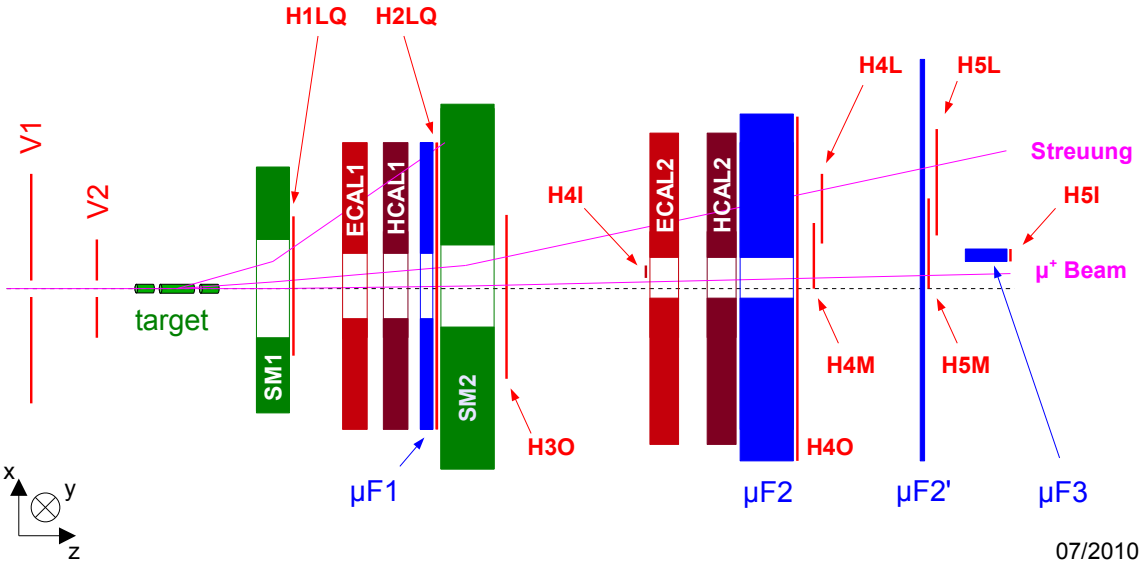


Abbildung 4.6: Trigger-Elemente des COMPASS-Experiments.

In Abbildung *4.6* sind alle Elemente des COMPASS-Trigger-Systems aufgeführt. Die beiden neuen Hodoskope H1 & H2 bilden den LAS-Trigger, H3O & H4O den OUTER-Trigger, H4I & H5I den INNER-Trigger, H4M & H5M den MIDDLE-Trigger und H4L & H5L den LADDER-Trigger. Die Hodoskope H2 und H4O sind in der Mitte geteilt, damit die einzelnen Szintillatorstreifen nicht zu lang sind. Diese Trigger bilden zusammen den Myon-Trigger. Der CALO-Trigger für Hadronen triggert auf ECAL1, HCAL1 und HCAL2. Die übrigen beiden Hodoskope sind Vetos, ein drittes Veto-Hodoskop fehlt in der Abbildung und befindet sich 20m vor dem Target.

Der Aufstellungsort der verschiedenen Trigger-Elemente im Experiment bestimmt die akzeptierten Streuwinkel und damit auch die jeweils abgedeckten kinematischen Regionen. Dies folgt aus Gleichung *2.12* (Seite 4) und ist in Abbildung *4.7* für die einzelnen Trigger dargestellt.

4.4.1 Veto-System

Das Veto-System besteht aus drei Hodoskopen, die noch vor dem Target stehen und auf Myonen im äußeren Bereich des Strahls (sog. Halo-Myonen) triggern. Die Veto-Hodoskope arbeiten nicht paarweise zusammen sondern klassifizieren Myonen als Halo allein durch die Tatsache, dass sie eines der Vetos passiert haben. Liegt ein Veto-Signal vor, werden alle anderen eventuell anliegenden Trigger-Signale für eine gewisse Zeit unterdrückt. Bei voller Strahlintensität umfasst diese Totzeit, in der keine Daten genommen werden können bis zu 20%. Um diesen Verlust zu minimieren, werden nicht alle Trigger mit dem Vetosystem gekoppelt.

4.4.2 Myon-Trigger

Mit Hilfe einer einfachen Spuranalyse kann überprüft werden, ob sich die Flugbahn eines Myons bis in das Target zurückverfolgen lässt und es sich damit um ein gestreutes Myon handeln kann. Durch die beiden Spektrometermagnete SM1 und SM2 wird die Flugbahn der Myonen in x-Richtung impulsabhängig abgelenkt, sodass eine geometrische Bestimmung nur für die in die y-z-Ebene projizierte Flugbahn möglich ist. Die Hodoskope des LAS-Triggers und des OUTER-Triggers haben horizontale Szintillatorstreifen und können daher die für die geometrische Pfadrekonstruktion

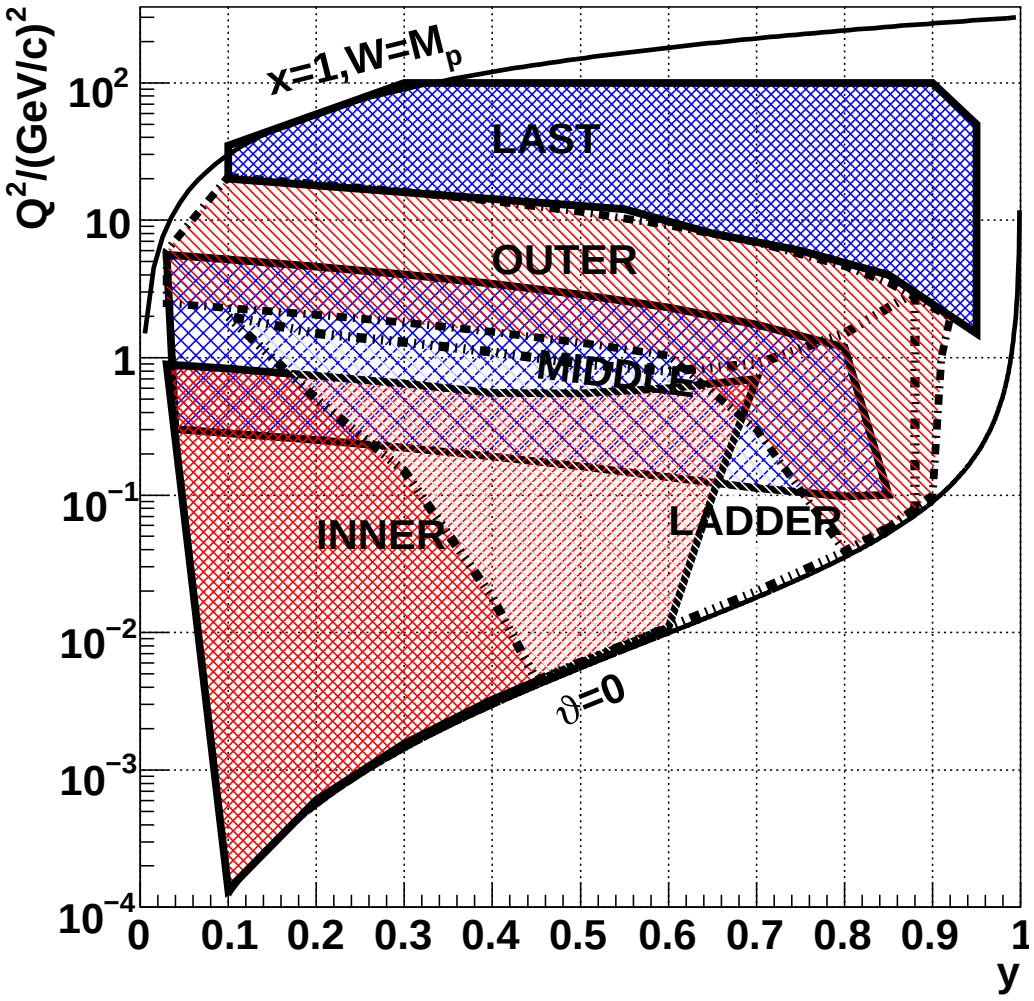


Abbildung 4.7: Übersicht der kinematischen Regionen der verschiedenen Trigger. Der LAS-Trigger ist in dieser Abbildung mit LAST abgekürzt. Grafik durch PD Dr. Jörg Pretz zur Verfügung gestellt.

benötigten y-Positionen an zwei z-Positionen bestimmen (s. auch Abbildung *4.8* auf Seite 29).

Diese auch als vertical target pointing bezeichnete Methode funktioniert jedoch nicht mehr bei sehr kleinen Streuwinkeln. Für solche Fälle nutzt man die impulsabhängige Ablenkung in der x-z-Ebene und fordert für am Target gestreute Myonen einen Energieverlust und damit eine größere Ablenkung als bei ungestreuten Myonen. Die Hodoskope des INNER-Triggers und des LADDER-Trigges haben vertikale Szintillatorstreifen, mit denen sich die für den sog. Energieverlust-Trigger benötigten x-Positionen der Myonen an zwei z-Positionen bestimmen lassen. Der MIDDLE-Trigger hat sowohl horizontale als auch vertikale Streifen und kann somit für beide Trigger-Methoden verwendet werden.

Die eigentliche Prüfung der Trigger-Bedingung erfolgt bei beiden Methoden über eine komplexe Koinzidenzprüfung zwischen den beiden zum Trigger gehörenden Hodoskopen. Aufgrund ihres bekannten Abstandes und der festen Strahlenergie ist auch die time-of-flight für ein gestreutes Myon zwischen den Hodoskopen eine bekannte Größe. Dieser Zeitunterschied wird durch längere Signalkabel für das näher am Target stehende Hodoskop ausgeglichen, sodass die Signale beider Hodoskope in

der Trigger-Baracke gleichzeitig[9] ankommen. Über die in Kapitel *3.3* beschriebene Koinzidenzschaltung und die dazugehörende Koinzidenzmatrix lassen sich all jene Pfade (Kombinationen von Szintillatorstreifen beider Hodoskope) unterdrücken, die nicht der gewünschten Trigger-Bedingung genügen (s. Abbildung *4.8*).

Weitere Informationen zu dem Myon-Trigger des COMPASS-Experiments sind in [6] zu finden.

4.4.3 Kalorimeter-Trigger

Die ECALs und HCALs sind schnell genug, sodass ihre Detektorsignale auch direkt als Trigger genutzt werden können. Die HCALs werden benutzt, um bei semi-inklusiven Messungen auf Hadronen zu triggern. Bei HCAL1 wurde nach der Installation von ECAL1 ein Rückgang der Rate festgestellt, d.h. dass einige Hadronen ihre Energie bereits im ECAL1 deponieren. Der für Hadronenereignisse gedachte CALO-Trigger umfasst daher neben den beiden HCALs auch die Signale aus dem ECAL1.

Mit ECAL1 kann zusätzlich auf die beiden Photonen aus den π^0-Zerfällen getriggert werden.

4.4.4 HCAL1-Myon-Trigger

Wenn Myonen durch die HCALs hindurchfliegen, werden sie zwar kaum beinflusst, hinterlassen aber eine deutliche Signatur im unteren Energiebereich (Landau-Verteilung). Die Energie-Schwellwerte sind normalerweise so konfiguriert, dass der HCAL1-Trigger auf diese Ereignisse nicht anspricht. Da die HCALs jeweils zwei Trigger-Ausgänge besitzen, bei denen die Schwellwerte getrennt definiert werden können, ist es aber auch möglich, nur auf diesen Peak im unteren Energiebereich zu triggern. Die beiden Schwellen werden dazu um den Myon-Peak herum platziert, die untere wird dann als Trigger und die obere als Veto bei Hadron-Ereignissen genutzt. Da bei dieser Trigger-Variante keine Targetspur geprüft werden kann, muss der Trigger mit allen drei Vetos gekoppelt werden, um Halo zu unterdrücken.

Eine zeitlang war dies die einzige Möglichkeit, um auf Myonen mit großen Streuwinkeln (großem Impulsquadratübertrag Q^2) zu triggern. Durch den neuen LAS-Trigger ist diese Region mittlerweile aber auch über den Myonen-Trigger zugänglich.

[9] Alle Triggersysteme werden zeitkalibriert und durch zusätzliche Delays eventuelle Abweichungen ausgeglichen.

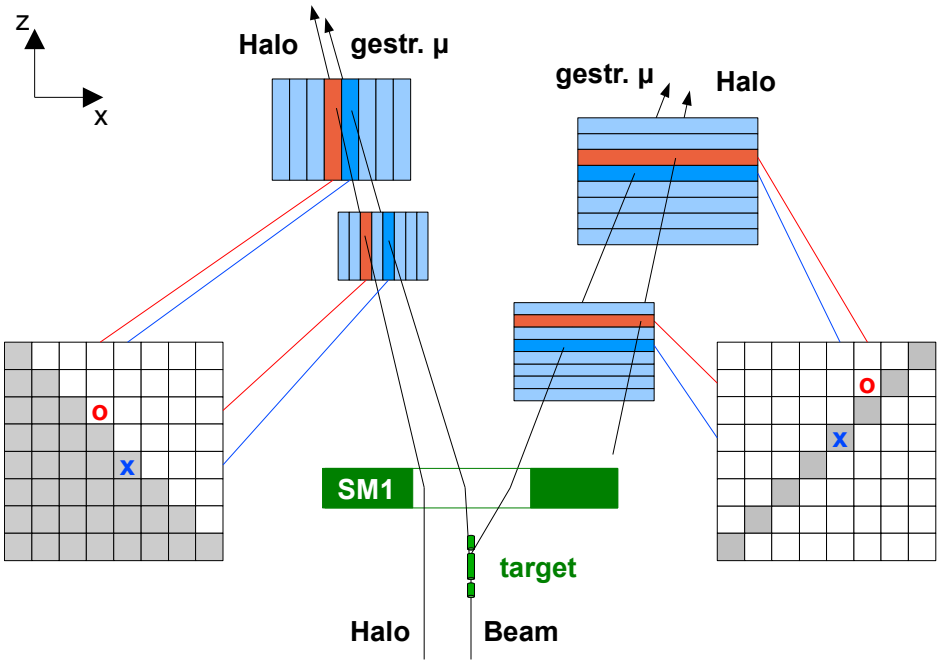


Abbildung 4.8: Vereinfachte schematische Darstellung eines Energieverlust-Triggers (links) und eines Geometrie-Triggers (rechts) zusammen mit ihren Koinzidenzmatrizen. Die grauen Felder markieren die erlaubten Kombinationen zwischen den Szintillatorstreifen, alle anderen Kombinationen werden unterdrückt. Beim Energieverlust-Trigger werden all jene Pfade unterdrückt, die nicht stark genug abgelenkt sind. Beim Geometrie-Trigger werden all jene Pfade unterdrückt, die nicht ins Target interpoliert werden können.

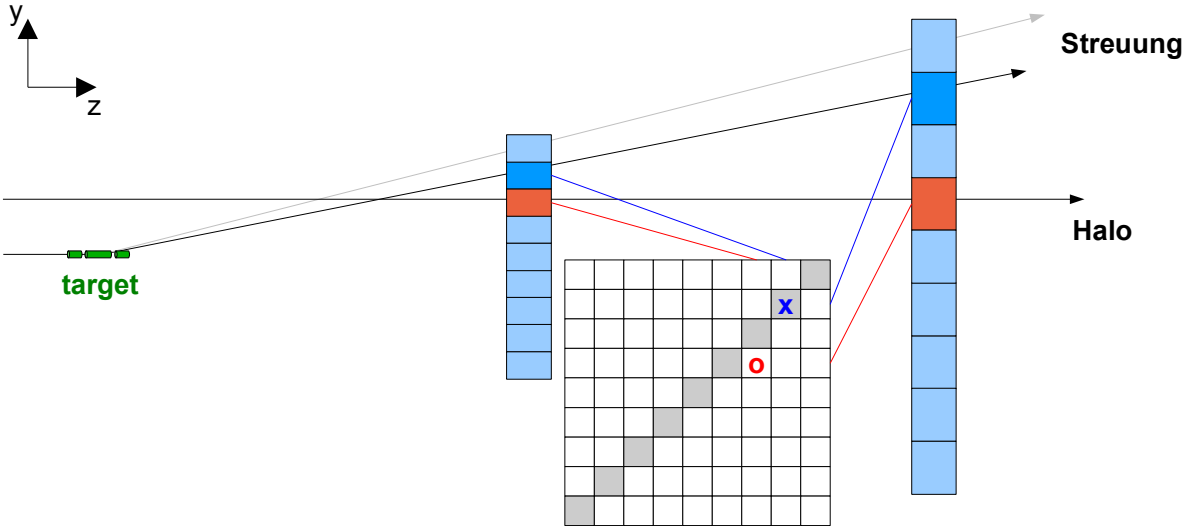


Abbildung 4.9: Seitenansicht eines Geometrie-Triggers zum besseren Verständnis seiner diagonalen Koinzidenzmatrix.

5. Einführung in die FPGA-Technologie

Die für den neuen LAS-Trigger benötigten Schaltungen werden auf einem FPGA[1] umgesetzt - einer Halbleiter-Komponente mit einer großen Anzahl von frei programmierbaren Logikbausteinen. Die Schaltungen werden dabei nicht permanent im FPGA implementiert, sondern können über eine Konfigurationsdatei geladen und auch wieder gelöscht werden. Für das COMPASS-Experiment ergibt sich der Vorteil, dass der FPGA aufgrund seiner generischen Struktur für verschiedene Anwendungen innerhalb des Experiments verwendet werden kann. Die Trigger-Schaltungen können an neue Bedingungen angepasst oder durch andere Schaltungen ersetzt werden.

Nach einer kurzen Darstellung der Entwicklungsgeschichte des FPGAs wird der Aufbau des in der vorliegenden Diplomarbeit verwendeten Virtex 5 von Xilinx beschrieben. Anschließend werden das Konzept der Hardwareprogrammierung eingeführt und der Design-Flow erläutert, mit denen die benötigten Konfigurationsdaten für den FPGA erstellt werden.

[1] Field Programmable Gate Array, regelmäßige Anordnung von programmierbaren Logikbausteinen.

5.1 Entwicklung vom Transistor bis zum FPGA

Das erste Patent, das die Funktionsweise eines Transistors beschreibt, wurde bereits 1925 vergeben. Aber erst 1947 gelang es Mitarbeitern der Bell Laboratories, einen Germanium-Transistor herzustellen. Nachdem ab 1954 die Siemens & Halske AG in der Lage war, hochreines Silizium herzustellen, konnten erstmals Halbleiter-Transistorelemente in Serie produziert werden. Anfang der 70er Jahre war die Miniaturisierung bereits so weit fortgeschritten, dass mehrere dieser Elemente gleichzeitig auf einem Halbleiter gefertigt werden konnten. Diese integrierten Schaltungen (ICs[2]) gaben den Startschuss für die heutige Mikroelektronik und es konnten mit der Zeit immer komplexere Schaltungen auf immer kleineren Räumen untergebracht werden. Bei diesen ICs ist die Struktur sowie die Funktion nach der Fertigung nicht mehr veränderbar. Aber bereits 1970 gab es die ersten Anstrengungen, Struktur und Funktion zu entkoppeln, sodass ein Halbleiter-Modul nach der Fertigung zunächst keine Funktion hatte und dann je nach Aufgabenstellung programmiert werden konnte. Die ersten programmierbaren Logikbausteine (allgemein als PLDs[3] bezeichnet) konnten nur einmal programmiert werden. Ihre Eingangssignale waren in mehreren Kombinationen über UND-Gatter und ODER-Gatter miteinander verknüpft und bei den nicht benötigten Signalwegen wurden nachträglich die Sicherungen durchgebrannt (Fuse-Technik).

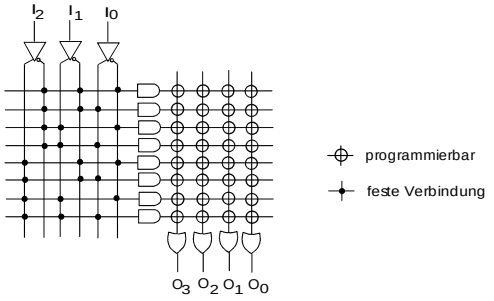


Abbildung 5.1: Grundstruktur eines PROM. Bei diesem Typ können die ODER-Gatter einmalig über die Fuse-Technik frei programmiert werden. [38]

Neben der Fuse-Technik wurde auch die sog. Antifuse-Technik entwickelt, bei der in jedem Signalweg eine sperrende Diode saß, die durch sehr hohe Ströme zerstört werden konnte und dadurch leitend wurde. Weitere PLD-Typen neben dem in Abbildung *5.1* gezeigten PROM[4] waren die PALs[5] und PLAs[6]. Die PROMs und PALs sind im Prinzip Sonderfälle des PLA, bei denen entweder nur das ODER- oder nur das UND-Gatter programmiert werden konnte (s. Abbildung *5.2*).

Die ersten mehrfach programmierbaren PLDs waren die 1983 eingeführten vPALs von AMD (variable PAL), später setzte sich aber die Bezeichnung GAL[7] der Firma Lattice Semiconductors durch. Sie nutzten EPROM[8] und später EEPROM[9]. Bei

[2] Integrated Circuits
[3] Programmable Logic Devices
[4] Programmable Read-Only Memory
[5] Programmable Array Logic
[6] Programmable Logic Array
[7] Generic Array Logic
[8] Erasable Programmable Read-Only Memory
[9] Electrically Erasable Programmable Read-Only Memory

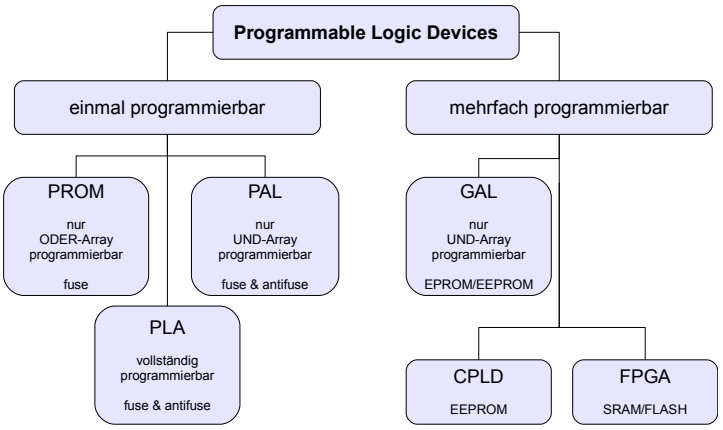


Abbildung 5.2: Übersicht über die verschiedenen PLDs

EPROM besteht eine Speicherzelle aus einem Transistor mit einem kontaktfreien sog. floating gate. Durch Anlegen eines hohen Stroms wird dieses Gate über den Tunneleffekt geladen, wodurch sich die Ansteuerspannung des Transistors ändert (d.h. er schaltet nicht mehr). Durch Bestrahlung mit UV-Licht wird das Gate wieder entladen. Bei EEPROM erfolgt das Entladen des Gates über einen Löschstrom.

Während PLAs, PALs und GALs inzwischen durch die flexibleren CPLDs[10] verdrängt wurden, sind PROMS noch immer im Einsatz, jetzt aber als mehrfachprogrammierbare EPROMS oder EEPROMS. Die CPLDs bestehen aus mehreren Blöcken ähnlich der GALs, können aber vollständig programmiert werden. Für die Verbindung von einem Eingangs-Pin zu einem Logik-Element bzw. zu einem Ausgangs-Pin gibt es immer nur genau einen Signalweg, wodurch die Signallaufzeiten innerhalb der CPLDs sehr gut vorhersagbar sind. Die CPLDs eignen sich besonders für rein kombinatorische Anwendungen und um eine große Zahl paralleler Operationen auszuführen. Für Schaltungen, die ein komplexes Routing benötigen, sind die CPLDs aber aufgrund der festen Input-Output Verbindungen ungeeignet. Außerdem gibt es für jeden Eingangspin nur einen FlipFlop, sodass für speicherintensive Anwendungen nicht genug Speicher zur Verfügung steht. Diese Lücke schließt der FPGA. Hier werden anstatt fester UND-Gatter bzw. ODER-Gatter sog. Logiktabellen (SRAM-Zellen) verwendet und in einem Array beliebig interkonnektierbarer Logikblöcke angeordnet. Dadurch sind zwar die Signallaufzeiten von einem Eingangs-Pin zu einem Ausgangs-Pin nicht mehr so gut vorhersagbar, durch das freie Routing und die deutlich höhere Anzahl von FlipFlops können nun jedoch wirklich komplexe Schaltungen realisiert werden.

Bis vor kurzem ist es noch nicht möglich gewesen, FPGAs mit nicht-flüchtigen Speicherelementen zu fertigen, sodass sie ihre Programmierung nicht dauerhaft halten können. Ihnen werden daher CPLDs zur Seite gestellt, die bei Bedarf die benötigte Konfiguration aus einem Flashspeicher auslesen und in Form eines seriellen Datenstroms an die Konfigurations-PINs des FPGAs senden. Diese mit multi-die bezeichnete Methode erlaubt die Entwicklung von SRAM-basierten FPGAs, die nach dem Einschalten direkt einsetzbar sind. Die dynamischen Inhalte der FlipFlops oder RAM-Segmente bleiben nach dem Ausschalten bei dieser Methode aber nicht erhalten. Das funktioniert erst mit den neueren FLASH-basierten FPGAs, z.B. mit den FPGAs der LatticeXP2-Familie der Lattice Semiconductor Corporation.

[10] Complex Programmable Logic Devices

5.2 Der Virtex 5 von Xilinx

Der Virtex 5 ist ein in 65nm-Halbleitertechnik gefertigter FPGA (12 layer copper-CMOS[11]). Die hier verwendete Variante (XC5VSX95T-2) besteht aus einem Array von 46×160 Logikblöcken, den sog. Configurable Logic Blocks (CLBs). Diese CLBs sind jeweils mit einer Switch-Matrix verbunden, über die eine Verbindung zu jeder anderen CLB möglich ist. Die INPUT- und OUTPUT-Pads an den Rändern sind ebenfalls über Switch-Matrizen mit dem Array verbunden (s. Abbildung *5.3*).

Innerhalb einer CLB befinden sich zwei sog. Slices. Sie besitzen keine direkte Verbindung zueinander, sondern müssen ebenfalls über die Switch-Matrix routen. Die einzige Ausnahme bildet die Direktverbindung über die Carry-Chain: Sie ist eine 1-Bit Verbindung, um Überträge z.B. bei arithmetischen Operationen besonders schnell zu übermitteln. Die Carry-Chain läuft in jeder Spalte von unten nach oben und kann in jeder Slice der Spalte abgegriffen oder verändert werden.

Eine Slice enthält vier aus SRAM bestehende Logiktabellen, sog. Look Up Tables (LUTs). Jede dieser LUTs besitzt sechs Eingänge und zwei Ausgänge, es lässt sich damit jede beliebige logische 6-zu-1 oder 5-zu-2 Operation abbilden (s. Abbildung *5.4*). Die Eingangssignale können dabei als Adresssignal für die SRAM-Zellen aufgefasst werden: Jede Eingangssignal-Kombination adressiert eine 1-Bit-Information, die dann auf den Ausgang gelegt wird.

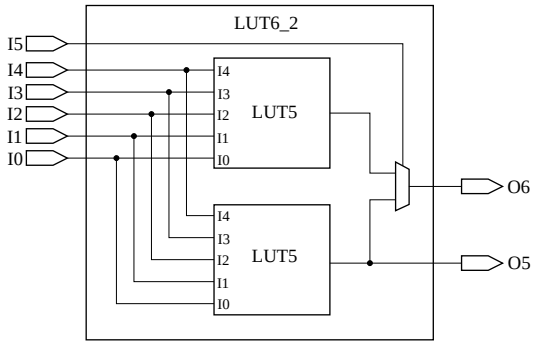


Abbildung 5.4: Die LUTs des Virtex 5 liefern auf Wunsch auf einem zweiten Ausgang (O5) den Wert der angelegten aber um ein Bit reduzierten Adresse (höchste Bit also immer 0). Die LUTs lassen sich daher als zwei LUTs mit fünf identischen Eingangssignalen auffassen (ohne Verzögerungen). An Ausgang O5 liegt immer die *untere* LUT an und an Ausgang O6 in Abhängigkeit von I5 entweder die *obere* oder auch die *untere* LUT. In dieser sog. LUT6_2 können daher auch zwei logische 5-zu-1 Operationen implementiert werden. [19]

Neben den LUTs enthält jede Slice auch vier 1-Bit-Speicher, die als FlipFlop oder Latch eingesetzt werden können. Das Routing innerhalb der Slice wird über mehrere Multiplexer bestimmt. Einige davon sind durch die Konfigurationsdatei des FPGAs festgelegt (programmiert), andere sind dynamisch, da ihre Steuersignale beschaltet werden können. Mit diesen dynamischen Multiplexern ist es z.B. möglich, die vier LUTs einer Slice zu kaskadieren und einen 16-zu-1 Multiplexer in einer Slice zu implementieren, ohne dazu die Signale aus der Slice herausführen zu müssen.

[11] Complementary Metal Oxide Semiconductor

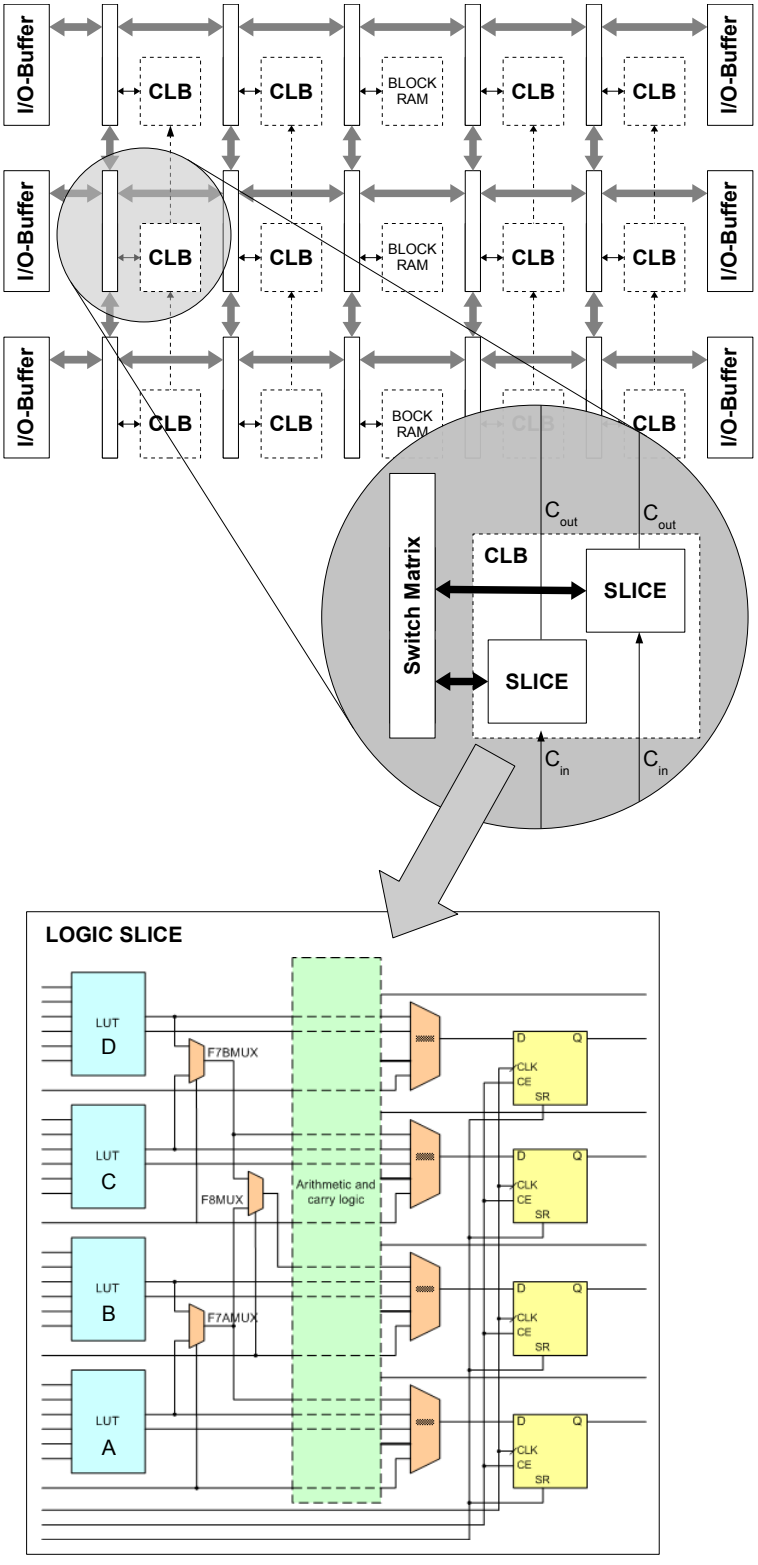


Abbildung 5.3: Schematischer Aufbau des verwendeten Virtex 5 von Xilinx (XC5VSX95T-2). Die vier LUTs einer Slice werden mit D6LUT, C6LUT etc. bezeichnet. Jede CLB besteht aus zwei Slices, meist aus einer SLICEM (optimiert für Speicher-Operation, z.B. Schieberegister) und einer SLICEL (optimiert für Logik-Operationen), teilweise aber auch aus zwei SLICELs.

Aus den 46×160 CLBs dieses Virtex 5 ergeben sich bei acht LUTs pro CLB insgesamt 58.880 LUTs und genauso viele FlipFlops. Weiterhin befindet sich eine große Anzahl dedizierter Elemente auf dem FPGA, die spezielle Aufgaben übernehmen können:

Dazu gehören:

- IODELAYs mit denen die Ein- und Ausgangssignale um bis zu 5ns [23] verzögert werden können, um interne Laufzeitunterschiede auszugleichen. Die Verzögerung kann in 64 Stufen zu je 75ps eingestellt werden.
- 36KBit Block-RAM (insgesamt 8MBit).
- Dynamic clock manager (DCM) zur Manipulation des Taktes (Phasenverschiebung, Multiplizierung etc.).
- DSP[12]-Slices, die für Algorithmen der digitalen Signalverarbeitung optimiert sind, z.B. für die schnelle Fourier-Transformation (FFT[13]).

Neben den einfachen Slices für logische Operationen (auch SLICEL genannt), gibt es einige, die für die Implementierung eines Schieberegisters optimiert sind. In diesen SLICEMs gibt es einen zusätzlichen Signalweg, der bei zwei benachbarten Registern das letzte Bit des einen mit dem ersten Bit des anderen Registers verbindet. Dieser Shift verläuft in einer Spalte von oben nach unten (im Gegensatz zu Carry-Chain).

Die Switch-Matrizen, die das Routing zwischen den CLBs ermöglichen, werden durch Transistoren geschaltet. Die Verbindungen sind damit nicht starr und können flexibel konfiguriert werden. Wie in Abbildung *5.5* zu erkennen, können auch sehr leicht Knotenpunkte erzeugt werden.

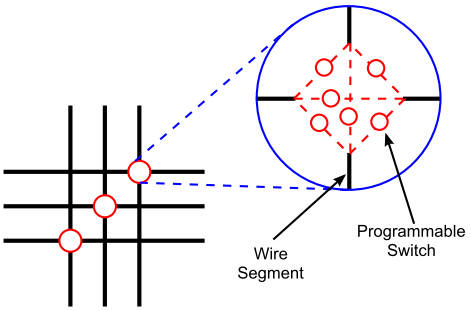


Abbildung 5.5: Prinzip der transistorgesteuerten Switch-Matrix.

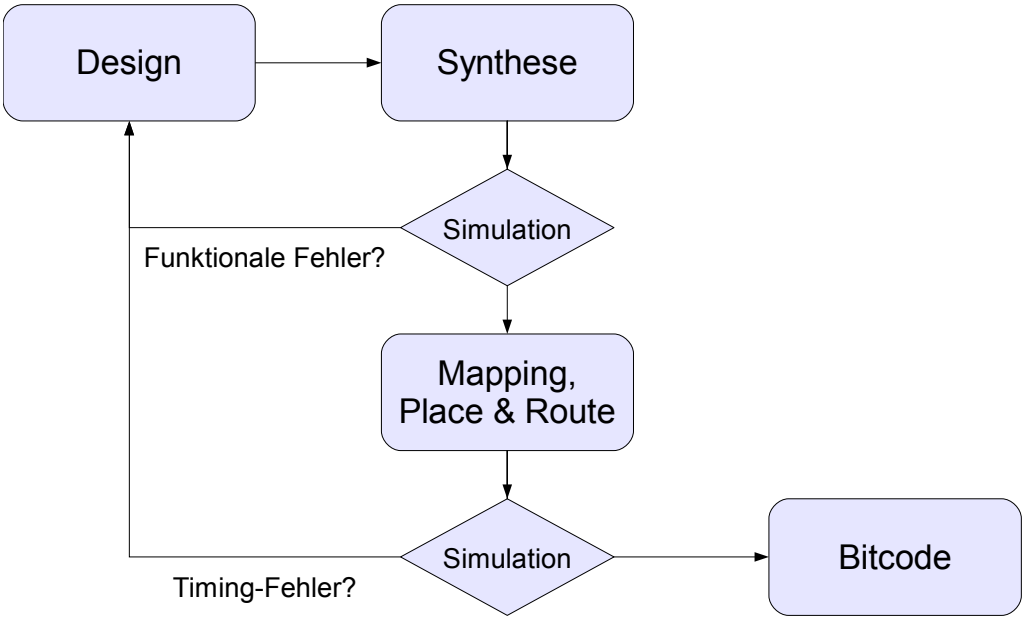
Die FPGAs der Firma Xilinx werden nach der Produktion anhand ihres Timingverhaltens in die drei Speed-Grades -3, -2 und -1 klassifiziert, wobei -3 der schnellste und -1 der langsamste ist. Der Speed-Grade wird meistens an die Typenbezeichnung angehängt. Für jeden Signalweg innerhalb des FPGAs gibt es Grenzwerte, die eingehalten werden müssen, um einen bestimmten Speed-Grade zu erreichen. Die genauen Bedingungen für die unterschiedlichen Speed-Grades können in [21] nachgelesen werden. Die unterschiedlichen Timings ergeben sich durch Prozessvariationen und durch die statistischen Schwankungen der lokalen Ladungsträgerdichte im Substrat [26]. Bei dem XC5VSX95T wurden keine FPGAs gefertigt, die die Bedingungen der dritten Stufe erfüllen, sodass der in der vorliegenden Diplomarbeit verwendete XC5VSX95T-2 die schnellste Variante dieses Typs ist.

[12] Digital Signal Processing
[13] Fast Fourier Transform

5.3 FPGA Design-Flow

Der automatisierte Design-Prozess mit Hilfe der ISE-Design Suite von Xilinx besteht aus mehreren Schritten und erzeugt die Konfigurationsdatei bzw. den Bitcode, mit dem die gewünschte Schaltung im FPGA implementiert werden kann. Durch das Laden des Bitcodes werden die SRAM-Zellen in den LUTs initialisiert, die Multiplexer in den Slices konfiguriert und das Routing in den Switch-Matrizen festgelegt.

Der Design-Flow gliedert sich in die folgenden Phasen:



Die Programmierung bzw. das Design einer Schaltung erfolgt über eine Hardwarebeschreibungssprache (HDL). Ein großer Unterschied zur herkömmlichen Softwareprogrammierung besteht darin, dass Schaltungen parallel ablaufen können, während herkömmlicher Programmcode immer sequentiell ausgeführt wird. In dieser Arbeit wurde mit der Hardwarebeschreibungssprache Verilog gearbeitet, die stark an die Syntax von ANSI C angelehnt ist. Die zu entwickelnde Schaltung wird zunächst über ihr Verhalten beschrieben (behavioral modelling), das erfolgt analog zur Softwareentwicklung über das Zuweisen und Abfragen von Variablen (s. Listing *5.1*). Diese Variablen entsprechen bei der Hardwareentwicklung Signalen (wire) und Speicherwerten (reg).

Bei der Synthetisierung wird aus dem Quellcode zunächst ein Schaltplan mit den notwendigen logischen Elementen, eine sog. Netlist erzeugt. Nach der Synthese kann mit dem Design-Tool zur Überprüfung der Schaltung eine Funktionssimulation durchgeführt werden. Die Verzögerungen durch die Signallaufzeiten werden hier jedoch noch nicht berücksichtigt. Alternativ zur Verhaltensbeschreibung kann der Entwickler die Schaltung auch direkt über die logischen Gatter beschreiben (gate-level-modeling). Bei der Gatterbeschreibung werden Module miteinander verknüpft, die den logischen Gattern entsprechen.

Entspricht die Funktionssimulation den Erwartungen, erfolgt mit dem Mapping eine Anpassung der Netlist an die spezielle FPGA-Architektur. Es wird z.B. geprüft, ob durch die Verwendung der LUT6_2 der Platzverbrauch der Schaltung reduziert werden kann. Der Entwickler hat wiederum die Möglichkeit, auch direkt auf diesem

FPGA-spezifischen Level zu designen. Er kann manuell FPGA spezifische Elemente instanziieren und die tatsächlichen Pins konnektieren (s. Listing *5.1*).

Im Anschluss erfolgt anhand der durch die Speed-Grade-Klassifizierung bekannten maximalen Laufzeiten das Placing und das Routing. Wird auf dem FPGA z.B. ein Schieberegister programmiert, das die Informationen der FlipFlops bei jedem Takt weiterschiebt, dann müssen die Signale innerhalb eines Takts den nächsten FlipFlop erreichen. Werden die sog. Setup- und Hold-Zeiten nicht eingehalten, kommt es zu einer Funktionsstörung und einem undefinierten Zustand des Systems - ähnlich den Systemabstürzen, wenn ein PC mit übertakteter CPU betrieben wird. Die Design-Software von Xilinx platziert die Elemente der Schaltung so, dass die Timing-Vorgaben eingehalten werden. Ist das Routing nicht möglich, wird eine entsprechende Fehlermeldung ausgegeben.

Im Unterschied zu CPLDs gibt es auf einem FPGA für ein festes Placing mehrere Routings, sodass die Design-Software hier nochmals alle Pfade analysieren und zeitkritische Pfade bevorzugen kann. Man spricht oft nur noch von *Place & Route*, da sich beide Prozesse gegenseitig beeinflussen.

Über sog. Constraints kann der Entwickler den automatisierten Place & Route-Prozess mit zusätzlichen Beschränkungen beeinflussen. Es ist möglich, für einen Signalweg eine maximale Laufzeit vorzugeben, die beim Place & Route-Prozess eingehalten werden muss. Zusätzlich können über die LOC[14]- und die BEL[15]-Constraints die Positionen von LUTs und FlipFlops genau vorgegeben und mit den ROUTE-Constraints bestimmte Signalpfade erzwungen werden. Die LOC-Positionen werden dabei über ein Koordinatensystem der Slices mit Ursprung in der unteren linken Ecke des FPGAs festgelegt. Da jede CLB zwei Slices enthält und das Koordinatensystem jeweils bei 0 anfängt, wird die Slice in der oberen rechten Ecke über die Koordinate SLICE_X91Y159 angesprochen. Die Route zwischen zwei Punkten wird über eine komplexe relative Pfadbeschreibung festgelegt, deren exakte Bedeutung sich dem Autor entzieht. Die Routing-Angaben können daher nicht manuell erzeugt sondern nur mit Hilfe des FPGA-Editors der ISE-Design-Suite aus einem gerouteten Design extrahiert werden.

In Listing *5.2* sind Beispiele für diese Constraints gegeben. Die komplette Liste der möglichen Constraints kann in [20] eingesehen werden.

[14] Location Constraints, eine Constraints-Definition, um ein logisches Element in einer bestimmten Slice zu platzieren.

[15] Basic Element of Logic, eine Constraints-Definition, um ein logisches Element innerhalb einer Slice zu positionieren.

Listing 5.1: Beschreibung einer UND-Schaltung über ihr Verhalten, die benutzten Gatter und die benutzten FPGA-Elemente (Verilog Code).

```
1  module UND_Verhalten (out, a, b);
2    output out; input a, b;
3    assign out = a & b;
4  endmodule
5
6
7  module UND_Gatter (out, a, b);
8    output out; input a, b;
9    and G2 (out, a, b);
10 endmodule
11
12
13 module UND_FPGA_Element (out, a, b);
14
15   //Wire Definition mit zusätzlicher Direktive, um den wire
16   //zu erzwingen, er darf während der Optimierung nicht aus
17   //dem Design entfert werden.
18   (* S="TRUE" *) output out;
19   (* S="TRUE" *) input a, b;
20
21   //Hier wird zunächst eine LUT mit 5 aktiven Eingangssignalen
22   //instanziiert. Die logische Operation wird durch die INIT-VALUE
23   //der SRAM-Zelle definiert. In diesem Beispiel ist die
24   //Operation (I0 & I1) unabhängig von den Signalen an den Eingängen
25   //I2, I3 und I4, weshalb sich nicht konnektiert werden müssen.
26
27   //Die Lock-Direktive zwingt die Verwendung der angegeben
28   //PINs. Ohne diese Angabe kann während der Optimierung
29   //die Pinbelegung geändert werden (dabei wird auch die
30   //INIT-VALUE entsprechend angepasst).
31   (* LOCK PINS="ALL" *)
32   LUT5 #(.INIT(32'h88888888)) LUT5_UND (
33     .O(out),
34     .I0(a),
35     .I1(b));
36 endmodule
```

Listing 5.2: Beispiele für Constraints der Elemente aus Listing *5.1*.

```
1  //Für das out-Signal des Moduls UND_Verhalten wird
2  //eine Maximale Laufzeit definiert.
3  NET "UND_VERHALTEN/out" MAXDELAY = 10ns;
4
5  //Hier werden alle Elemente des Moduls UND_GATTER
6  //in einem bestimmten Bereich des FPGAs platziert.
7  AREA_GROUP "UND_GATTER/*" RANGE = SLICE_X2Y25:SLICE_X3Y33;
8
9
10 //Hier wird die manuell instanziierte LUT_UND aus
11 //Listing 5.1 auch manuell platziert.
12 //Über die BEL-Angabe (Basic Element of Logic) wird
13 //auch die Position in der Slice festgelegt.
14 INST "LUT5_UND" LOC = SLICE_X1Y123;
15 INST "LUT5_UND" BEL = D6LUT;
16
17
18 //Wenn zwei Elemente fest platziert sind, dann kann auch
19 //die Route zwischen beiden vorgeschrieben werden.
20 NET "UND_FPGA_Element/out"
21 ROUTE ="{3;1;5vsx95tff1136;8b018a51!-1;-3784;-23752;S!0;-843;-824!1;-1738;"
22 "3944!2;1738;-256!3;843;296;L!}";
```

6. Realisierung der 64 Meantimer und der Koinzidenzschaltung auf einem FPGA

Das Ziel dieser Diplomarbeit ist die Entwicklung von 64 Meantimern, deren Ausgangssignale über eine 32×32 Koinzidenzschaltung miteinander verglichen werden. Je nach Prüfergebnis soll dann ein Trigger-Signal erzeugt werden. Weiterhin sollen die Meantimer einen dynamischen Bereich von ca. 30ns besitzen. Darüber hinaus soll das gesamte System unter Verwendung eines FPGAs umgesetzt werden.

Auf einem FPGA bietet sich die Umsetzung des in Abbildung *6.1* dargestellten TDL-Meantimers an. Werden die beiden gegenläufigen TDLs dieses Meantimer-Typs durch eine getaktete Schaltung realisiert, dann werden pro Meantimer zwei Schieberegister benutzt, welche die Eingangssignale mit jedem Takt um ein Delay-Element weiterschieben. Für das Auflösungsvermögen ist dann nur der maximal mögliche Takt entscheidend. Der verwendete Virtex 5 (XC5VSX95T-2) von Xilinx kann bei maximal 500MHz betrieben werden, wodurch die Signale alle 2ns um ein Delay-Element weitergeschoben werden können. Dies entspricht nicht den Anforderungen des COMPASS-Experiments, da auf diese Weise nur eine Auflösung von ca. 1ns erreicht werden kann. Um eine höhere Auflösung zu erreichen, wird in der vorliegenden Diplomarbeit eine ungetaktete Schaltung entwickelt. Die Delay-Elemente der TDLs werden bei dieser Methode allein durch die Signallaufzeiten innerhalb des FPGAs erzeugt.

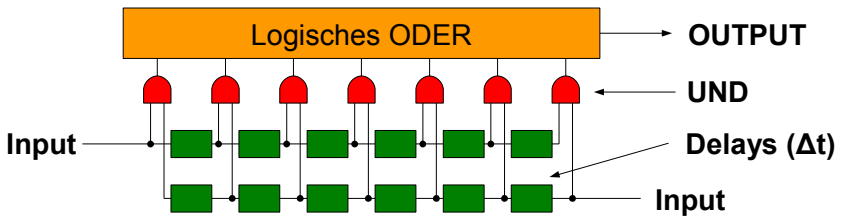


Abbildung 6.1: Die Delay-Elemente eines TDL-Meantimers müssen alle die gleiche Verzögerung Δt besitzen, damit der Meantimer linear arbeitet.

Die Herausforderung dieser Arbeit besteht in der Entwicklung der beiden gegenläufigen und identischen TDLs mit konstanten Delay-Elementen und der zeitstabilen Verbindung der einzelnen UND-Gatter-Ausgänge zu einem großen ODER-Segment. Dazu werden zunächst einige Untersuchungen durchgeführt, um das allgemeine Verhalten einer ungetakteten Schaltung auf dem FPGA und die Kontrollierbarkeit der Signallaufzeiten durch die Xilinx-Design-Software zu bestimmen. Mit den Erkenntnissen aus diesen Untersuchungen wird dann das Konzept für einen TDL-Meantimer entworfen. Danach wird ein Verfahren entwickelt, das die Pfade mit den gesuchten Laufzeiteigenschaften für das ODER-Segment findet. Anschließend wird der Prototyp im Labor mit Testsignalen vermessen und das Testergebnis diskutiert.

Die darauf folgenden Abschnitte beschreiben die Platzierung der 64 parallelen Meantimer und die Entwicklung der Koinzidenzschaltung. Außerdem wird das Web-Interface erläutert, mit dem über eine VME-Schnittstelle die IODELAYs der Meantimer und die Koinzidenzmatrix konfiguriert werden können. Zum Schluss werden einige Tests durchgeführt, um die Funktion der entwickelten Komponenten zu überprüfen.

6.1 Entwicklung des Prototypen

6.1.1 Voruntersuchungen

Während der Synthetisierungsphase wird das Design durch die Xilinx-Software optimiert. Da diese Optimierungen für getaktete Schaltungen ausgelegt sind, können sie sich bei ungetakteten Schaltungen störend auswirken: Logische Elemente können durch die Optimierungen unter Umständen zusammengefasst oder entfernt werden, wodurch sich das Laufzeitverhalten der Schaltung ändert.

Mit einer speziellen Direktive können einzelne Verbindungen (wires) geschützt werden (s. Listing *6.1*), wodurch auch die konnektierten logischen Elemente erhalten bleiben. Der Aufwand für das Einfügen dieser Direktiven ist so gering, dass im Folgenden fast alle Signalwege auf diese Weise geschützt werden und so Fehler durch die Optimierung von Beginn an ausgeschlossen sind.

Listing 6.1: Codesegment mit zwei speziellen Direktiven, um Optimierungen während der Synthetisierung zu unterbinden.

```
1  module leftdelay(In,Out,aOut);
2
3          // Die S-Direktive schützt den wire während der Optimierung
4          (* S="TRUE" *) input wire In;
5          (* S="TRUE" *) output wire Out;
6          (* S="TRUE" *) output wire aOut;
7
8          // Die LOCK_PINS-Direktive verhindert die Pinvertauschung
9          (* LOCK_PINS="ALL" *)
10         LUT6_2 #(.INIT(64'hF0F0_F0F0_F0F0_F0F0)) IDelay_LUT (
11                 .O5(aOut),
12                 .O6(Out),
13                 .I5(1'b1),
14                 .I2(In)
15         );
16
17 endmodule
```

Wenn eine LUT von Hand instanziiert und sowohl die INIT-VALUE[1] als auch die Pinbelegung manuell vorgenommen wird, können durch die Optimierungen dennoch die Anschlusspins vertauscht werden (die INIT-VALUE wird dabei entsprechend angepasst). Da sich für eine Verbindung von einem festen Punkt zu allen sechs Input-Pins einer LUT unterschiedliche Laufzeiten ergeben (s. Abbildung *6.2*), stört auch diese Optimierung das Laufzeitverhalten einer ungetakteten Schaltung. Über eine weitere Direktive kann diese Optimierung ebenfalls unterbunden werden (s. Listing *6.1*).

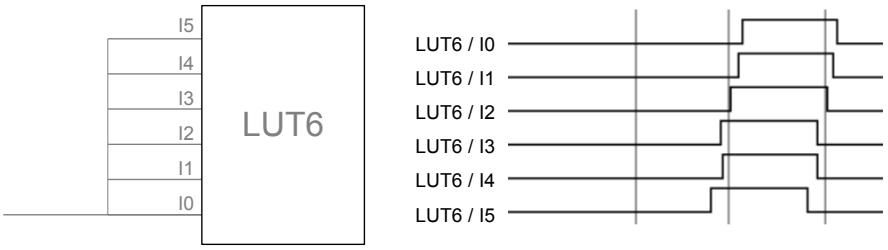


Abbildung 6.2: Die Timing-Simulation einer LUT6, bei der alle sechs Input-Pins mit dem selben Signal beschaltet sind, zeigt zwischen I0 und I5 eine Abweichung von ca. 700ps.

Mit Hilfe der MAXDELAY-Constraints können die maximalen Laufzeiten für einzelne Signalpfade definiert werden. Es gibt aber keine Constraints, um die minimalen Laufzeiten festzulegen[2]. Daher ist es nicht so einfach möglich, für mehrere Pfade identische Laufzeiten zu erzwingen. Werden z.B. für fünf Signalpfade die gleichen oberen Grenzen festgelegt, dann liegen sie im Anschluss zwar alle unterhalb dieser Grenze, weichen aber stark voneinander ab. Wird eine sehr geringe Grenze gewählt, ist es der Xilinx-Software nicht mehr möglich, diese Bedingung für alle Signalpfade zu erfüllen. Die MAXSKEW-Constraint kann ebenfalls nicht eingesetzt werden. Diese erlaubt zwar die Begrenzung der Differenz von Signalen mit einem gemeinsamen Knotenpunkt, nicht jedoch von unabhängigen Signalen. Die Signalpfade des FPGAs müssen daher einzeln analysiert werden, um die benötigten identischen Signallaufzeiten zu finden. Um den Prototyp später unkompliziert in 64facher Ausführung parallel auf dem FPGA platzieren zu können, ohne dass sich die Meantimer gegenseitig beeinflussen, sollten die TDLs horizontal in einer Zeile oder vertikal in einer Spalte verlaufen. Aus der CLB-Struktur des FPGAs ergeben sich abwechselnde SLICEM- und SLICEL-Spalten, weshalb zunächst eine spaltenweise Implementierung angestrebt wird.

Für die Laufzeitanalyse werden zwei LUTs über die LOCATION-Constraints fest auf dem FPGA positioniert und danach die Timings für die Pfade zwischen den beiden LUTs mit Hilfe der Xilinx-Software bestimmt. Eine erste grobe Analyse der Laufzeiten zwischen zwei LUTs in einer Slice und zwischen zwei LUTs in unterschiedlichen Slices ergibt:

- Die Laufzeiten zwischen zwei LUTs der selben Slice sind deutlich kürzer als die Laufzeiten zwischen zwei LUTs verschiedener Slices. Für die Umsetzung einer regelmäßigen und aus mehr als vier Sprüngen bestehenden TDL kommen daher nur Inter-Slice-Verbindungen in Frage. Andernfalls erfolgt nach vier

[1] Speicherinhalt der LUT nach der Initialisierung, definiert die logische Operation.
[2] Bei Altera-FPGAs hingegen gibt es eine solche set_min_delay-Constraint.

kleinen Sprüngen (eine Slice enthält vier LUTs) ein großer Sprung, wodurch die Linearität des Meantimers nicht mehr gewährleistet ist.

- Die Struktur des FPGAs weist jeweils alle 20 CLB-Zeilen eine kleine Lücke auf (s. Abbildung *6.3*), die eine zusätzliche Laufzeit von 3ps verursacht. Diese minimale Störung der Linearität der TDLs kann vernachlässigt werden.
- Wird eine Konfiguration auf dem FPGA verschoben (Veränderung der absoluten LOCATION-Constraints unter Beibehaltung der relativen Positionen der LUTs), treten teilweise andere Laufzeiten auf. Die Switch-Matrizen sind demnach nicht identisch!

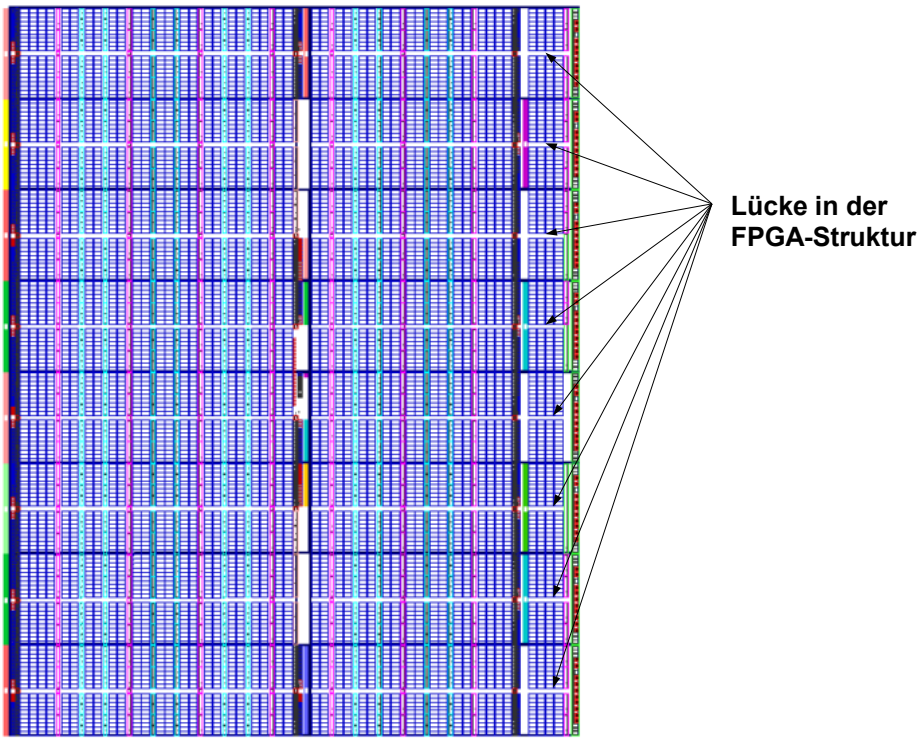


Abbildung 6.3: Diese Darstellung des FPGAs ist mit dem Tool PlanAhead der ISE-Design-Suite erstellt worden. Der Timing-Simulation nach verursachen die markierten Lücken eine zusätzliche Laufzeit von 3ps.

6.1.2 Entwicklung der gegenläufigen TDLs

Aus diesen Ergebnissen folgt, dass eine Realisierung von 64 identischen Meantimern nur dann möglich ist, wenn für die TDLs der Meantimer eine Konfiguration gefunden werden kann, die in genügend vielen Slices die gleiche Laufzeit besitzt.

Wie in Abbildung *5.3* zu erkennen ist, enthält jede CLB zwei Slices. Die rechte Slice ist immer eine SLICEL, die linke Slice ist entweder eine SLICEM oder eine SLICEL. Da die Switch-Matritzen von CLB zu CLB variieren, werden in den 46 rechten SLICELs mehr identische Lösungen vermutet, als in den 10 linken SLICELs und den 36 SLICEMs. Die genauere Analyse der Laufzeiten zwischen zwei übereinander angeordneten CLBs wird daher zwischen zwei SLICELs durchgeführt. Die Ergebnisse dieser Analyse sind in Tabelle *6.1* aufgeführt, das Analyse-Setup ist in Abbildung *6.4* dargestellt. Die Untersuchungen werden in der 6. Slice-Spalte bzw. in der 3. CLB-Spalte (rechte SLICEL) durchgeführt.

Tabelle 6.1: Analyse der Signallaufzeiten zwischen zwei LUTs in den rechten SLICELs zweier übereinander angeordneter CLBs (s. auch Abbildung *6.4*). Die Analyse erfolgt für jeweils alle möglichen Kombinationen in beide Richtungen.

	Pin: I0		Pin: I3	
	up	**down**	**up**	**down**
D6LUT	751 ps	836 ps	545 ps	430 ps
C6LUT	834 ps	828 ps	423 ps	*417 ps*
B6LUT	818 ps	897 ps	419 ps	611 ps
A6LUT	837 ps	717 ps	*418 ps*	463 ps

	Pin: I1		Pin: I4	
	up	**down**	**up**	**down**
D6LUT	756 ps	769 ps	320 ps	472 ps
C6LUT	761 ps	791 ps	*303 ps*	495 ps
B6LUT	766 ps	*739 ps*	309 ps	*305 ps*
A6LUT	*739 ps*	797 ps	312 ps	435 ps

	Pin: I2		Pin: I5	
	up	**down**	**up**	**down**
D6LUT	627 ps	*493 ps*	233 ps	278 ps
C6LUT	637 ps	488 ps	*252 ps*	282 ps
B6LUT	491 ps	522 ps	398 ps	*247 ps*
A6LUT	*493 ps*	641 ps	386 ps	245 ps

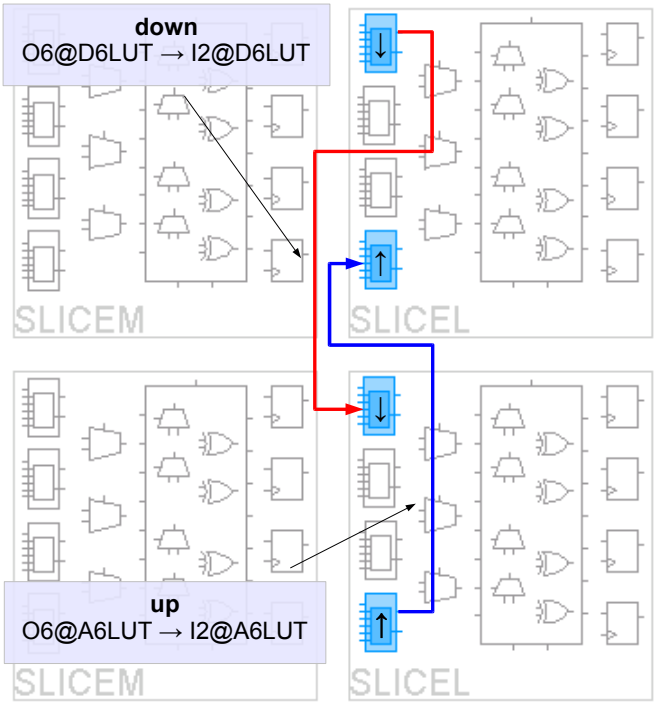


Abbildung 6.4: Analyse-Setup für zwei Pfade zwischen zwei verschiedenen LUTs. Die dargestellten Pfade besitzen beide eine Laufzeit von 493ps (s. Tabelle *6.1*) und eignen sich daher für die gegenläufigen TDLs der Meantimer.

Alle kursiv gedruckten Paare in Tabelle *6.1* werden als Kandidaten für die gegenläufigen TDLs der Meantimer betrachtet, da sie entweder gleich sind oder nur eine sehr geringe Differenz aufweisen. Eine Überprüfung dieser Verbindungen in den übrigen Slices des FPGAs ergibt, dass nur die in Abbildung *6.4* gezeigte Pin:I2-Kombination in allen 46 rechten SLICEL-Spalten die gleiche Laufzeit besitzt.

Es sind damit für den Meantimer zwei gegenläufige TDLs mit konstantem Δt gefunden. Der Wert für $\Delta t = 579$ps ergibt sich aus der Zeit, die das Signal für die Delay-LUT (86ps) und für den gefundenen Pfad (493ps) benötigt. Eine Slice, die ein Segment der beiden gegenläufigen TDLs bildet, wird im Folgenden als Step-Slice bezeichnet. Da auf dem FPGA nur 46 Spalten zur Verfügung stehen, in denen die Meantimer implementiert werden können, müssen pro Spalte mindestens zwei der 64 benötigten Meantimer konstruiert werden. Um einen dynamischen Bereich von ca. 30ns zu erreichen, müssen die TDLs aus mindestens 52 Step-Slices bestehen. Da pro Spalte 160 Slices zur Verfügung stehen, ist die geforderte Länge realisierbar.

Zur Überprüfung des erwarteten konstanten Δt werden die TDLs in der geforderten Länge erstellt und eine Timing-Simulation durchgeführt. Die Länge der TDLs wird dabei auf 54 Step-Slices festgelegt, wodurch sich ein dynamischer Bereich von mindestens 30ns ergibt. Die Simulation der TDLs bestätigt die Erwartungen und ist in Abbildung *6.5* abgebildet.

Da das Projekt unter großem zeitlichen Druck steht, wird an dieser Stelle auf eine Analyse der horizontal ausgerichteten TDLs verzichtet und der Prototyp mit den bereits gefundenen vertikalen TDLs weiterentwickelt.

6.1.3 Platzierung der UND-Gatter

Als Nächstes wird mit Hilfe einer weiteren Laufzeitanalyse nach möglichen Platzierungen für die UND-LUTs gesucht. Um sicherzustellen, dass sich die TDL-Laufzeiten nicht ändern, werden die beiden Signale zur Bildung der UND-Verknüpfungen am zweiten Ausgang der Delay-LUTs - an dem sog. O5-Pin - abgegriffen. Die beiden 493ps-Pfade der TDLs werden bei den weiteren Untersuchungen über ROUTE-Constraints erzwungen. Die beiden abgegriffenen Signale werden als lAnd[3] und als rAnd bezeichnet. Die INIT-VALUEs der Delay-LUTs werden so konfiguriert, dass ein an I2 anliegendes Signal stets an O5 und an O6 weitergereicht wird (Signalduplizierung).

Es liegt nahe, die UND-Verknüpfung von lAnd und rAnd direkt in der Step-Slice durchzuführen - entweder in der B6LUT oder in der C6LUT. Um die Linearität der TDLs durch die UND-LUTs nicht zu zerstören, müssen jeweils alle lAnd-Signale und alle rAnd-Signale eines Meantimers identische Laufzeiten besitzen. Eine eventuelle Differenz zwischen lAnd und rAnd kann über die IODELAYs der beiden Eingangssignale eines Meantimers korrigiert werden.

Die Laufzeitanalyse ergibt, dass diese Bedingung weder für die B6LUT noch für die C6LUT in allen 46 SLICEL-Spalten erfüllt werden kann. Die PIN-BEL-Kombination, bei der möglichst viele Spalten diese Bedingung erfüllen, ist in Abbildung *6.6* gezeigt: Wird die UND-Verknüpfung in der B6LUT durchgeführt und das lAnd-Signal auf den I1-Pin und das rAnd-Signal auf den I4-Pin gelegt, ergeben sich die 33 Lösungen, die in Tabelle *6.2* angegeben sind.

[3] Im weiteren Verlauf der Arbeit wird das Signal durch die DOWN-TDL als das rechte Eingangssignal und das Signal durch die UP-TDL als das linke Eingangssignal bezeichnet.

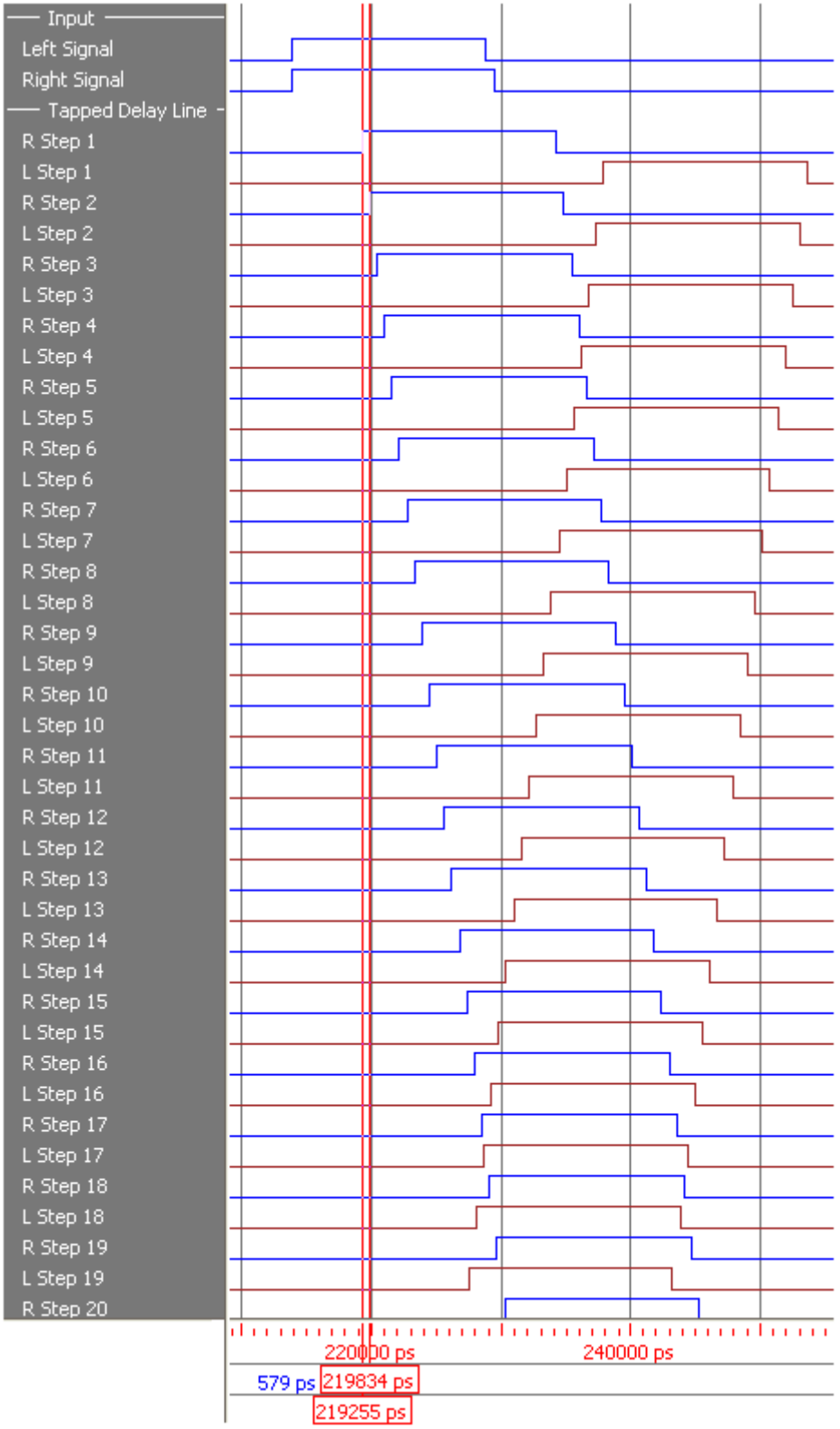


Abbildung 6.5: Timing-Simulation der TDLs nach dem Place & Route. Die beiden gegenläufigen Signale laufen wie erwartet gleichmäßig durch die Tapped Delay Lines.

Tabelle 6.2: Übersicht der Laufzeiten für lAnd und rAnd. Bei den Spalten ohne eine lAnd-Angabe variiert die Laufzeit innerhalb der Spalte, sodass dort keine linearen Meantimer konstruiert werden können.

Slice-Spalten	lAnd [ps]	rAnd [ps]
3,7,11,23,35,47,51,63,75,79,83,87,91	-	185
49	614	185
85	616	185
81,89	622	185
5	624	185
15,27,39	632	185
61,73	633	185
13,17,19,25,29,31,37,41,43,57,59,69,71	634	185
1,55,67	660	185
9,21,33,45,	662	185
53,65,77	663	185

Es können also weiterhin mindestens 64 lineare Meantimer in den verbleibenden 33 Spalten des FPGAs konstruiert werden. Die genaue Implementierung der TDLs und der UND-Gatter ist in Abbildung *6.6* dargestellt. Der Quellcode einer solchen Step-Slice ist im Anhang in Listing *A.2* aufgeführt.

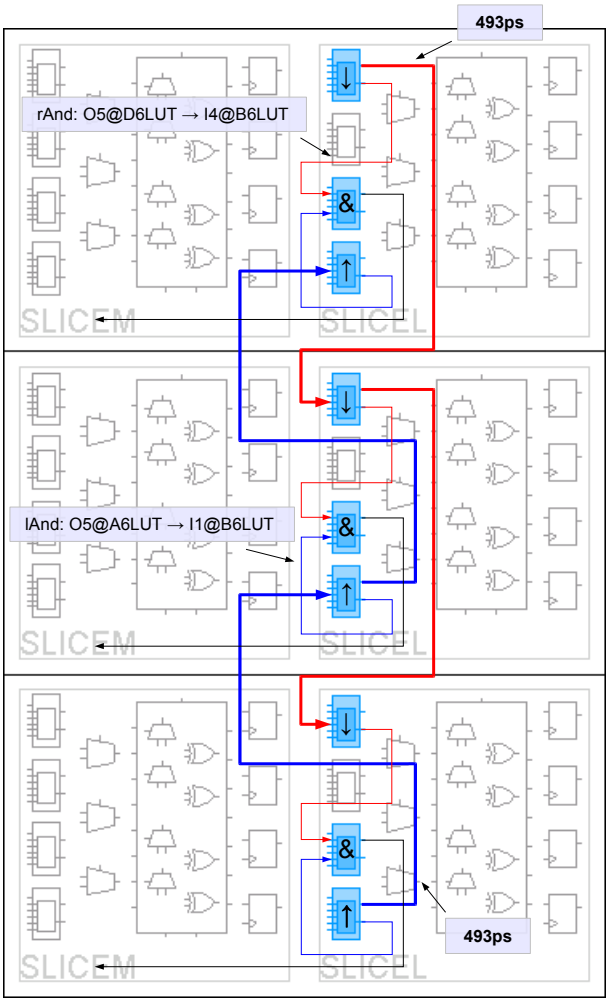


Abbildung 6.6: Endgültige Platzierung und Pinbelegung der gegenläufigen TDLs mit zwei Delay-LUTs und einer UND-LUT pro Step-Slice. Es werden nur die rechten Slices der CLBs benutzt, die linken sind noch komplett frei.

6.1.4 Entwicklung des ODER-Segments

Im letzten Schritt der Entwicklung des Prototypen müssen alle 54 UND-Signale der Step-Slices in einem großen ODER zusammengeführt werden (s. Abbildung *6.1*). Außerdem müssen insgesamt 64 Meantimer parallel auf dem FPGA platziert werden können, ohne dass sie sich gegenseitig stören. Deshalb wird versucht, das gesamte ODER-Segment in den noch freien linken Slices der CLBs zu platzieren, sodass ein Meantimer komplett in einer CLB-Spalte untergebracht werden kann.

Da die LUTs des Virtex 5 nur sechs Input-Pins besitzen, wird eine ODER-Kaskade mit mindestens drei Ebenen benötigt. Die Pfade einer Ebene müssen alle die gleiche Signallaufzeit besitzen, damit die Linearität des Meantimers erhalten bleibt (s. Abbildung *6.7*). Als Nächstes wird mit Hilfe einer Laufzeitanalyse nach Positionen für die 6-zu-1 ODER-LUTs der 1. Ebene gesucht, die diese Bedingung erfüllen.

Für die Positionierung dieser ODER-LUTs gibt es pro Slice vier Möglichkeiten und für jede dieser Positionen gibt es 6! mögliche Pinbelegungen, also 2880 Kombinationen. Die Laufzeitanalyse wird daher mit Hilfe eines kleinen Visual Basic Scripts automatisiert. Es variiert für die ODER-LUT die Pinbelegung (PIN-Permutation), die Position in der Slice (BEL-Permutation) und auf Wunsch auch die Position der Slice selbst (LOC-Permutation). Das Script erzeugt den nötigen Verilog-Quellcode und führt über die Xilinx Tcl[4]-Shell ein Place & Route durch. Aus der dabei erstellten Log-Datei werden die Laufzeitinformationen extrahiert und die Differenz der Signale berechnet. Diese Differenzinformation wird zusammen mit der PIN-BEL-LOC-Konfiguration in einer separaten Log-Datei gespeichert, sodass diese Informationen später analysiert werden können. Sollten die Laufzeiten aller Signale identisch sein, werden für alle gefundenen Signalpfade die ROUTE-Constraints gespeichert. Diese automatisierte Laufzeitanalyse wird nicht nur mit einer einzigen, sondern parallel mit allen ODER-LUTs der ersten Ebene durchgeführt.

Wie sich herausstellt, ergibt diese Analyse jedoch keine brauchbaren Lösungen, da die Laufzeiten im besten Fall um mehr als 600ps abweichen. Um die Wahrscheinlichkeit für identische Laufzeiten zu maximieren, wird die Konstruktion der ODER-Kaskade verändert, sodass in jeder Ebene nur noch zwei Signale miteinander verodert werden. Es ergibt sich folglich eine ODER-Kaskade mit sechs Ebenen. Die Analyse der modifizierten Schaltung liefert zwar ebenfalls keine Lösungen mit identischen Laufzeiten, jedoch gibt es eine Kombination, deren Signalpfade nur um 2ps voneinander abweichen. Die automatisierte Laufzeitanalyse wird nun auch für den Rest der ODER-Kaskade durchgeführt. Die dadurch gefundenen Pfade weisen ebenfalls nur sehr geringe Abweichungen auf und sind in Tabelle *6.3* angegeben. Die gesamte Abweichung über alle Ebenen beträgt maximal 34ps, die Störung der Linearität durch diese Abweichungen kann vernachlässigt werden.

Eine Simulation des Prototypen nach dem Place & Route und eine schematische Darstellung seiner Platzierung auf dem FPGA sind in Abbildung *6.8* und *6.9* zu sehen. Die Ergebnisse der Simulation sind in Tabelle *6.4* aufgeführt.

[4] Tool command language

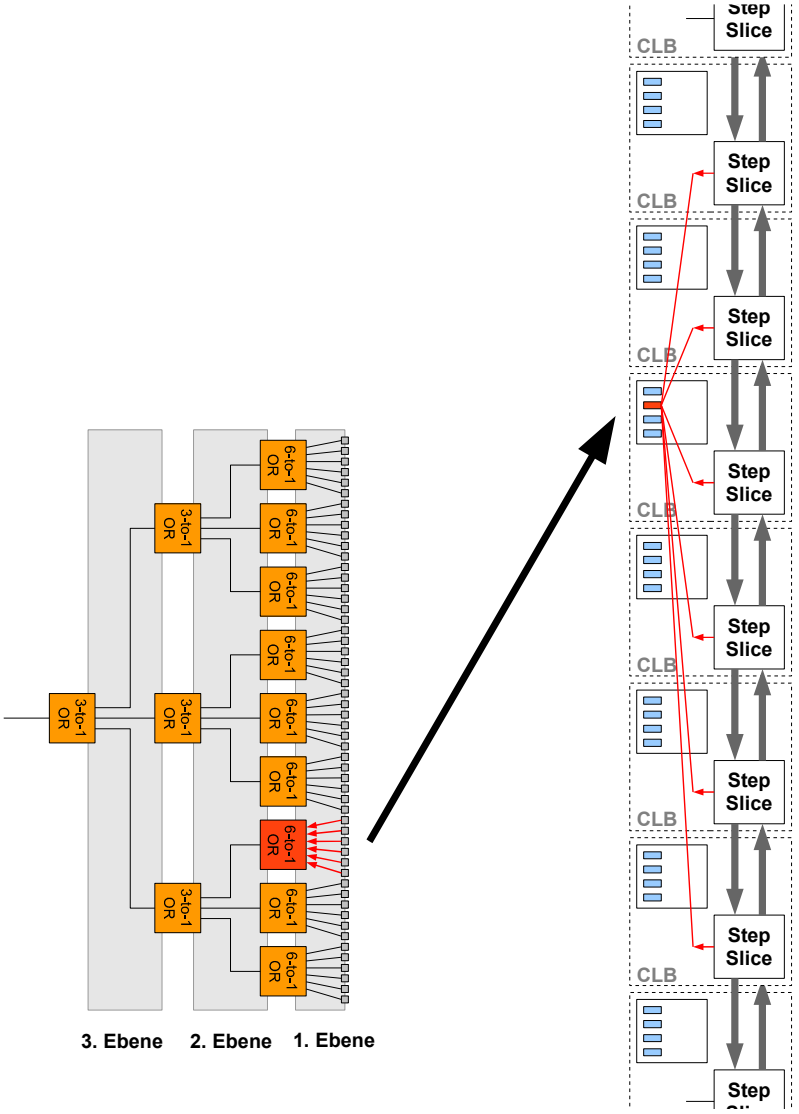


Abbildung 6.7: Schematische Darstellung der ersten Version des ODER-Segments, das alle 54 UND-Signale der Step-Slices zusammenführt. Es ist als 3-stufige ODER-Kaskade ausgeführt. Damit die Linearität des Meantimers nicht gestört wird, müssen alle Signale einer Ebene identische Laufzeiten aufweisen. Um diese Bedingung zu erfüllen, können sowohl die Pinbelegung der ODER-LUTs, ihre Position innerhalb der Slice als auch die Position der Slice selbst variiert werden. Wie sich später zeigt, kann für diese Variante keine Kombination gefunden werden, bei der die Laufzeiten identisch sind. Im endgültigen Design wird daher eine 6-stufige ODER-Kaskade benutzt, die nur aus 2-zu-1 ODER-LUTs besteht.

Der Jitter des Ausgangssignals wird durch das Auflösungsvermögen

$$\frac{1}{2}\Delta t = \frac{579\text{ps}}{2} \approx 290ps$$

und durch die maximale Abweichung von 34ps in der ODER-Kaskade bestimmt. In der Simulation liegt der Jitter bei 192ps und somit innerhalb der vorhergesagten Grenze. Damit ist die Konstruktion des Prototypen abgeschlossen. Seine exakte Implementierung ist aus dem Quellcode im Anhang in Listing *A.3* ersichtlich.

Tabelle 6.3: Liste der minimalen und maximalen Signallaufzeiten in den sechs Ebenen der ODER-Kaskade des endgültigen Designs. Diese Lösungen wurden durch die automatisierte Laufzeitanalyse gefunden.

Ebene	Min [ps]	Max [ps]	Abweichung [ps]
1	387	385	2
2	452	450	2
3	462	456	6
4	659	648	11
5	670	664	6
6	921	914	7
Gesamt	3551	3517	34

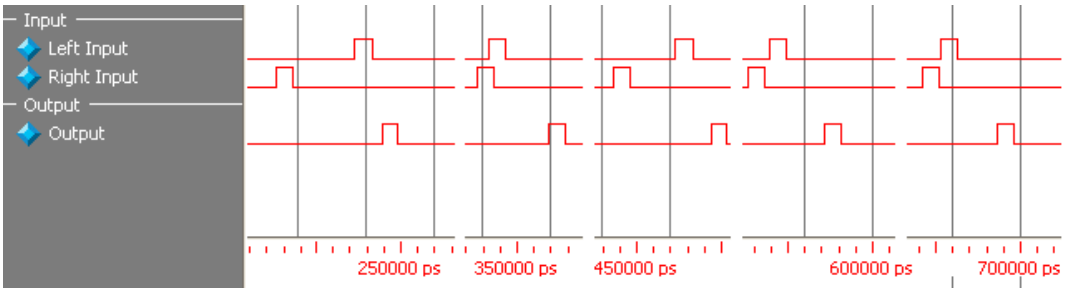


Abbildung 6.8: Eine Simulation des Prototypen für fünf verschiedene Eingangssignaldifferenzen. Die Ergebnisse sind in Tabelle *6.4* aufgelistet.

Tabelle 6.4: Simulationsdaten des Prototypen. Die Differenz Δ zwischen dem berechneten Center und dem Output ist nahezu unabhängig von den Eingangssignalen, der Jitter beträgt nur 192ps.

Left Input [ps]	Right Input [ps]	Center [ps]	Output [ps]	Δ [ps]
236645	213346	224995,5	244770	19774,5
341645	338346	339995,5	359730	19734,5
461645	443346	452495,5	472214	19718,5
569645	563346	566495,5	586078	19582,5
676645	671346	673995,5	693579	19583,5

Abbildung 6.9: Position und Platzverbrauch des Prototypen auf dem FPGA. In der Detailansicht der verwendeten CLB-Spalte sind die rechten Step-Slices und die linken ODER-Kaskaden-Slices zu erkennen. Die CLB-Spalte ist zur besseren Erkennbarkeit alle 10 CLBs umgebrochen.

6.1.5 Testmessung mit dem Prototypen und Diskussion der Ergebnisse

Der Aufbau für die Testmessung ist in Abbildung *6.10a* dargestellt. Mit einem Dual-Gate-Generator wird alle 100ns ein 5ns Impuls erzeugt. Mit einer Fan-Out-Einheit wird das Signal verdreifacht. Eines dieser Signale wird als Referenzsignal auf ein Oszilloskop gegeben, die beiden anderen werden über schaltbare Delay-Elemente verzögert, um die Signale eines Szintillatorstreifens zu simulieren. Diese beiden verzögerten Signale werden nochmals dupliziert und je einmal auf das GANDALF-Board und das Oszilloskop geführt. Das Ausgangssignal des GANDALF-Boards wird ebenfalls auf das Oszilloskop gegeben.

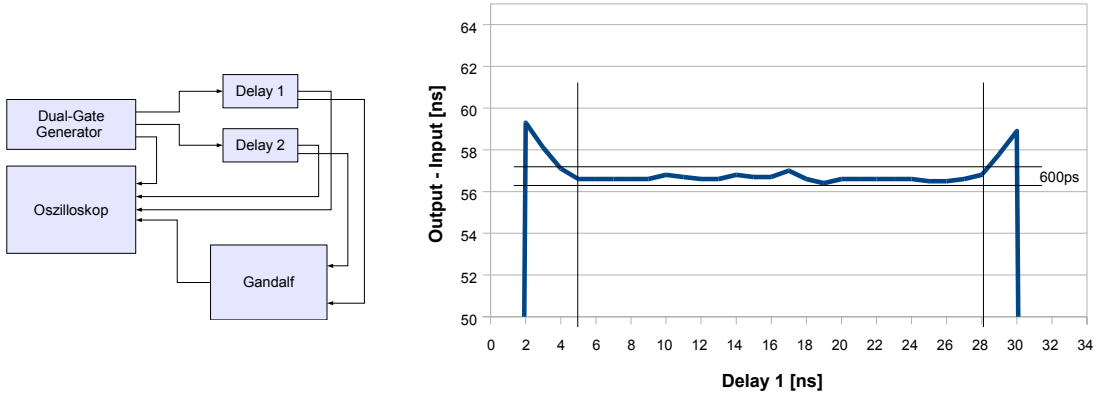


Abbildung 6.10: a) Der Aufbau für die Testmessung (links).
b) Die grafisch dargestellten Ergebnisse der Testmessung (rechts).

Die Ergebnisse der Testmessung sind in Tabelle *6.5* aufgeführt und in Abbildung *6.10b* grafisch dargestellt. Die Summe der beiden Delays ist bei allen Messwerten gleich, sodass innerhalb des dynamischen Bereichs von 30ns die Differenz Δ zwischen dem ursprünglichen Signal und dem Ausgangssignal des Meantimers ebenfalls bei allen Messwerten gleich sein sollte. Es ist deutlich zu sehen, dass der Meantimer nur in einem zentralen Bereich von ca. 23ns wie erwartet arbeitet, an den Rändern aber zu spät kommt. Die Testergebnisse werfen daher drei konkrete Fragen auf:

1. Warum kommt das Meantimer-Ausgangssignal im Randbereich zu spät?
2. Warum ist der dynamische Bereich kürzer als die erwarteten 30ns?
3. Warum ist der Jitter des Ausgangssignals mit 600ps deutlich größer als erwartet?

Zu 1: Der Effekt an den Rändern ist eine konstruktionsbedingte Eigenschaft eines TDL-Meantimers. Er tritt auf, wenn sich die beiden Signale knapp außerhalb des Meantimers treffen. Der dynamische Bereich des Prototypen beträgt somit nur 23ns, da die Randbereiche nicht dazu gehören. Das theoretische Auflösungsvermögen des Prototypen liegt demnach bei:

$$\frac{1}{2}\Delta t = \frac{23\text{ns}}{2 \cdot 54} \approx 210\text{ps}\,.$$

Tabelle 6.5: Die Ergebnisse der Testmessung. Der Prototyp ist nicht zeitkalibriert, sodass die beiden Eingangssignale vom Input-Pin des FPGAs bis zur Meantimer-Schaltung unterschiedliche Laufzeiten besitzen. Die Messdaten sind daher nicht symmetrisch. Die mit einem * versehenen Messdaten entsprechen nicht den Erwartungen.

Delay 1 [ns]	**Delay 2 [ns]**	**Delay 1 + Delay 2 [ns]**	**Δ [ns]**
1,0	31,0	32	kein Output
1,5	30,5	32	kein Output
2,0	30,0	32	59,3*
3,0	29,0	32	58,1*
4,0	28,0	32	57,1*
5,0	27,0	32	56,6
6,0	26,0	32	56,6
7,0	25,0	32	56,6
8,0	24,0	32	56,6
9,0	23,0	32	56,6
10,0	22,0	32	56,8
11,0	21,0	32	56,7
12,0	20,0	32	56,6
13,0	19,0	32	56,6
14,0	18,0	32	56,8
15,0	17,0	32	56,7
16,0	16,0	32	56,7
17,0	15,0	32	57,0
18,0	14,0	32	56,6
19,0	13,0	32	56,4
20,0	12,0	32	56,6
21,0	11,0	32	56,6
22,0	10,0	32	56,6
23,0	9,0	32	56,6
24,0	8,0	32	56,6
25,0	7,0	32	56,5
26,0	6,0	32	56,5
27,0	5,0	32	56,6
28,0	4,0	32	56,8
29,0	3,0	32	57,8*
30,0	2,0	32	58,9*
30,5	1,5	32	kein Output
31,0	1,0	32	kein Output

Diese Randerscheinung lässt sich sehr einfach erklären. Sowohl das linke als auch das rechte Signal sind 5ns lang. Entsprechend der Testmessung hat das linke Signal die TDL nach 23ns komplett durchquert und ist nach 28ns vollständig ausgelaufen. Wenn das rechte Signal erst nach 25ns in die TDL einläuft, trifft es sofort auf den Rest des linken Signals und das erste UND-Gatter erzeugt über die ODER-Kaskade das Ausgangssignal. Dieses Ausgangssignal ist jedoch zu spät, da die ersten beiden Nanosekunden des linken Signals verpasst wurden. Eine Veranschaulichung dieses Effekts ist in Abbildung *6.11* gegeben.

Die beobachtete Randerscheinung kann mit eigens dafür entwickelten Schaltungen unterdrückt werden. Diese sitzen wie Schutzkappen an den Rändern der Meantimer und werden daher als CAPs bezeichnet (s. Abbildung *6.12*). Sie bestehen aus einer Gate-LUT, die den Meantimer bei Bedarf blockiert, einem Status-Bit, mit dem der aktuelle Zustand festgehalten wird und einem TrailingEdge-Detektor. Die

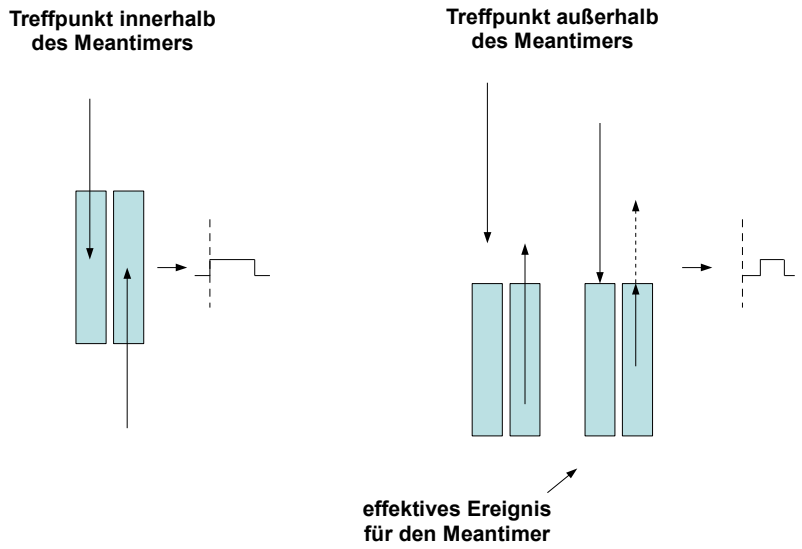


Abbildung 6.11: Vereinfachte Darstellung der einlaufenden Signale und der beiden TDLs des Meantimers zur Verdeutlichung des Randeffekts bei den Testmessungen. Die UND-Verknüpfungen zwischen den TDLs und das ODER-Segment sind nicht dargestellt.

logische Konfiguration der Gate-LUT ist in Tabelle *6.6* für die Signale der lower CAP angegeben. Erreicht das linke Eingangssignal (lIn) die Gate-LUT vor dem auslaufenden Signal der gegenläufigen TDL (rOut), dann wird das Status-Bit gesetzt. Dieses Status-Bit schaltet die Gate-LUT durch, sodass lIn unabhängig von rOut an O6 weitergereicht wird. Durch den TrailingEdge-Detektor in der CAP wird das Status-Bit wieder gelöscht, sobald lIn komplett eingelaufen ist. Sollte das Status-Bit nicht gesetzt sein, dann wird ein Einlaufen von lIn unterbunden, wenn rOut bereits anliegt. Die upper CAP funktioniert analog. Eine erneute Messung zeigt, dass die unerwünschten Randerscheinungen durch die CAPs unterdrückt werden: Bei den in Tabelle *6.5* mit einem * versehenen Messwerten wird kein Ausgangssignal mehr erzeugt.

Tabelle 6.6: Logische Konfiguration der Gate-LUT der lower CAP.

Status	rOut	lIn	lPass	SET Status
0	0	0	0	0
0	0	1	1	1
0	1	0/1	0	0
1	0/1	1	1	0
1	0/1	0	0	0

Zu 2: Die Verkürzung des dynamischen Bereichs kann dadurch erklärt werden, dass die Signallaufzeiten, mit denen die Xilinx-Software die Simulation durchführt, die maximalen Laufzeiten der Signalwege bei einer Temperatur von 85°C sind. Diese maximalen Laufzeiten sind identisch mit den in Kapitel *5.2* erwähnten SpeedGrade-Grenzwerten. Die tatsächlichen Signallaufzeiten im FPGA sind oft kürzer als diese Grenzwerte, insbesondere, wenn die Betriebstemperatur deutlich unter 85°C liegt. Die über die Laufzeitanalyse bestimmten Werte stellen somit nur eine obere Grenze dar: Die tatsächlichen Δt sind kürzer als die theoretischen 579ps.

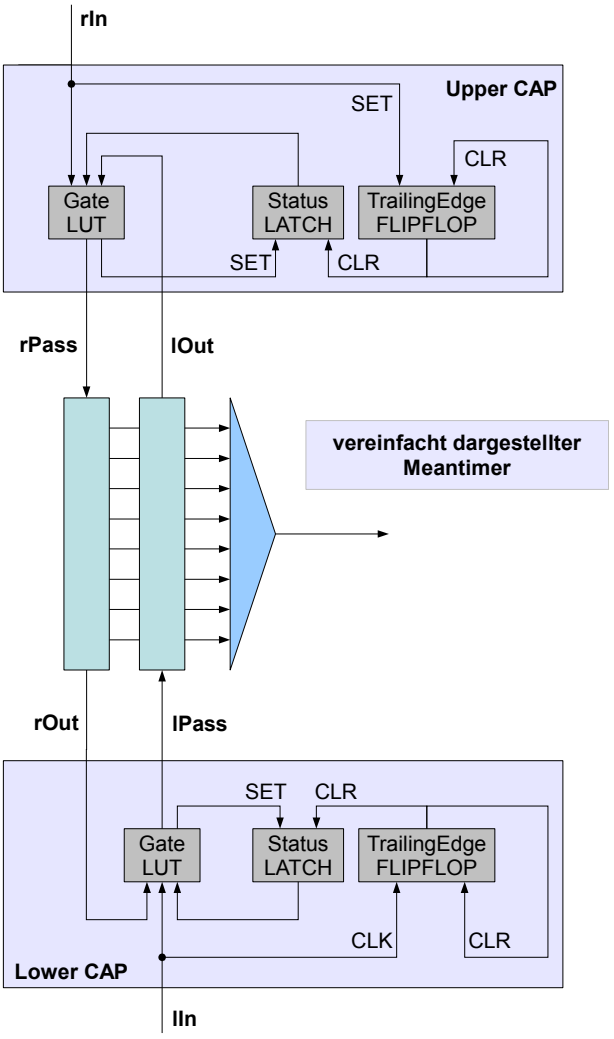


Abbildung 6.12: Schaltbild der beiden CAPs. Die TDLs des Meantimers und das ODER-Segment sind vereinfacht dargestellt.

Für eine getaktete Schaltung stellt eine solche Abweichung von der Simulation kein großes Problem dar, da es sich nicht negativ auf die Schaltung auswirkt, wenn z.B. ein Signal schon früher als simuliert einen FlipFlop erreicht. Erst durch das Taktsignal wird das angelegte Signal vom FlipFlop übernommen. Für eine getaktete Schaltung ist daher nicht die tatsächliche Laufzeit, sondern nur die maximale Laufzeit relevant, um sicherzustellen, dass das Signal auf jeden Fall vor dem Takt das gewünschte Ziel erreicht.

Zu 3: Der beobachtete Jitter des Ausgangssignals wird durch lokale Temperaturschwankungen innerhalb des FPGAs verursacht, die im ungetakteten Betrieb nicht durch das Taktsignal kontrolliert werden können. Je länger ein Signal ungetaktet durch den FPGA geroutet wird, desto größer wird der Jitter. Die Testmessung zeigt, dass der Jitter des Ausgangssignals ca. 600ps beträgt. Das theoretische Auflösungsvermögen von ca. 210ps wird demnach nicht erreicht bzw. durch Laufzeitschwankungen überlagert. Hierzu wird später mit Hilfe der TDCs[5] am COMPASS-Experiment eine genauere Messung durchgeführt.

[5] Time-to-Digital-Converter

Zusammenfassung

Die Testmessung zeigt, dass der Prototyp funktioniert. Es ergeben sich folgende Eigenschaften:

- Der dynamische Bereich des Meantimers beträgt 23ns, nicht wie gefordert 30ns.
- Er besitzt ein theoretisches Auflösungsvermögen von ca. 210ps.
- Durch Laufzeitschwankungen beträgt der Jitter des Ausgangssignals ca. 600ps, das theoretische Auflösungsvermögen kann daher nicht beobachtet werden.
- Ein störender konstruktionsbedingter Effekt an den Rändern der Meantimer kann durch eine zusätzliche Schutzschaltung unterbunden werden.

Trotz des kürzeren dynamischen Bereichs werden die Eigenschaften des Prototypen als ausreichend betrachtet, da die TDLs durch zusätzliche Step-Slices bei Bedarf verlängert werden könnten.

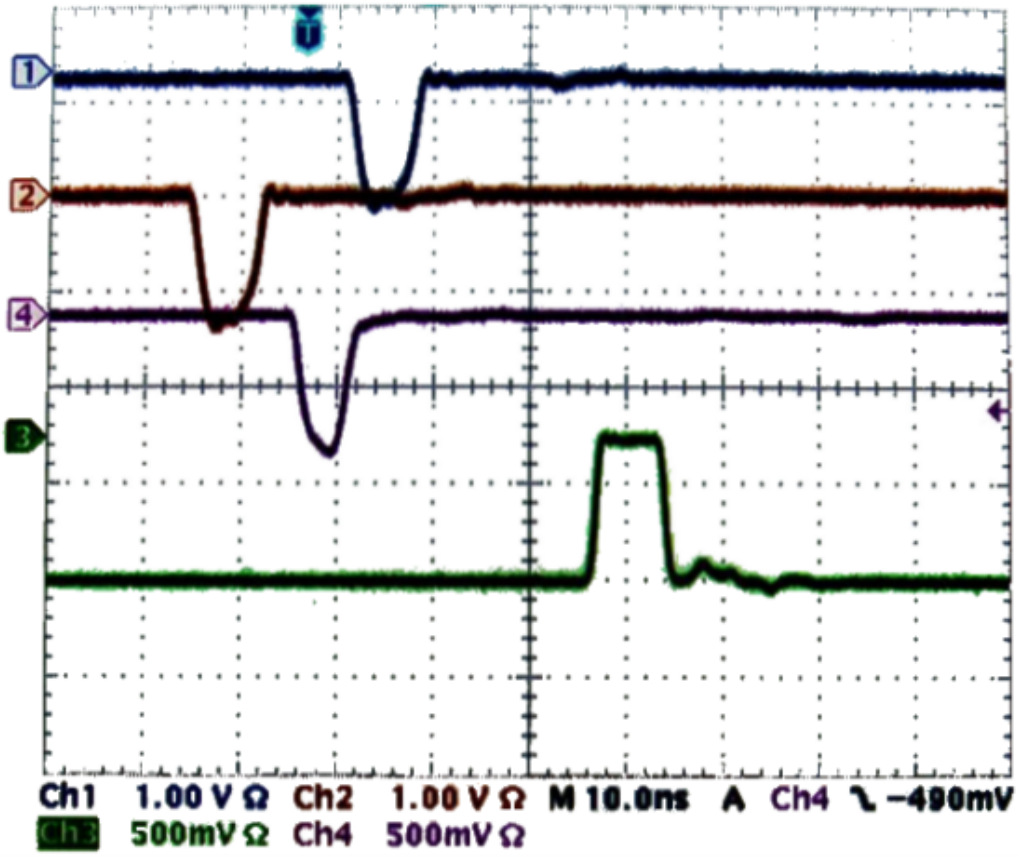


Abbildung 6.13: Die ersten Signale des Prototypen. Signal 1 ist das ursprüngliche Signal und stellt den Ereigniszeitpunkt dar. Es wurde zusätzlich verzögert, um es auch auf dem Oszilloskop sichtbar zu machen. Die verzögerten Signale 2 und 4 simulieren die beiden Ausgangssignale des Szintillatorstreifens. Das 3. Signal ist das Ausgangssignal des Meantimers.

6.2 Platzierung der 64 parallelen Meantimer

Bei der Entwicklung des Prototypen wurde bereits sichergestellt, dass die TDLs in 33 Slice-Spalten des FPGAs mit identischen Laufzeiten konstruiert werden können. Da dies für die ODER-Kaskade nicht gilt, muss für jede der möglichen Spalten eine eigene ODER-Kaskade entwickelt werden. Diese ODER-Kaskaden werden sich in zwei Punkten unterscheiden:

- In ihren Gesamt-Durchlaufzeiten und
- in ihren Differenzen zwischen minimaler und maximaler Signallaufzeit durch alle sechs Ebenen.

Die Unterschiede in den Gesamt-Durchlaufzeiten können durch die IODELAYs der jeweiligen Eingangssignale ausgeglichen werden. Die Differenzen zwischen den minimalen und maximalen Laufzeiten der Ebenen werden so gering wie möglich gehalten, sodass sie analog zum Prototyp vernachlässigt werden können.

Durch die Testmessung des Prototypen ist bekannt, dass die mit der Laufzeitanalyse bestimmten Werte nicht den tatsächlichen Laufzeiten entsprechen. Zum jetzigen Zeitpunkt ist jedoch keine alternative Analysemethode bekannt. Da der Prototyp gut funktioniert, wird diese Methode auch weiterhin eingesetzt. Es wird davon ausgegangen, dass sich Signalwege mit identischen bzw. nahezu identischen maximalen Laufzeiten auch bzgl. ihrer Abweichungen sehr ähnlich verhalten. Die Linearität jeder einzelnen ODER-Kaskade sollte somit erhalten bleiben und sich allein die Durchlaufzeit gegenüber den theoretischen Werten verkürzen. Im Folgenden wird die Abweichung von den theoretischen Laufzeiten nicht mehr explizit erwähnt.

In jeder der 33 möglichen Spalten werden zwei Meantimer konstruiert. Um den Platz für die spätere Koinzidenzschaltung zu maximieren, werden die Meantimer in zwei Blöcken am unteren und am oberen Rand des FPGAs platziert (s. Abbildung *6.14*). In der Mitte des FPGAs ergibt sich dadurch ein Freiraum von 40 CLB-Zeilen.

Zunächst werden mit Hilfe der automatisierten Laufzeitanalyse die ODER-Kaskaden aller 66 möglichen Meantimer analog zum Prototyp einzeln entwickelt und anschließend alle Meantimer parallel platziert. Ein erster Platzierungstest zeigt jedoch, dass sich die ODER-Kaskaden von zwei nebeneinander positionierten Meantimern wider Erwarten gegenseitig beeinflussen. Die Design-Software von Xilinx kann den Schritt Place & Route nicht erfolgreich abschließen. Die Laufzeitanalyse muss daher modifiziert werden. Die ODER-Kaskaden werden nun nicht mehr am Stück, sondern Ebene für Ebene analysiert, d.h. es wird zunächst die 1. Ebene nacheinander für alle 66 möglichen Positionen bestimmt und danach ein Platzierungstest durchgeführt. Sobald dieser Test erfolgreich ist, wird die nächste Ebene entwickelt. Mit diesem Verfahren kann das Place & Route für jede Ebene erfolgreich durchgeführt werden. Die Analyse aller sechs Ebenen benötigt dabei fast zwei Monate. Die fertigen Meantimer sind schematisch in Abbildung *6.14* dargestellt, die unterschiedlichen Durchlaufzeiten sind in Tabelle *6.7* aufgeführt.

Mehrere kurze Testmessungen zeigen, dass sich die neuen Meantimer analog zum Prototyp verhalten. Für die Platzierung der 64 Meantimer stehen demnach weiterhin 66 Positionen zur Verfügung. Im nächsten Kapitel wird die benötigte Koinzidenzschaltung entwickelt, wobei die beiden Meantimer in Spalte 1 nicht mehr verwendet werden.

Tabelle 6.7: Detaillierte Auflistung der minimalen und maximalen Laufzeiten in den sechs Ebenen (L1 - L6) der ODER-Kaskaden. Für jeden Meantimer sind auch die Summen der minimalen und maximalen Signallaufzeiten innerhalb der ODER-Kaskade angegeben (Min & Max). Die Differenz (Diff) dieser Signallaufzeiten, die zu einer Störung der Linearität der Meantimer führen können, sind in den übrigen Spalten meist geringer als beim Prototyp und können daher ebenfalls vernachlässigt werden. Außerdem sind die Werte für die beiden UND-Signale in den Step-Slices (lAnd & rAnd) aufgeführt. Alle Werte sind in Pikosekunden angegeben, die Spaltennummer entspricht der X-Koordinate des Slice-Koordinatensystems.

Spalte	L1	L2	L3	L4	L5	L6	Max	Min	Diff	rAnd	lAnd
1	385-387	450-452	456-462	661-666	708-714	961-968	3649	3621	28	185	660
5	385-387	450-452	456-462	648-659	664-670	914-921	3551	3517	34	185	624
9	385-387	450-452	456-462	661-666	710-721	1095-1098	3786	3757	29	185	662
13	385-387	450-452	456-462	659-665	675-681	927-932	3579	3552	27	185	634
15	385-387	450-452	456-462	658-665	603-608	1174-1177	3751	3726	25	185	632
17	385-387	450-452	456-462	662-666	676-683	1090-1095	3745	3719	26	185	634
19	385-387	450-452	456-462	660-665	888-898	1057-1058	3922	3896	26	185	634
21	385-387	450-452	456-462	661-666	710-721	1095-1098	3786	3757	29	185	662
25	385-387	450-452	456-462	659-665	675-681	927-932	3579	3552	27	185	634
27	385-387	450-452	456-462	658-665	603-608	1174-1177	3751	3726	25	185	632
29	385-387	450-452	456-462	662-666	676-683	1090-1095	3745	3719	26	185	634
31	385-387	450-452	456-462	660-665	888-898	1057-1058	3922	3896	26	185	634
33	385-387	450-452	456-462	661-666	710-721	1095-1098	3786	3757	29	185	662
37	385-387	450-452	456-462	659-665	675-681	927-932	3579	3552	27	185	634
39	385-387	450-452	456-462	658-665	603-608	1174-1177	3751	3726	25	185	632
41	385-387	450-452	456-462	662-666	676-683	1090-1095	3745	3719	26	185	634
43	385-387	450-452	456-462	660-665	888-898	1057-1058	3922	3896	26	185	634
45	385-387	450-452	456-462	661-666	710-721	1095-1098	3786	3757	29	185	662
49	385-387	450-452	456-462	620-625	644-659	1218-1219	3804	3773	31	185	614
53	385-387	450-452	456-462	658-665	713-721	1090	3777	3752	25	185	663
55	385-387	450-452	456-462	660-665	603-608	948-950	3524	3502	22	185	660
57	385-387	450-452	456-462	662-666	676-683	1090-1095	3745	3719	26	185	634
59	385-387	450-452	456-462	657-665	601-608	1173-1176	3750	3722	28	185	634
61	385-387	450-452	456-462	662-667	674-680	1095-1098	3746	3722	24	185	633
65	385-387	450-452	456-462	658-665	713-721	1090	3777	3752	25	185	663
67	385-387	450-452	456-462	660-665	603-608	948-950	3524	3502	22	185	660
69	385-387	450-452	456-462	662-666	676-683	1090-1095	3745	3719	26	185	634
71	385-387	450-452	456-462	657-665	601-608	1173-1176	3750	3722	28	185	634
73	385-387	450-452	456-462	662-667	674-680	1095-1098	3746	3722	24	185	633
77	385-387	450-452	456-462	658-665	713-721	1090	3777	3752	25	185	663
81	385-387	450-452	456-462	648-565	662-667	912-919	3543	3513	30	185	622
85	385-387	450-452	456-462	626-631	647-661	1214-1221	3814	3778	36	185	616
89	385-387	450-452	456-462	648-656	662-667	912-919	3543	3513	30	185	622

Abbildung 6.14: Darstellung der Meantimer in allen 66 möglichen Positionen mit Hilfe des Tools PlanAhead der ISE-Design-Suite. Die grünen Linien sind die Verbindungen von den INPUT-Pads zu den Rändern der Meantimer und vom letzten ODER der Kaskade zu den OUTPUT-Pads. PlanAhead stellt diese Verbindungen als gerade Linien und nicht entsprechend dem tatsächlichen Routing dar.

6.3 Entwicklung der Koinzidenzschaltung

Die direkte Implementierung der 32×32 Koinzidenzschaltung erfordert die parallele Prüfung aller 1024 Kombinationen. Es müssen dazu von den Ausgängen der 2×32 Meantimer insgesamt 2048 Signalpfade gefunden werden, die sich paarweise mit identischen Laufzeiten in jeweils einem UND-Gatter treffen. Bei der Entwicklung der ODER-Kaskaden wurden 768 ganz ähnliche Pfade gesucht und es zeigte sich, dass das Analyseverfahren fast zwei Monate benötigte, um für alle diese Pfade eine Lösung zu finden. Für die Koinzidenzschaltung wird diese Analyse sogar noch länger dauern, sodass der direkte Ansatz verworfen wird.

Es wird vielmehr versucht, die matrixartige und regelmäßige Struktur der Koinzidenzschaltung auszunutzen, die bereits in Abbildung *3.9* auf Seite 16 zu erkennen ist. Bei der in Abbildung *6.15* gezeigten Matrixschaltung durchlaufen die 2×32 Signale der beiden Meantimer-Gruppen die Koinzidenzprüfungen nacheinander. Die entstehenden Laufzeitunterschiede werden durch passende Verzögerungen außerhalb der Matrixschaltung ausgeglichen.

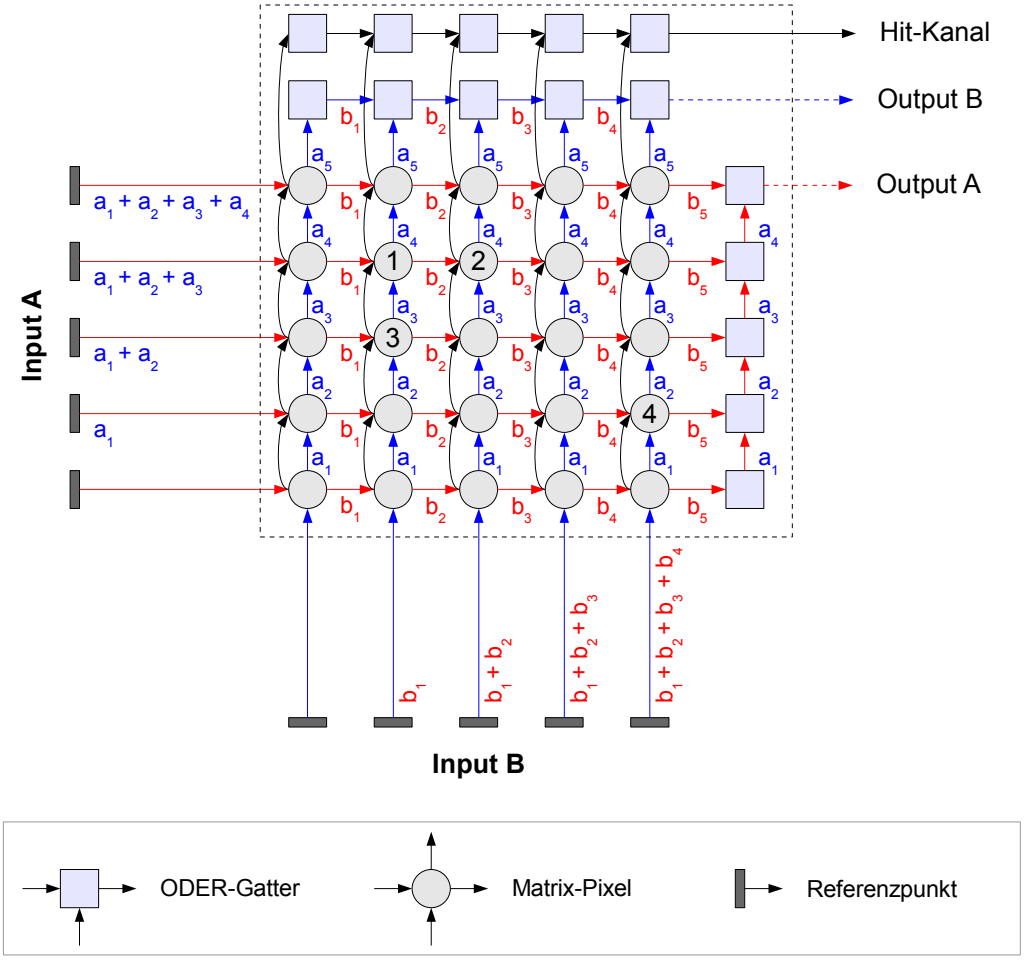


Abbildung 6.15: Die vertikalen Signallaufzeiten innerhalb der Matrixschaltung von einem Matrix-Pixel zum nächsten sind mit $a_1 ... a_5$ bezeichnet, die horizontalen mit $b_1 ... b_5$. Der Hit-Kanal transportiert die Information einer positiven Koinzidenz zwischen einem Signal der Gruppe A und einem Signal der Gruppe B zum Ausgang der Matrixschaltung.

Es ist ohne weiteres ersichtlich, dass durch die zusätzlichen Verzögerungen außerhalb der Matrixschaltung, die Laufzeiten in allen horizontalen Signalwegen von den Referenzpunkten bis zum letzten ODER-Segment der Matrixschaltung identisch sind:

$$\Delta_A = a_1 + a_2 + a_3 + a_4 \; + \; b_1 + b_2 + b_3 + b_4 + b_5 \, .$$

Dies gilt analog für alle vertikalen Signalwege:

$$\Delta_B = b_1 + b_2 + b_3 + b_4 \; + \; a_1 + a_2 + a_3 + a_4 + a_5 \, .$$

Die Matrixschaltung fungiert damit für beide Signalgruppen als ein zeitstabiles ODER-Segment. Durch die regelmäßige Struktur der Matrixschaltung ist außerdem gewährleistet, dass sich zeitgleiche Meantimersignale auch zeitgleich an ihrem Matrix-Pixel (UND-Gatter) treffen. Die Laufzeiten zwei sich kreuzender Signale von ihren Referenzpunkten bis zu ihrem Treffpunkt sind identisch. Dies wird exemplarisch für die mit 1,2,3 und 4 markierten Matrix-Pixel gezeigt:

$$\begin{aligned} \Delta_{A1} &= a_1 + a_2 + a_3 + b_1 \, , \\ \Delta_{B1} &= b_1 + a_1 + a_2 + a_3 \, , \end{aligned}$$

$$\begin{aligned} \Delta_{A2} &= a_1 + a_2 + a_3 + b_1 + b_2 \, , \\ \Delta_{B2} &= b_1 + b_2 + a_1 + a_2 + a_3 \, , \end{aligned}$$

$$\begin{aligned} \Delta_{A3} &= a_1 + a_2 + b_1 \, , \\ \Delta_{B3} &= b_1 + a_1 + a_2 \, , \end{aligned}$$

$$\begin{aligned} \Delta_{A4} &= a_1 + b_1 + b_2 + b_3 + b_4 \, , \\ \Delta_{B4} &= b_1 + b_2 + b_3 + b_4 + a_1 \, . \end{aligned}$$

Die benötigten Verzögerungen außerhalb der Matrixschaltung können durch die IODELAYs eingestellt werden. Die Konstruktion der Koinzidenzschaltung basiert damit hauptsächlich auf der Entwicklung von parallelen Signalwegen mit identischen Laufzeiten, wie sie bereits bei den TDLs genutzt werden.

Um die Koinzidenzen in den einzelnen Matrix-Pixeln auch registrieren zu können, wird ein zusätzlicher Hit-Kanal benötigt. Dieser Kanal wird, wie in Abbildung *6.15* gezeigt, parallel zur vertikalen Ausbreitungsrichtung in der Matrixschaltung implementiert. Treffen sich zwei Signale in einem Pixel, dann wird diese Information auf den Hit-Kanal gegeben und bis zum Ausgang der Schaltung durchgereicht. Um dort ein zeitstabiles Koinzidenzsignal zu erhalten, muss der Hit-Kanal die selben Laufzeiten besitzen wie die parallel verlaufenden horizontalen und vertikalen Signalwege. Es ist davon auszugehen, dass dies nicht möglich sein wird, daher wird die Information des Hit-Kanals über eine UND-Verknüpfung mit einem der beiden zeitstabilen ODER-Signale der Matrixschaltung nachträglich wieder zeitstabilisiert. Diese Methode wird als ReTiming bezeichnet. Sie setzt voraus, dass eine Information im Hit-Kanal die Matrixschaltung schneller durchquert als im ODER-Kanal.

Da jeder Matrix-Pixel drei Ausgangssignale liefert, kann er nicht über eine einzelne LUT6_2 umgesetzt werden. Es werden daher analog zu den Step-Slices drei LUTs eingesetzt. Dabei fungieren zwei der LUTs als Signalduplikatoren, die jeweils das horizontale und das vertikale Eingangssignal unverändert durchreichen und eine Kopie der Signale auf die dritte LUT leiten. In dieser Hit-LUT wird die Koinzidenzprüfung mit einem einfachen UND-Gatter durchgeführt. Als drittes Eingangssignal liegt das Ausgangssignal der vorhergehenden Hit-LUT an, das über eine ODER-Verknüpfung mit dem Ergebnis der Koinzidenzprüfung zusammengeführt wird. Außerdem liegt an der Hit-LUT ein in Abbildung *6.15* nicht dargestelltes Steuersignal an, um die Koinzidenz bei Bedarf unterdrücken zu können (s. auch Kapitel *3.3*). Die logische Konfiguration der Hit-LUT ist in Tabelle *6.8* aufgeführt.

Tabelle 6.8: Logische Konfiguration der Hit-LUT.

Enable	HitIn	aAnd	bAnd	Out
0	0	x	x	0
0	1	x	x	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	x	x	1

Wenn jeder Matrix-Pixel der Matrixschaltung analog zu den Step-Slices in einer Pixel-Slice umgesetzt wird, dann belegt die Schaltung einen Bereich von 33×34 CLBs. Die Freifläche zwischen den Matrixblöcken (im Folgenden als Matrixbereich bezeichnet) beträgt 45×40 CLBs und ist damit ausreichend groß. Die bereits in der Step-Slice genutzte, nach oben verlaufende TDL kann in der Pixel-Slice als vertikaler Signalweg wiederverwendet werden. Die A6LUT der Pixel-Slice ist daher bereits belegt. Analog zu Kapitel *6.1.2* wird nun eine Laufzeitanalyse aller horizontalen Inter-CLB-Verbindungen des Matrixbereichs durchgeführt. Dabei wird nach einem Signalweg durch die B6LUT, C6LUT oder D6LUT der Pixel-Slice gesucht, der die in Abbildung *6.15* erkennbaren Bedingungen für die b_i-Signale im gesamten Matrixbereich erfüllt.

Die Laufzeitanalyse zeigt, dass nur der horizontale Signalweg durch die B6LUT über den I2-Pin für die Matrixschaltung in Frage kommt. Bei dieser PIN-BEL-Kombination besitzt der horizontale Signalweg zwischen 32 der 33 Spalten in allen 34 Zeilen die gleiche Signallaufzeit. Zwischen der 18. und der 19. Spalte variiert die Laufzeit um lediglich 10ps. Die geforderten Bedingungen werden also ähnlich wie bei den ODER-Kaskaden nicht komplett, aber ausreichend gut erfüllt. Die Ergebnisse der Analyse sind in Tabelle *6.10* aufgeführt.

Es müssen noch die Pfade für den vertikalen Hit-Kanal, für die beiden koinzidenzbildenden Signale aAnd und bAnd (analog zu lAnd und rAnd in den Step-Slices) und für die drei Richtungswechsel-Brücken topSwitch, rightSwitch und hitSwitch gefunden werden (s. Abbildung *6.16*). Die aAnds müssen analog zu a_i in jeder Zeile identisch sein, die bAnds analog zu b_i in jeder Spalte. Eine Differenz zwischen aAnd und bAnd kann wieder über die IODELAYs der beteiligten Meantimer korrigiert werden. Weiterhin müssen jeweils alle Switch-Signale die gleiche Laufzeit besitzen. Für die einzelnen Hit-Kanal-Segmente h_i gilt, dass sie kürzer als a_i und b_i sein

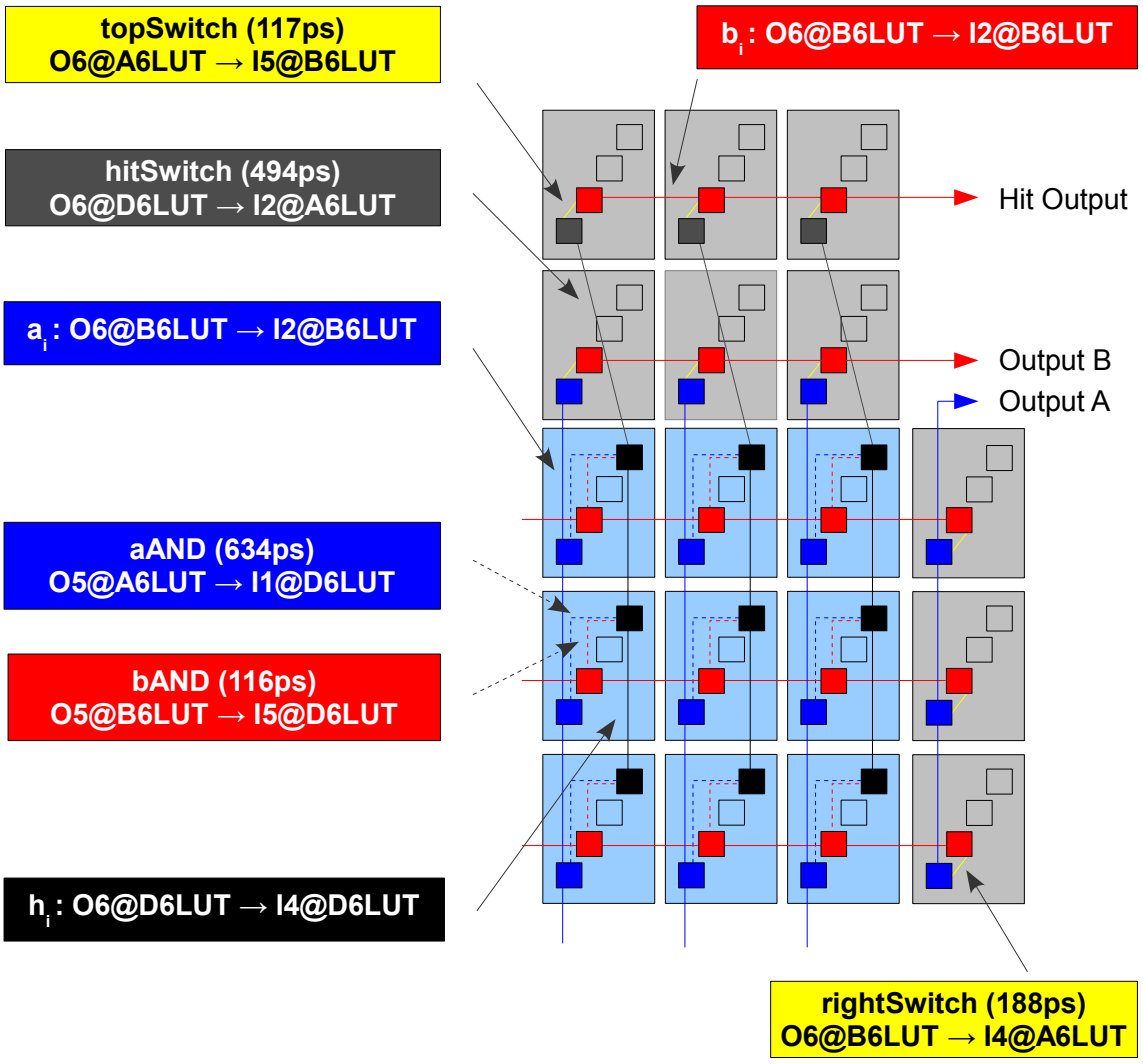


Abbildung 6.16: Darstellung der gefundenen PIN-BEL-Kombinationen der verschiedenen Elemente der Matrixschaltung. Abgebildet sind neun der 1024 regelmäßigen Pixel-Slices und die dazugehörenden Teile der drei Oder-Segmente. Die rote und die blaue LUT in der Pixel-Slice duplizieren die Eingangssignale. In der schwarzen LUT werden die Kopien auf Koinzidenz geprüft (UND-Gatter) und wenn nötig auch der Hit-Kanal aktualisiert.

müssen. Glücklicherweise können alle Bedingungen erfüllt werden, die Ergebnisse sind Tabelle *6.11* und Abbildung *6.16* zu entnehmen.

Mit einem IODELAY des Virtex 5 lassen sich die Eingangssignale um maximal 5ns verzögern. Das ist jedoch nicht mehr ausreichend, um auch die benötigten Verzögerungen außerhalb der Matrixschaltung einzustellen. Um größere Verzögerungen zu erreichen, könnten zwei oder mehr IODELAYs in Reihe geschaltet werden. Hierbei zeigt sich jedoch, dass dadurch enorme zusätzliche Laufzeiten (+50%) innerhalb des FPGAs entstehen, da diese IODELAYs nur in einem großen Abstand zueinander platziert werden können. Die dadurch entstehende Vergrößerung des Jitters des Ausgangssignals ist nicht akzeptabel. Um die benötigten Verzögerungen dennoch einstellen zu können, werden die in Abbildung *6.17* gezeigten schaltbaren SWITCH-Delays entwickelt. Sie schalten je nach angelegtem Steuersignal eine zusätzliche Verzögerung von ca. 1ns in den Signalweg, sind frei platzierbar und führen nicht zu unerwünschten zusätzlichen Laufzeiten innerhalb des FPGAs.

Die Pixel-Slices nutzen wie die Step-Slices nur die rechten SLICELs der CLBs. Alle linken Slices im Matrixbereich sind frei und können für die zusätzlichen SWITCH-Delays genutzt werden.

Tabelle 6.9: Logische Konfiguration des SWITCH-Delays.

Enable	Input	Delay-In	Output	Delay-Out
0	0	x	0	0
0	1	x	1	0
1	0	0	0	0
1	1	0	0	1
1	1	1	1	1
1	0	1	1	0

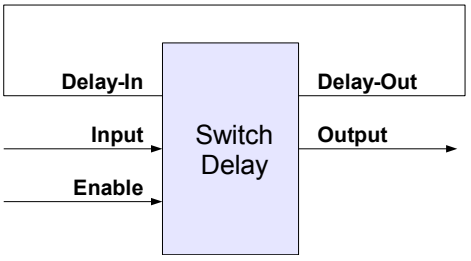


Abbildung 6.17: In Abhängigkeit des angelegten Enable-Signals wird der Input entweder direkt auf den Output gelegt oder über Delay-Out und Delay-In verzögert.

Die beiden Hodoskope, die beim COMPASS-Experiment die Signale für die 64 Meantimer liefern, werden mit H1 und H2 bezeichnet. Die Signale von H1 werden auf Input B und die Signale von H2 auf Input A der Matrixschaltung gegeben (s. Abbildung *6.15*). Dementsprechend wird im Folgenden der Output B mit H1-OUT und der Output A mit H2-OUT bezeichnet. Das Ausgangssignal des Hit-Kanals wird als PM-OUT (Pure Matrix) bezeichnet. Das über H1-OUT zeitstabilisierte PM-Signal wird als MATRIX-OUT bezeichnet. Außerdem wird ein logisches ODER zwischen H1-OUT und H2-OUT benötigt, dieses Ausgangssignal wird als FULL-OUT bezeichnet.

Tabelle 6.10: Die Tabelle zeigt die Ergebnisse der Laufzeitanalyse für den horizontalen Signalweg in der B6LUT über den I2-Pin. Der Spalten-Index 0 bezieht sich auf das Signal zwischen der 1. und der 2. Spalte. Zwischen allen Spalten (außer zwischen der 18. und der 19.) ist die Laufzeit in allen 34 Zeilen wie gefordert gleich. Die Werte von b_{17} sind für jede Zeile einzeln aufgelistet, um die Abweichungen aufzuzeigen.

Spalten-Index	b_i **[ps]**	**Zeilen-Index**	b_{17} **[ps]**
		33	738
		32	738
0	493	**31**	736
1	520	**30**	736
2	493	**29**	736
3	582	**28**	736
4	496	**27**	736
5	520	**26**	736
6	493	**25**	728
7	520	**24**	728
8	493	**23**	728
9	582	**22**	728
10	496	**21**	728
11	520	**20**	728
12	493	**19**	728
13	520	**18**	728
14	493	**17**	728
15	582	**16**	728
16	496	**15**	728
17	728,736,738	**14**	728
18	496	**13**	728
19	590	**12**	728
20	493	**11**	728
21	520	**10**	728
22	493	**9**	728
23	515	**8**	728
24	496	**7**	728
25	590	**6**	728
26	493	**5**	736
27	520	**4**	736
28	493	**3**	736
29	515	**2**	736
30	496	**1**	736
31	590	**0**	736

Tabelle 6.11: Die Laufzeiten der hier aufgeführten Signale sind in allen Spalten des Matrixbereichs identisch. Die Unterschiede in den Zeilen entstehen durch die bereits auf Seite 44 angesprochenen Lücken in der FPGA Struktur.

Zeilen-Index	a_i **[ps]**	h_i **[ps]**	**aAnd [ps]**	**bAnd [ps]**
4,5,24,25	496	323	634	116
$\star$	493	320	632	116

6.4 Entwicklung einer Web-Schnittstelle zur Konfiguration des Systems

6.4.1 Die VME-Schnittstelle

Bevor die entwickelte Trigger-Elektronik eingesetzt werden kann, muss eine Zeitkalibrierung durchgeführt werden. Dazu müssen die IODELAYs in den INPUT-Pads und die SWITCH-Delays zur Laufzeit konfiguriert werden können. Des Weiteren soll es zur Laufzeit möglich sein, die Koinzidenzmatrix der Koinzidenzschaltung zu verändern.

Das GANDALF-Board besitzt für diese Aufgabe ein CPLD-Interface, das als Schnittstelle zwischen dem FPGA und dem VME-BUS dient. Mit Hilfe spezieller VME-Tools können bestimmte Block-RAM-Zellen auf dem FPGA über ein einfaches auf dem Crate-PC ausgeführtes Shell-Skript ausgelesen und geändert werden. Die VME-Tools werden durch die Entwickler des GANDALF-Boards zur Verfügung gestellt.

Listing 6.2: Einfaches Shell-Skript, um auf die Speicherzellen innerhalb des FPGAs zuzugreifen.

```
1  #!/bin/bash
2
3  TOOL_DIR=/compass/gandalf/tools
4  BOARD_ID=E003
5  HEX_VALUE=00000000
6  BRAM_ADDR=DFC
7  ACCESS_MODE=6
8
9  #Update value of BRAM-Cell
10 ${TOOL_DIR}/vme_write ${BOARD_ID}${ACCESS_MODE}${BRAM_ADDR} ${HEX_VALUE}
11
12 #Trigger FPGA update
13 ${TOOL_DIR}/vme_write ${BOARD_ID}7040 2
```

Für den ACCESS_MODE können drei verschiedene Werte genutzt werden. Mit 2 wird der Inhalt der adressierten 32-Bit-Zelle ausgelesen und auf der Konsole ausgegeben und mit 6 wird ihr Inhalt durch HEX_VALUE überschrieben.

Über ACCESS_MODE=7 wird keine Block-RAM-Zelle, sondern direkt eines der 256 sog. Fastregister (FlipFlops) adressiert. Wird einem solchen 1-Bit-Fastregister, wie in Listing *6.2*, der Wert 2 übergeben, setzt es das CPLD-Interface zunächst auf 1 und nach 25ns automatisch auf 0 zurück. Auf diese Weise steht die Information, dass neue Daten in den Block-RAM geschrieben wurden, direkt innerhalb des FPGAs zur Verfügung[6]. Auf dem FPGA kann daraufhin ein Prozess in Gang gesetzt werden, der den Block-RAM neu ausliest und z.B. die IODELAYs oder die Koinzidenzmatrix rekonfiguriert.

6.4.2 Entwicklung einer getakteten Konfigurationsschaltung

Um auf die steigende Flanke des Update-Fastregisters reagieren zu können, wird eine mit 40 MHz getaktete Konfigurationsschaltung entwickelt. Ihre Aufgabe besteht darin, nach einem Speicherupdate in allen konfigurierbaren Elementen des FPGAs

[6] Alternativ müsste der FPGA den Inhalt einer bestimmten Block-RAM-Zelle überwachen.

einen RESET durchzuführen und sie danach gemäß der Block-RAM-Inhalte neu einzustellen.

Die Verzögerung in den IODELAYs wird in 64 Stufen zu je 75ps eingestellt. Um die Verzögerung z.B. auf 12×75ps einzustellen, muss für 12 Takte sowohl auf den sog. INC-Pin als auch auf den sog. CE-Pin des IODELAYs eine logische 1 gegeben werden (s. Abbildung *6.18*). Nach einem Speicherupdate wird nacheinander für jedes IODELAY die neue Verzögerungsstufe aus der entsprechenden Block-RAM-Zelle[7] in ein 6-Bit-Register kopiert und dort anschließend mit jedem Takt heruntergezählt. Das logische ODER aller 6-Bit des Registers ist mit dem INC-Pin und dem CE-Pin des IODELAYs verbunden, sodass die gewünschte Verzögerungsstufe eingestellt wird.

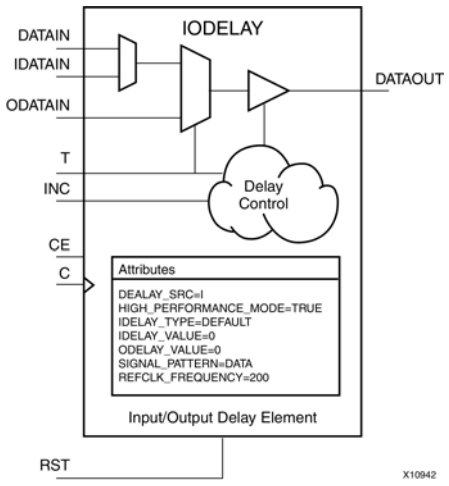


Abbildung 6.18: Das IODELAY-Element. Das zu verzögernde Signal kann entweder direkt von einem INPUT-Pad auf den IDATAIN-Pin oder von einem anderen Element des FPGAs auf den DATAIN-Pin gegeben werden. Wenn das zu verzögernde Signal danach direkt auf ein OUTPUT-Pad gegeben wird, muss der ODATAIN-Pin beschaltet werden. Über den INC-Pin wird gesteuert, ob mit jedem Takt eine Verzögerungsstufe hinzugefügt oder entfernt werden soll. [22]

Um die SWITCH-Delays analog zu den IODELAYs konfigurieren zu können, werden 21 von ihnen in Reihe geschaltet[8] und zusammen mit einem 21-Bit-Shiftregister als ein neues Modul aufgefasst (s. Abbildung *6.19*). Die Ausgangssignale der einzelnen FlipFlops des Shiftregisters sind mit den Steuereingängen der SWITCH-Delays verbunden und am Eingang des Shiftregisters liegt eine logische 1 an. Dieser SWITCH-Delay-Block kann nun analog zu den IODELAYs konfiguriert werden: Wenn auf den CE-Pin des Shiftregisters eine logische 1 gegeben wird, schaltet mit jedem Takt ein weiteres SWITCH-Delay seine zusätzliche Verzögerung in den Signalweg.

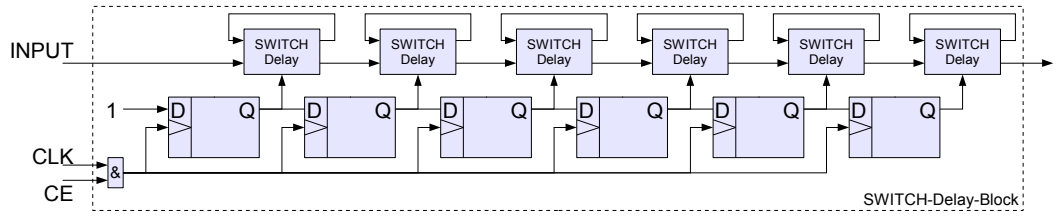


Abbildung 6.19: Ein SWITCH-Delay-Block mit sechs SWITCH-Delay-Elementen.

Alle anderen Elemente auf dem FPGA werden über Steuersignale konfiguriert. Für jede Funktion wird dabei ein FlipFlop benötigt, dessen Ausgangssignal als Steuersignal auf eine LUT gegeben wird. Nach einem Speicherupdate werden diese FlipFlops

[7] In einer 32-Bit Block-RAM-Zelle werden die 6-Bit-Werte von vier IODELAYs gespeichert.
[8] Jedes SWITCH-Delay kann eine zusätzliche Verzögerung von ca. 1ns in den Signalweg schalten.

entsprechend der Block-RAM-Daten neu konfiguriert. Dies gilt für die folgenden Elemente:

- Die UND-Gatter zwischen den TDLs können einzeln an- und ausgeschaltet werden, dadurch lässt sich der dynamische Bereich der Meantimer zu Testzwecken verkürzen.
- Die Schutzfunktion der Gate-LUTs in den CAPs kann deaktiviert werden.
- Der Output des letzten ODER-Gatters der ODER-Kaskade kann unterdrückt werden, sodass die Meantimer einzeln deaktiviert werden können.
- Die 1024 Koinzidenzprüfungen der Matrixschaltung können einzeln unterdrückt werden.
- Zu Testzwecken können alle Meantimer mit dem Signal eines internen Signalgenerators[9] beschaltet werden.

Mit den internen Testsignalen wird in Kapitel *6.5* ein Funktionstest der Matrixschaltung durchgeführt. Um die Matrixschaltung für diesen Test zu kalibrieren, wird das Testsignal als Referenz auch auf einem externen Oszilloskop benötigt und daher als sog. Alignment-Signal nach außen geführt. Das Alignment-Signal kann über ein feines und ein grobes Delay zwischen 50ns und 150ns verzögert werden, um es mit den verschiedenen Ausgängen der Matrixschaltung überlagern zu können[10].

Ein weiteres Element der getakteten Konfigurationsschaltung ist der Output-Selector, mit dem vier der sechs möglichen Ausgangssignale (H1-OUT, H2-OUT, FULL-OUT, PM-OUT, MATRIX-OUT und das Alignment-Signal) auf die vier NIM-Ausgänge des GANDALF-Boards gelegt werden können. Der Output-Selector besteht aus mehreren Demultiplexern, die ebenfalls über Steuersignale konfiguriert werden.

Weitere Details zu der getakteten Konfigurationsschaltung können dem Quellcode im Anhang in Listing *A.1* entnommen werden.

6.4.3 Das Web-Interface

Um die Konfiguration der diversen Elemente nicht über die Konsole durchführen zu müssen, wird auf dem Crate-PC ein Web-Server installiert. Ein Web-Interface dient als grafische Bedienoberfläche, über das der Nutzer die aktuellen Einstellungen einsehen und verändern kann. Es wird dabei die Java-basierte AJAX[11]-Technik eingesetzt, sodass jede User-Aktion im Hintergrund ein PHP[12]-Skript auf dem Web-Server ausführt, das die notwendigen Aufrufe von vme_write durchführt. Über das Web-Interface kann die aktuelle Konfiguration des Systems auch gespeichert und bei Bedarf später wieder geladen werden.

Das Interface wurde unabhängig von dieser Diplomarbeit entwickelt und für die hier benötigte Anwendung adaptiert. Die Abbildungen *6.20* und *6.21* zeigen Screenshots des Web-Interfaces.

[9] Wird durch 120MHz Takt erzeugt: Ein Takt high, 31 Takte low (8.3ns Impuls alle 266.6ns).
[10] Die Laufzeit von den INPUT-Pads durch die Meantimer und die Matrixschaltung bis zu einem der OUTPUT-Pads beträgt ca. 100ns.
[11] Asynchronous JavaScript And XML
[12] Personal Home Page Tools

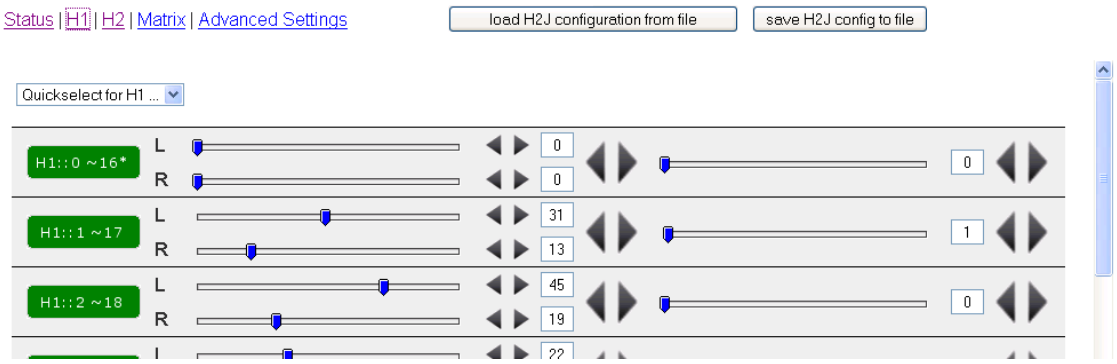


Abbildung 6.20: Auf der linken Seite befinden sich die Schieberegler, um die feinstufigen Verzögerungen der IODELAYs der beiden Eingangssignale der Meantimer einzustellen. Rechts sind die Schieberegler für die groben SWITCH-Delay-Blöcke zu erkennen, welche die Ausgangssignale der Meantimer auf dem Weg in die Matrixschaltung verzögern. Werden die beiden IODELAY-Regler eines Meantimers gleichzeitig verändert, wird der Meantimer als Ganzes verschoben. Auf diese Weise kann die Verzögerung außerhalb der Matrixschaltung feinjustiert werden. Durch einen Klick auf die grünen Buttons können die Meantimer deaktiviert werden.

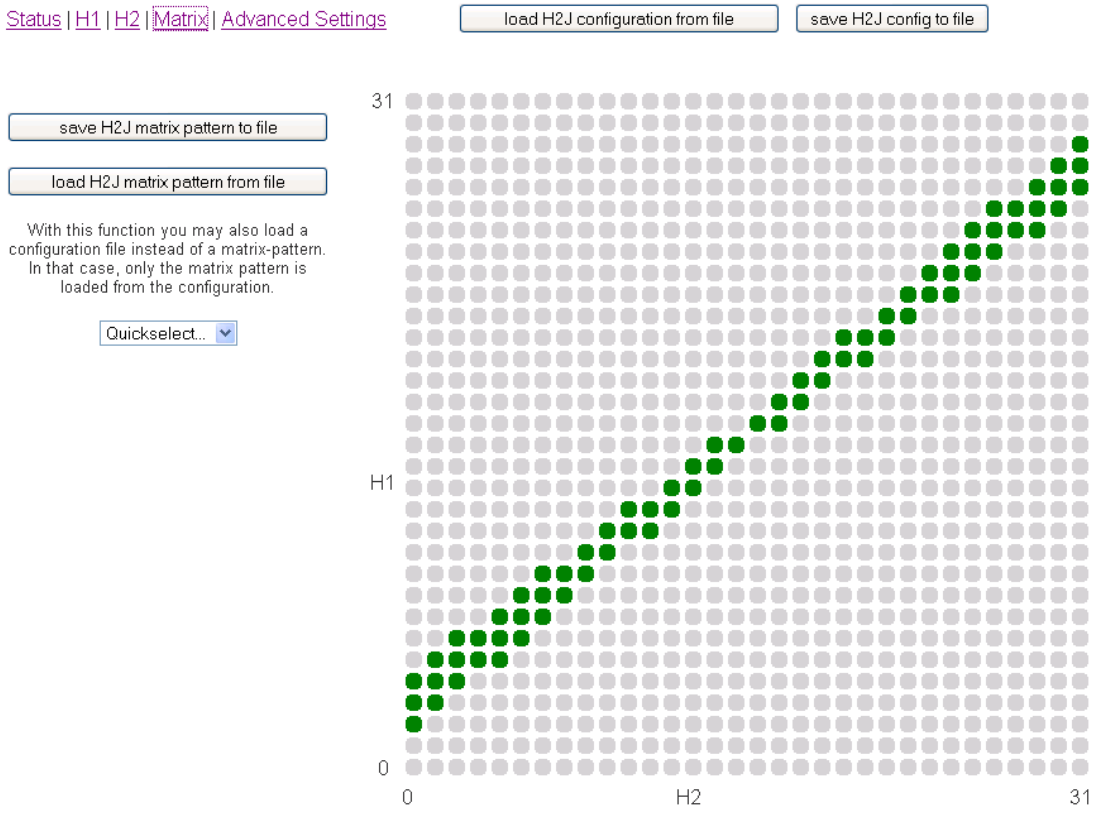


Abbildung 6.21: Bei den grauen Matrix-Pixeln der Koinzidenzmatrix wird die Koinzidenzprüfung unterdrückt. Durch einen Mausklick kann jeder Matrix-Pixel individuell an- bzw. ausgeschaltet werden.

6.5 Abschließender Funktionstest

Die Entwicklung der Trigger-Elektronik ist nun abgeschlossen und ihre wesentlichen Komponenten sind in Abbildung *6.22* zu erkennen.

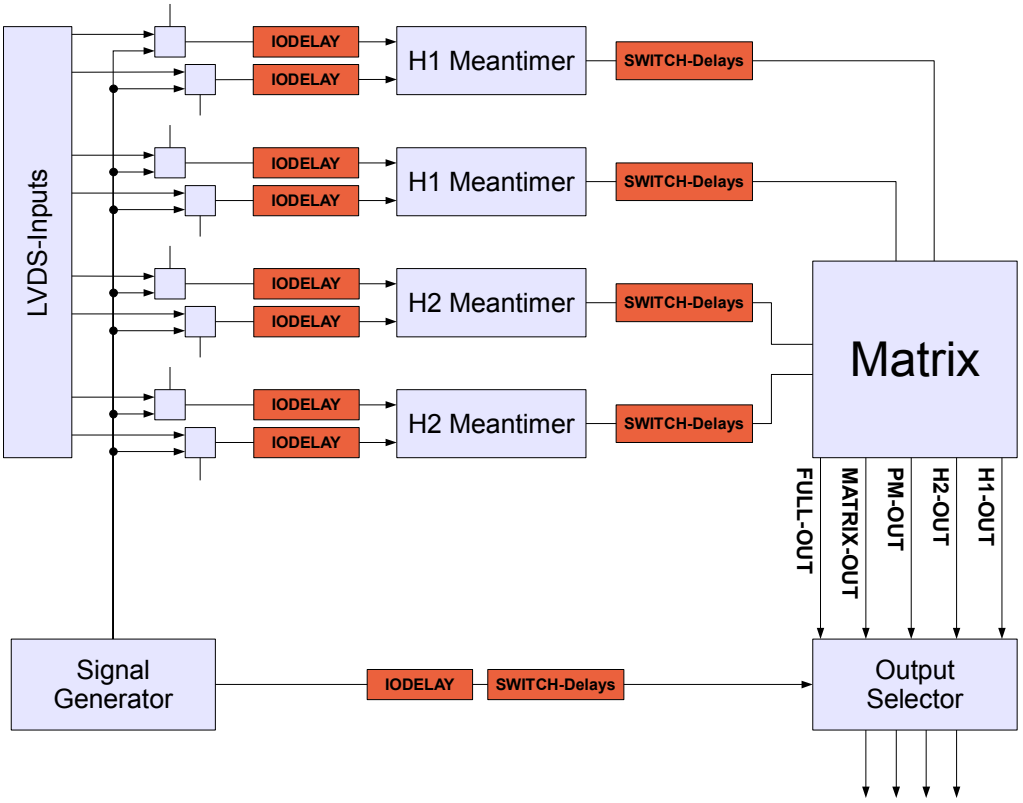


Abbildung 6.22: Schematische Darstellung der wichtigsten Elemente der entwickelten Trigger-Elektronik.

Das Zusammenspiel all dieser Komponenten wird an dieser Stelle mit Hilfe von Testsignalen überprüft. Die dafür nötige Zeitkalibrierung gestaltet sich folgendermaßen:

1. Zunächst werden alle Delays auf 0 gesetzt, die Testsignale aktiviert und alle UND-Gatter und beide CAPs aktiviert. Über den Output-Selector werden das Alignment-Signal, H1-OUT und H2-OUT auf die NIM-Ausgänge gelegt und auf dem Oszilloskop dargestellt. Auf das Alignment-Signal wird getriggert und es wird so verzögert, dass es sich auf dem Oszilloskop links von allen anderen Signalen befindet.

2. Als zweites wird mit Hilfe des Oszilloskops derjenige H1-Meantimer bestimmt, dessen H1-OUT-Signal die größte Differenz zum Alignment-Signal besitzt. Wenn dieser sog. Base-Meantimer, der die größte FPGA-interne Signallaufzeit besitzt, später korrekt eingestellt wird, können danach alle anderen Meantimer durch zusätzliche Verzögerung an ihn angepasst werden. Analog wird auch der Base-Meantimer für H2 bestimmt.

3. Anschließend werden die beiden Base-Meantimer und deren gemeinsamer Matrix-Pixel aktiviert. Alle anderen Meantimer und Matrix-Pixel werden deaktiviert und das PM-Signal wird über den Output-Selector auf dem Oszilloskop dargestellt. Mit den SWITCH-Delays werden die beiden Base-Meantimer relativ zueinander verschoben, bis auf dem PM-Signal eine Koinzidenz erkennbar ist.

4. Als Nächstes wird mit Hilfe des PM-Signals und den IODELAYs bestimmt, welcher der beiden Base-Meantimer in der aktuellen Konfiguration zuerst den Matrix-Pixel erreicht: Verzögert man den später einlaufenden Meantimer, dann verschiebt sich das PM-Signal. Wird hingegen der zuerst einlaufende Meantimer verzögert, dann bleibt das PM-Signal solange unverändert stehen, bis dieser Meantimer durch die Verzögerung zum später einlaufenden Meantimer wird. Dieser Test lässt sich sehr gut mit Hilfe des Alignment-Signals durchführen. Dazu wird das Alignment-Signal zunächst an dem PM-Signal vorbeigeschoben und die mathematische UND-Funktion zwischen dem Alignment-Signal und dem PM-Signal im Oszilloskop eingeblendet. Danach wird das Alignment-Signal soweit zurück in Richtung des PM-Signals geschoben, dass gerade noch kein logisches UND existiert. Wenn sich das PM-Signal jetzt nach rechts verschiebt, ist das sofort durch einen Peak in der UND-Kurve erkennbar. Mit dieser Methode werden die beiden Base-Meantimer so eingestellt, dass sie gleichzeitig am Matrix-Pixel ankommen.

5. In einem letzten Schritt werden wieder H1-OUT und H2-OUT auf das Oszilloskop gelegt und deren OUPUT-Delays so angepasst, dass die Gleichzeitigkeit der beiden Base-Meantimer auch auf dem Oszilloskop erkennbar ist. Nacheinander werden dann die einzelnen Meantimer aktiviert und über die SWITCH- und IODELAYs mit ihrem jeweiligen Base-Meantimer in Deckung gebracht. Die Base-Meantimer selbst dürfen dabei nicht mehr verschoben werden! Zur optischen Unterstützung kann erneut das Alignment-Signal benutzt werden. Die Zeitkalibrierung ist abgeschlossen, wenn sich alle H1-Meantimer zu einem Signal auf H1-OUT und alle H2-Meantimer zu einem Signal auf H2-OUT überlagern und dadurch die geforderten Bedingungen der Matrixschaltung erfüllt werden.

Die Funktion der Matrixschaltung wird nun mit einigen Tests überprüft. Dazu werden verschiedene Paare von H1- und H2-Meantimern[13] und ihr gemeinsamer Matrix-Pixel aktiviert. Wie erwartet kann jedesmal auf dem MATRIX-OUT-Signal die entsprechende Koinzidenz beobachtet werden. Wenn der Matrix-Pixel deaktiviert wird, verschwindet das MATRIX-OUT-Signal. Gleiches gilt auch, wenn alle anderen außer dem eigentlichen Matrix-Pixel aktiviert werden. Bei Aktivierung aller Matrix-Pixel und aller Meantimer überlagern sich die Koinzidenzen wie erwartet zu einem einzigen MATRIX-OUT-Signal.

Die Differenz zwischen dem Alignment-Signal und dem Koinzidenzsignal (MATRIX-OUT) ändert sich während der verschiedenen Tests nicht, d.h. die nachträgliche Zeitstabilisierung des PM-Signals funktioniert wie vorgesehen. Die konstruktionsbedingte Variation der Signallänge von MATRIX-OUT kann ebenfalls beobachtet werden.

Die Funktionsprüfung zeigt, dass die entwickelte Trigger-Elektronik wie erwartet funktioniert. Sie kann nun am COMPASS-Experiment installiert werden.

[13] Es ist immer nur ein H1- und ein H2-Meantimer gleichzeitig aktiv.

6.6 Zusammenfassung der Erkenntnisse bei der Entwicklung von ungetakteten Schaltungen

Folgende Erkenntnisse wurden während der Entwicklung der Meantimer und der Koinzidenzschaltung gewonnen:

- Die Laufzeit zwischen zwei LUTs auf dem FPGA ändert sich, sobald ein anderer PIN der Ziel-LUT angesteuert wird.
- Da die Switch-Matritzen nicht identisch sind, verändern sich die Laufzeiten innerhalb einer Schaltung, wenn diese auf dem FPGA verschoben wird. Pfade mit identischen Laufzeiten zu suchen ist daher sehr aufwändig. Bei der Entwicklung der TDLs müssen deshalb alle Kombinationen mehrfach gemessen und verschoben werden, um Pfade zu finden, bei denen die Laufzeiten sich nicht ändern.
- Die zeitstabile Zusammenführung mehrerer Signale an einem Punkt, um sie über eine logische Operation miteinander zu verknüpfen, ist ebenfalls sehr schwierig. In dieser Arbeit wurde dafür eine automatisierte Laufzeitanalyse eingesetzt, mit der die unzähligen PIN-BEL-LOC Kombinationen nacheinander überprüft wurden. Auch damit war es nicht immer möglich, Pfade mit identischen Laufzeiten zu finden. Erst durch die Reduktion auf zwei zeitstabil zusammengeführte Signale ergaben sich Pfade, deren Abweichung nur noch wenige Pikosekunden betrug.
- Da die Xilinx-Design-Software für die Entwicklung von getakteten Schaltungen entworfen wurde, arbeitet sie bei der Timing-Analyse mit den maximalen Signallaufzeiten bei der maximal zulässigen Betriebstemperatur. Für getaktete Schaltungen sind allein diese maximalen Laufzeiten relevant, um die sog. Setup- und Hold-Zeiten zu gewährleisten. Es ist völlig unerheblich, ob ein Signal schon deutlich früher als simuliert einen FlipFlop erreicht, weil es erst durch das Taktsignal hineingeschrieben wird. Bei ungetakteten Schaltungen zählt jedoch die tatsächliche Laufzeit, wodurch die in dieser Diplomarbeit entwickelte ungetaktete Schaltung bei Raumtemperatur ca. 20% schneller ist, als durch die Timing-Analyse erwartet wird.
- Diese Temperaturabhängigkeit führt bei lokalen Temperaturschwankungen auf dem FPGA zu Laufzeitschwankungen. Da diese Schwankungen nicht durch den Takt kontrolliert werden, ergibt sich bei besonders langen ungetakteten Laufzeiten innerhalb des FPGAs ein nicht unerheblicher Output-Jitter.

7. Integration der neuen Komponenten in das COMPASS-Trigger-System

In diesem Kapitel wird zunächst die Installation der neu entwickelten Trigger-Elektronik des LAS-Triggers mit allen zusätzlich benötigten Komponenten beschrieben. Anschließend erfolgt eine Messung mit den COMPASS-TDCs, um das bereits bekannte Jitterverhalten der ungetakteten Schaltung auf dem FPGA näher zu untersuchen. Danach wird der LAS-Trigger über das sog. Matrix-Pixel-Timing mit Strahlmyonen zeitkalibriert, sowie eine Messung des Trigger-Timings durchgeführt, um die Eigenschaften der neuen Trigger-Elektronik zu bestimmen.

7.1 Einbau der neuen Komponenten

Wie bereits in Kapitel *4.4* kurz erläutert, ist das Hodoskop H2 des LAS-Triggers in der Mitte geteilt, damit die einzelnen Szintillatorstreifen nicht zu lang werden. Es besteht aus den beiden Teil-Hodoskopen H2J und H2S[1]. Dies hat zur Folge, dass auch die Trigger-Elektronik doppelt ausgeführt werden muss.

Die Signalkabel der 192 Photomultiplier (64 von H1, 64 von H2J und 64 von H2S) enden gemeinsam auf einem großen Patchpanel. Von dort werden sie, wie in Abbildung *7.1* schematisch dargestellt, mit Lemo-Kabeln auf 12 Constant-Fraction-Diskriminatoren (CAEN V812 mit 16 Kanälen) gegeben und digitalisiert. Die verwendeten CFDs liefern für jeden ihrer Kanäle jeweils zwei Ausgangssignale, sodass alle Signale sowohl auf TDCs als auch auf die GANDALF-Boards gegeben werden können. Da die in Freiburg hergestellten TDCs und GANDALF-Boards LVDS-Eingänge besitzen, müssen die ECL[2]-Ausgangssignale der CFDs über sog. ECL-to-LVDS-Module konvertiert werden. Um für beide Hodoskop-Systeme, für H1-H2S und für H1-H2J, eine Koinzidenzprüfung durchführen zu können, müssen die Signale von H1 dupliziert und auf zwei GANDALF-Boards geführt werden. Dies wird durch zwei LVDS-Splitter realisiert. Die GANDALF-LVDS-Eingänge sind als HONDA-Stecker ausgeführt, sodass für die COMPASS-LVDS-Flachbandkabel noch Adapter benötigt werden. Das Trigger-Signal des LAS-Triggers wird durch eine nachträgliche ODER-Verknüpfung der beiden MATRIX-OUT-Signale erzeugt.

Während der Entwicklung in Bonn wurde mit einer etwas älteren Version des GANDALF-Boards gearbeitet. Für den Einsatz am COMPASS-Experiment stehen zwei GANDALF-Boards der neuesten Generation zur Verfügung. Diese besitzen jedoch einen Konstruktionsfehler, wodurch der dritte NIM-Ausgang nicht frei beschaltet werden kann, sondern stets das Signal des ersten NIM-Ausgangs spiegelt. Mit Hilfe des Output-Selectors können aber weiterhin die jeweils benötigten Signale auf die verbleibenden drei NIM-Ausgänge geschaltet werden.

7.2 Funktionstest der Meantimer

Die neue FPGA-basierte Trigger-Elektronik besteht aus den Meantimer und der Koinzidenzschaltung. Die letztere muss zeitkalibriert werden, damit sie ordnungsgemäß funktioniert. Daher wird als erster Funktionstest eine Messung der Meantimer durchgeführt, wodurch auch die Signalkette und einige der neu eingebauten Komponenten überprüft werden können. Die Messung wird nur mit einem der GANDALF-Boards durchgeführt.

Über das Web-Interface wird zunächst genau ein H2-Meantimer aktiviert. Zusätzlich zu den 192 Photomultiplier-Signalen wird auch das H2-OUT-Signal der Matrixschaltung auf einen TDC gegeben. Für jedes Myon, das den aktiven H2-Streifen durchquert, gibt es daher drei TDC-Einträge: Einen für das H2-OUT-Signal und zwei für die linken (t_j) und rechten (t_s) Photomultiplier-Signale. Aus t_j und t_s lässt sich später offline der eigentliche Ereigniszeitpunkt, die meantime berechnen. Trägt man die berechnete meantime und die durch den FPGA bestimmte meantime gegeneinander ab, sollte sich idealerweise eine Diagonale ergeben. Die in Abbildung *7.2* dargestellten Messergebnisse bestätigten somit eindrucksvoll, dass die Meantimer wie erwartet funktionieren.

[1] J: Jura, S: Salève
[2] Emitter Coupled Logic

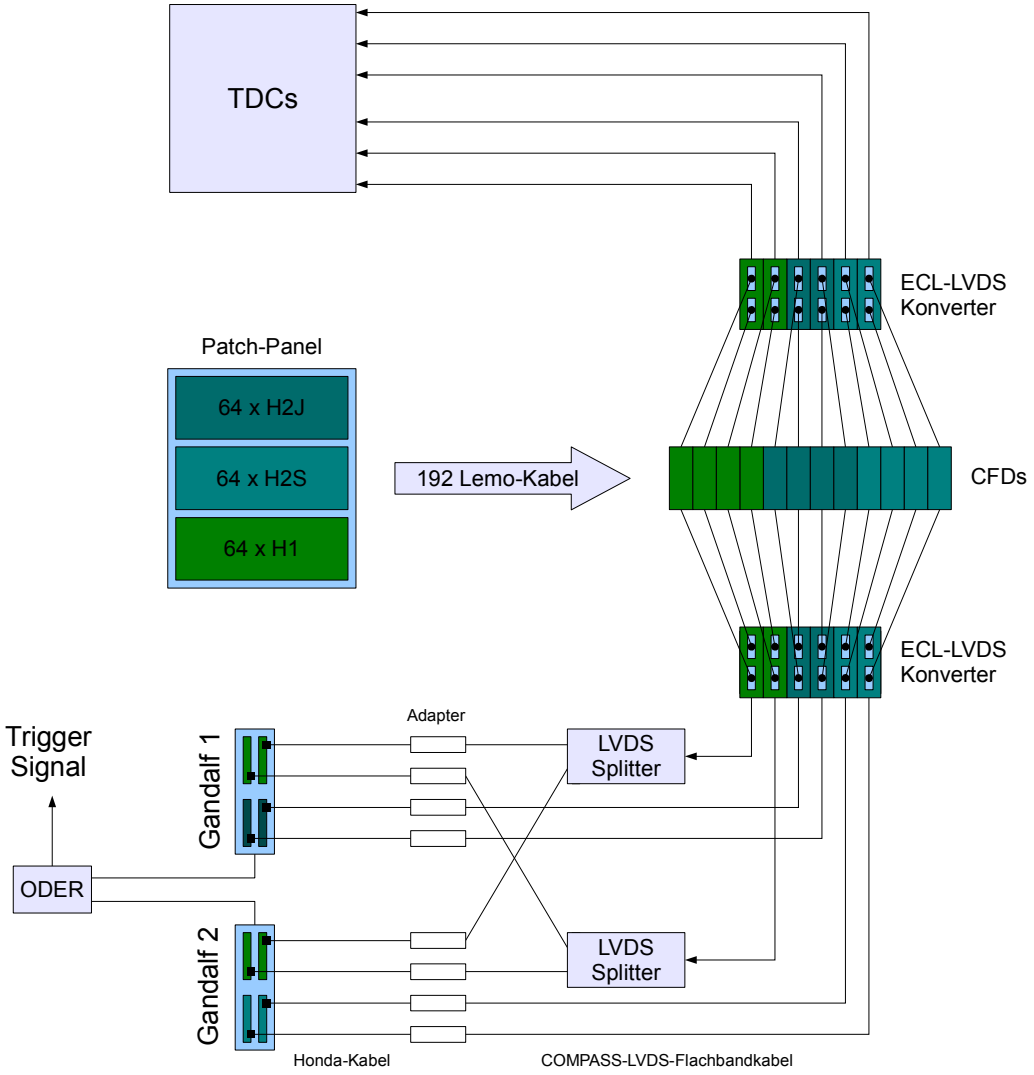


Abbildung 7.1: Schaltplan der elektronischen Komponenten des LAS-Triggers.

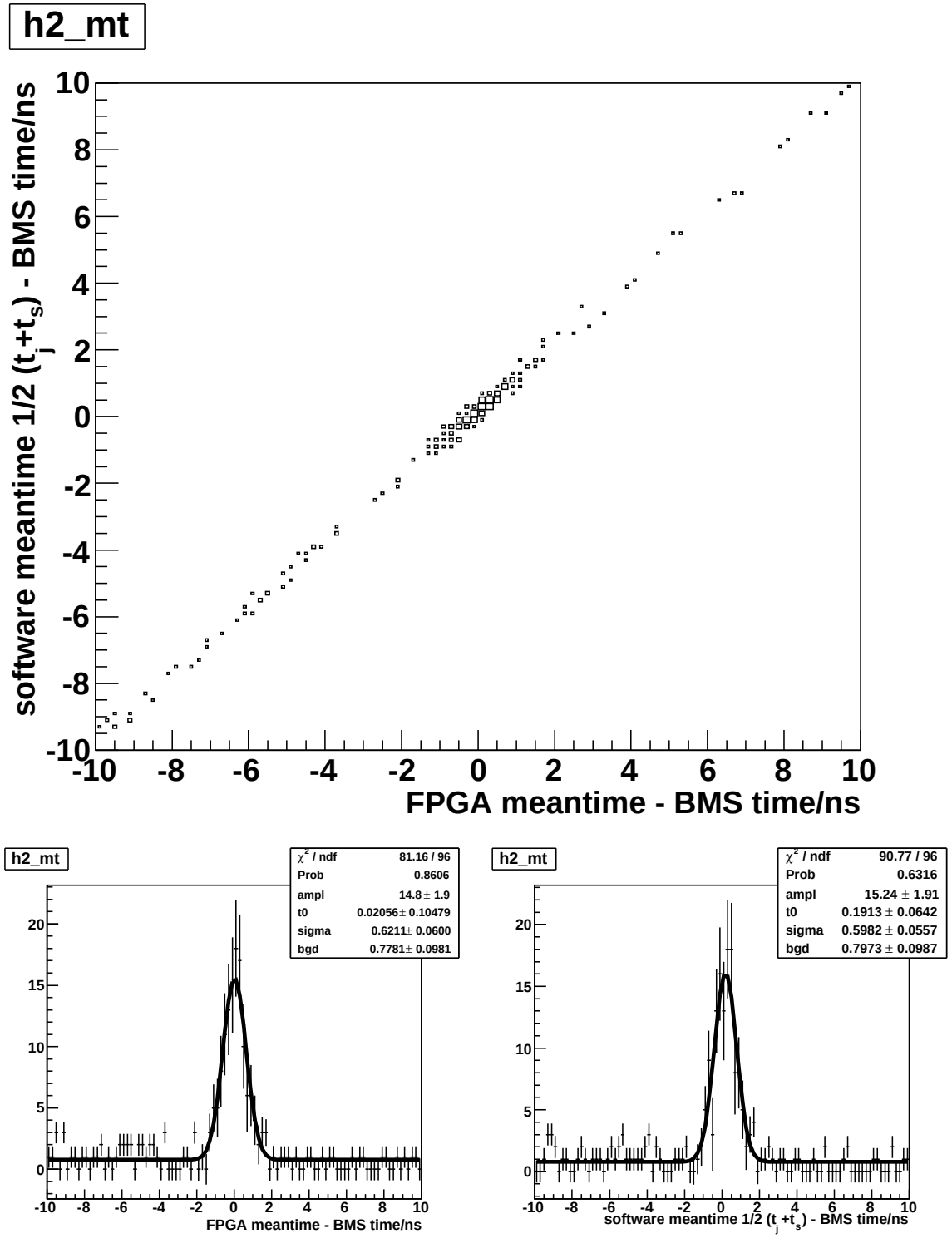


Abbildung 7.2: Aus dem oberen Diagramm ist ersichtlich, dass der FPGA-basierte Meantimer wie erwartet funktioniert: Die berechnete meantime stimmt mit der durch den FPGA bestimmten meantime überein. In den beiden unteren Diagrammen wird deutlich, dass das Sigma der durch den FPGA bestimmten meantime nur unwesentlich schlechter als die berechneten meantime ist, wobei die letztere z.B. von der Lichtausbreitung in den Szintillatorstreifen und den Anstiegszeiten der Photomultiplier abhängt.

7.3 Bestimmung des Jitters der ungetakteten FPGA-basierten Schaltung

Der in Kapitel *6.1.5* festgestellte Jitter des Ausgangssignals ist mit 600ps deutlich größer als erwartet. Es wird vermutet, dass er nicht nur durch das begrenzte Auflösungsvermögen der TDLs, sondern auch durch die lange ungetaktete Laufzeit innerhalb des FPGAs entsteht. Diese Vermutung wird nun durch eine wesentlich genauere Messung mit Hilfe der TDCs des COMPASS-Trigger-Systems überprüft. Dazu wird der Testsignal-Eingang der CFDs genutzt und dadurch ein extern angelegtes Testsignal parallel auf einen H1- und einen H2-Meantimer sowie auf die entsprechenden TDCs gegeben. Das MATRIX-OUT-Signal wird ebenfalls auf einen TDC gegeben, dient aber gleichzeitig als Triggersignal für die DAQ. Auf diese Weise kann die Testmessung analog zu einem normalen Run[3] über die DAQ erfasst werden. Aus den TDC-Daten der Testsignale und des MATRIX-OUT-Signals lässt sich anschließend offline der Jitter des Ausgangssignals bestimmen.

Bevor die eigentliche Testmessung durchgeführt wird, muss die Matrixschaltung analog zu der in Kapitel *6.5* beschriebenen Methode mit den externen Testsignalen zeitkalibriert werden. Für diese Testmessung wird nur ein GANDALF-Board verwendet.

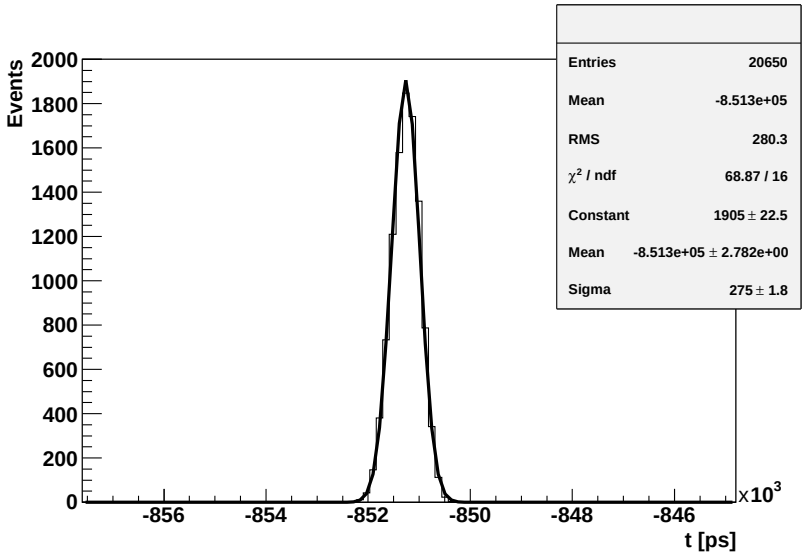


Abbildung 7.3: Differenz zwischen dem aus den TDC-Daten berechneten Koinzidenzzeitpunkt und dem gemessenen MATRIX-OUT-Signal. Die Laufzeit durch den FPGA beträgt ca. 100ns. Das Auflösungsvermögen der verwendeten TDCs beträgt 108ps, die Verbreiterung des Gaußpeaks durch die TDCs kann somit vernachlässigt werden.

Das Ergebnis der Testmessung ist in Abbildung *7.3* dargestellt. Da auf die linken und rechten Meantimereingänge die gleichen Testsignale gegeben werden, entstehen in den TDLs der Meantimer stets die gleichen Zentralereignisse. Der gemessene Jitter von MATRIX-OUT wird daher nur durch die Laufzeitschwankungen im FPGA und nicht durch das begrenzte Auflösungsvermögen der TDLs verursacht. Die Messung zeigt, dass die Laufzeit innerhalb des FPGAs im Mittel um ca. 280ps schwankt (RMS), wodurch das zeitliche Auflösungsvermögen der ungetakteten Schaltung wesentlich beeinflusst wird.

[3] Als Run wird eine zeitlich begrenzte Datennahme unter konstanten Bedingungen bezeichnet.

7.4 Zeitkalibrierung des LAS-Triggers

Im vorhergehenden Abschnitt wurde die Matrixschaltung mit Hilfe von externen Testsignalen kalibriert. Da die 192 Signalkabel von den Photomultiplieren bis zu den CFDs Laufzeitunterschiede von bis zu 3ns aufweisen (bei 60m Gesamtlänge) und die Photomultiplier verschiedene Anstiegszeiten besitzen, kann diese Kalibrierung nicht für das eigentliche Experiment genutzt werden. Einzig die durchgeführte Justierung der OUTPUT-Delays der drei NIM-Ausgänge kann übernommen werden.

In diesem Abschnitt wird erläutert, wie die gesamte Signalkette des LAS-Triggers inklusive aller Kabel und Komponenten mit Strahlmyonen kalibriert wird. Als Referenzpunkt für die Kalibrierung der Matrixschaltung dient dabei die sog. Triggerzeit. Sie ist zu t_{BMS} - dem Zeitpunkt, zu dem die Myonen die Beam-Momentum-Station passieren - um eine feste Zeitspanne verschoben. Die Ausgangssignale aller Trigger-Elemente des COMPASS-Trigger-Systems werden so kalibriert, dass sie sich zur Triggerzeit in der Trigger-Baracke treffen, wo dann durch eine logische Verknüpfung mit den Veto-Signalen das eigentliche Trigger-Signal für die DAQ erzeugt wird. Der LAS-Trigger ist demnach genau dann kalibriert, wenn das MATRIX-OUT-Signal für alle 1024 Matrix-Pixel gleichzeitig erfolgt und zur Triggerzeit in der Trigger-Baracke eintrifft.

Der LAS-Trigger besteht, wie bereits erwähnt, aus den beiden parallelen Hodoskopsystemen H1-H2S und H1-H2J, sodass auch zwei GANDALF-Boards bzw. zwei Matrixschaltungen kalibriert werden müssen. Die Zeitkalibrierung mit Strahlmyonen gestaltet sich folgendermaßen:

1. Da der dynamische Bereich der Meantimer mit 23ns kleiner als ursprünglich geplant ausfällt, werden zunächst die Laufzeitunterschiede in den Signalkabeln von den Photomultiplier bis zu den linken und rechten Eingängen der Meantimer durch die Auswertung der TDC-Daten eines aktuellen Runs überprüft. Eine eventuelle Links-Rechts-Asymmetrie muss korrigiert werden, damit der dynamische Bereich optimal ausgeleuchtet wird. Da die IODELAYs die Eingangssignale nur um maximal 5ns verzögern können und später primär für die Feinjustierung benötigt werden, müssen Unterschiede von mehr als 3ns durch zusätzliche Lemo-Kabel im Signalweg (zwischen Patchpanel und CFD-Eingang) ausgeglichen werden. Für einen H1-Kanal ist dies in Abbildung *7.4* dargestellt.

2. Neben der Links-Rechts-Asymmetrie der einzelnen Kanäle müssen in einem weiteren Schritt die relativen Unterschiede in den Signallaufzeiten von verschiedenen Kanälen untersucht werden. Ist diese Abweichung und damit der Offset zwischen den Meantimerausgängen im FPGA zu groß, reichen die SWITCH-Delays unter Umständen nicht mehr aus, um die Matrixschaltung zu kalibrieren. Größere Abweichungen werden daher über zusätzliche Kabel korrigiert[4]. Dazu werden für jeden Kanal aus den TDC-Daten der Myonereignisse die jeweiligen Ereigniszeitpunkte relativ zur Triggerzeit berechnet (meantime: $\frac{t_L+t_R}{2}$) und als Histogramm dargestellt. Da bei konstanter Strahlenergie die Flugzeit der am Target gestreuten Myonen von der BMS bis zu einem Streifen nur unwesentlich schwankt, wird in diesen Plots ein deutlicher Peak erwartet, der in

[4] Ein Meantimer wird als Ganzes verschoben, wenn in beiden Signalwegen, sowohl in den des linken als auch in den des rechten Meantimereingangs, zusätzliche Kabel eingefügt werden.

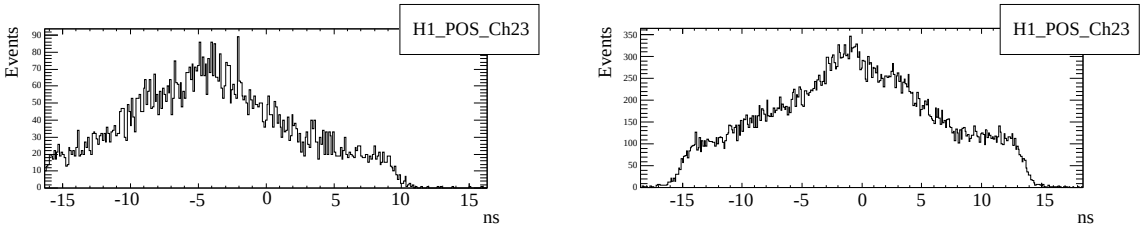


Abbildung 7.4: Die beiden Histogramme zeigen, wie sich die Links-Rechts-Asymmetrie von Kanal 23 von H1 mit Hilfe eines zusätzlichen 5ns-Kabels im Signalweg des linken Meantimereingangs korrigieren lässt. Auf der horizontalen Achse ist für jedes Myon aus den TDC-Daten $t_L - t_R$ aufgetragen, sodass sie auch als x-Position der Myonen innerhalb des Szintillatorstreifens interpretiert werden kann. Die Grafik zeigt außerdem, dass die maximale Differenz zwischen t_L und t_R nicht größer als 20ns ist und somit der in den Meantimern zur Verfügung stehende dynamische Bereich von 23ns ausreichend ist. Gleiches gilt für die Szintillatorstreifen von H2.

Abbildung *7.5* auch zu erkennen ist. Vergleicht man die Differenzen dieser Peaks gegenüber der Triggerzeit, ergeben sich die relativen Abweichungen zwischen den Szintillatorstreifen.

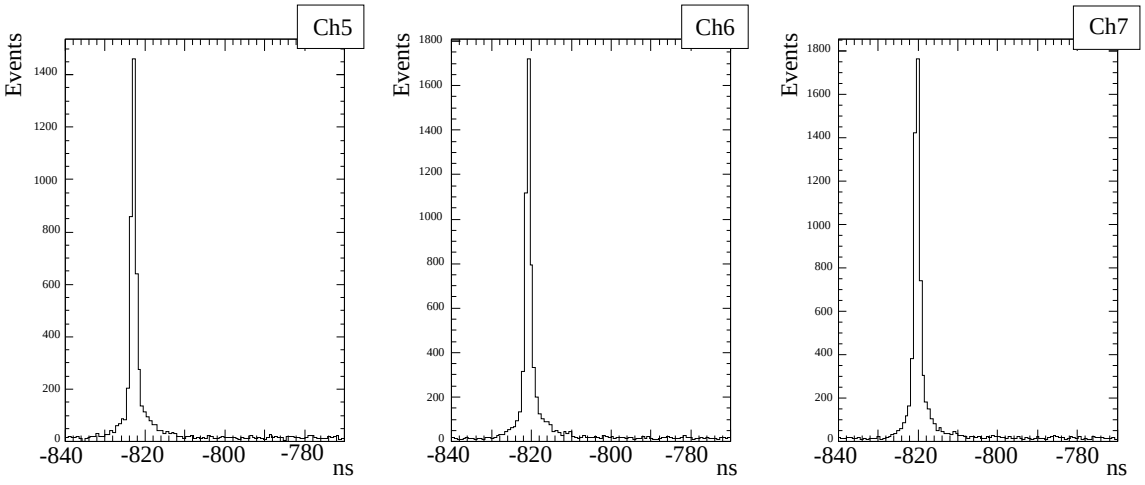


Abbildung 7.5: In dieser Abbildung sind auszugsweise die meantime-Histogramme für die Kanäle 5, 6 und 7 von H1 nach der Korrektur dargestellt. Auf der x-Achse ist die Differenz gegenüber der Triggerzeit aufgetragen. Die Laufzeitunterschiede in den verlegten Signalkabeln sind bei den drei dargestellten Szintillatorstreifen ausreichend gut korrigiert.

3. Als Nächstes erfolgt eine grobe Kalibrierung mit Hilfe des Oszilloskops. Dazu werden H1-OUT, H2-OUT und MATRIX-OUT auf dem Oszilloskop dargestellt, wobei das MATRIX-OUT-Signal als Trigger benutzt wird. Die CFDs sind so konfiguriert, dass die Signale der H1-Kanäle ca. 15ns und die der H2-Kanäle ca. 25ns lang sind. Im Unterschied zu den Testsignalkalibrierungen wird hier kein Leading-Edge-Alignment durchgeführt, sondern H1 mittig in H2 positioniert. Dadurch wird gewährleistet, dass analog zu der ReTiming-Schaltung im FPGA stets H1 für das Timing verantwortlich ist.

4. Der bereits bekannte Base-Pixel wird nun als einziger aktiviert und über die entsprechenden SWITCH-Delays der in Abbildung *7.6* dargestellte Zustand eingestellt. Der justierte H1-Kanal des Base-Pixels dient nun als Referenzsignal: Bei

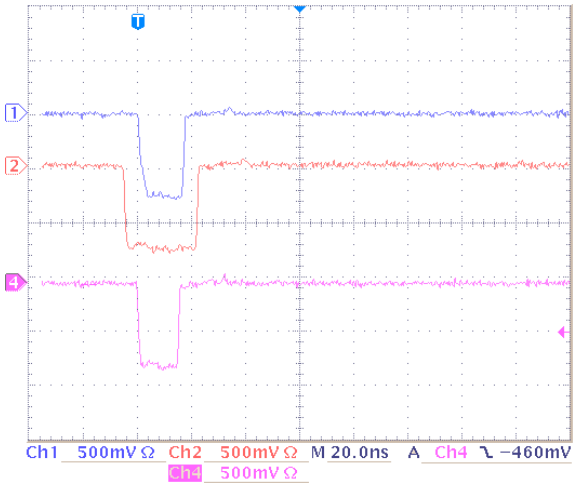


Abbildung 7.6: Justierung des Base-Pixels. Signal 1 ist das H1-OUT-Signal, Signal 2 das H2-OUT-Signal und Signal 4 das MATRIX-OUT-Signal.

allen anderen 31 Matrix-Pixeln in der gleichen Zeile werden die H2-Signale um dieses H1-Referenzsignal herum platziert. Analog dient anschließend der justierte H2-Kanal des Base-Pixels als Referenzsignal: Bei allen anderen 31 Matrix-Pixeln in der gleichen Spalte werden die H1-Signale mittig in dem H2-Referenzsignal positioniert. Konstruktionsbedingt ist nach der Justierung dieser 63 Matrix-Pixel die gesamte Matrixschaltung grob kalibriert.

5. Die feinstufige Kalibrierung wird nun mit Hilfe des sog. Matrix-Pixel-Timings durchgeführt. Dazu wird mit der grob kalibrierten Trigger-Elektronik ein neuer Run aufgenommen und in diesen TDC-Daten nach isolierten Pixel-Events gesucht, bei denen es in beiden Hodoskopen innerhalb einer bestimmten Zeitspanne in genau einem Szintillatorstreifen ein Event gab. Das MATRIX-OUT-Signal kann dann genau einem Matrix-Pixel zugeordnet und die Differenz zur Triggerzeit bestimmt werden. So ergibt sich für jeden Matrix-Pixel ein Histogramm, aus dem bei ausreichend guter Statistik die mittlere Abweichung zur Triggerzeit bestimmt werden kann. Diese Abweichungen können anschließend über die feinstufigen IODELAYs korrigiert werden. In Abbildung *7.7* ist ein solches Matrix-Pixel-Timing dargestellt.

Zum Abschluss der Kalibrierung wird ein Run mit dem LAS-Trigger durchgeführt und aus den TDC-Daten das sog. Trigger-Timing bestimmt. Dazu wird in einem Histogramm die Differenz des LAS-Trigger-Signals (MATRIX-OUT) gegenüber der Triggerzeit abgetragen. Der Trigger-Peak ist in Abbildung *7.8* zu erkennen, die Standardabweichung seines Gaußpeaks beträgt ca. 0.9ns. Um diesen Wert einordnen zu können, wird zum Vergleich der ähnlich dimensionierte OUTER-Trigger herangezogen, dessen Standardabweichung 1.3ns beträgt. Dieser verwendet die vom IPN Orsay[5] entwickelten Meantimer-Module und den im Physikalischen Institut der Universität Bonn entwickelten Matrix-Chip. Die Standardabweichung ist zum Einen ein Maß für die Güte der Kalibrierung und zum Anderen eine obere Grenze des zeitlichen Auflösungsvermögens des gesamten Systems.

[5] Institut de Physique Nucleaire d'Orsay

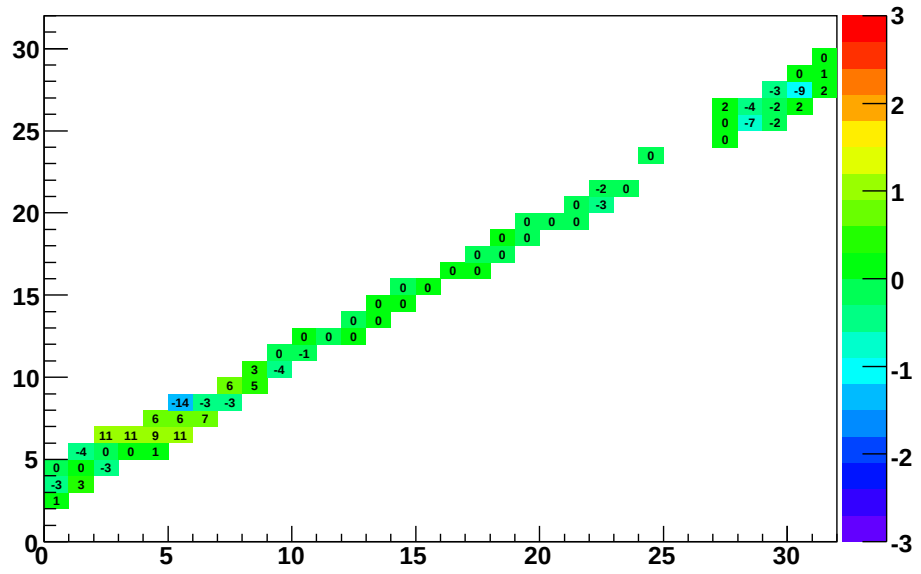


Abbildung 7.7: Auf der horizontalen Achse des Matrix-Pixel-Timings sind die H2-Meantimer und auf der vertikalen Achse die H1-Meantimer aufgetragen. Der zentrale Bereich ist hier bereits optimal kalibriert. Die Zahlen in den Matrix-Pixeln stellen die Abweichungen von der Triggerzeit ×100ps dar. Während des Matrix-Pixel-Timings war bereits die in Abbildung *6.21* gezeigte diagonale Koinzidenzmatrix aktiv, sodass nur für die relevanten Matrix-Pixel die Abweichungen zur Triggerzeit bestimmt werden. Damit ist auch gezeigt, dass die Matrixschaltung wie erwartet funktioniert. Bei den auf der Diagonale fehlenden Kanälen funktionierte einer der Photomultiplier nicht.

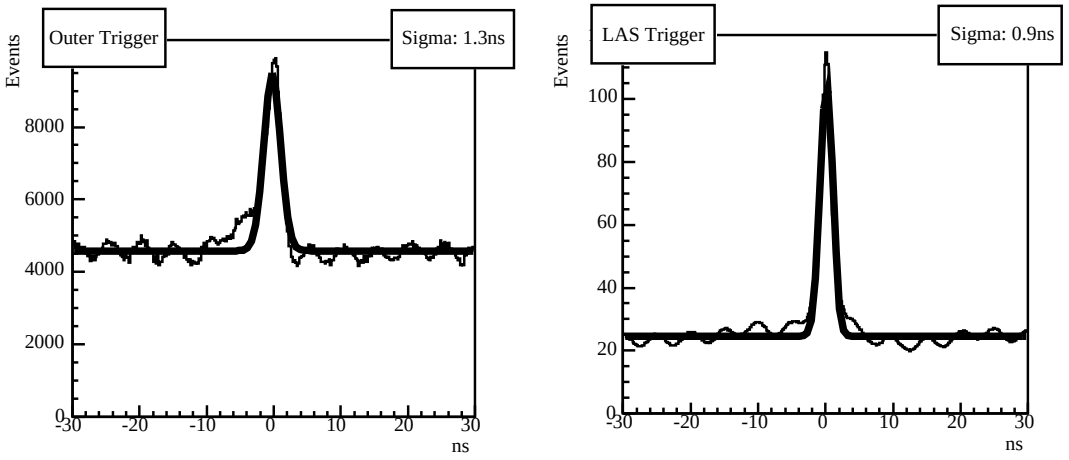


Abbildung 7.8: Die Standardabweichung des Trigger-Timings des kalibrierten LAS-Triggers bewegt sich mit 900ps in einem ähnlichen Bereich wie bei dem vergleichbaren, ähnlich dimensionierten OUTER-Trigger. Dieser Trigger wird schon seit mehreren Jahren am COMPASS-Experiment eingesetzt und basiert auf den Orsay-Meantimer-Modulen und auf dem in Bonn entwickelten Matrix-Chip. Die erkennbaren Schwankungen im Untergrund sind durch den Strahl bedingt.

8. Zusammenfassung und Ausblick

Für den zur Strahlzeit 2010 neu aufgebauten LAS-Trigger des COMPASS-Experiments, sollten im Rahmen dieser Diplomarbeit folgende Komponenten auf einem Virtex 5, einem FPGA der Firma Xilinx, umgesetzt werden:

- Insgesamt 64 identische, als ungetaktete Tapped Delay Line-Schaltungen ausgeführte Meantimer, mit einem dynamischen Bereich von 30ns.
- Eine ungetaktete 32×32 Koinzidenzschaltung, bei der alle 1024 Matrix-Pixel individuell aktiviert werden können.

Bis auf den dynamischen Bereich der Meantimer, der mit 23ns etwas kürzer als gefordert ausfällt, konnte diese Zielsetzung vollständig erfüllt werden. Die im vorangegangenen Kapitel diskutierten Messungen zeigen, dass der zur Verfügung stehende dynamische Bereich für den Einsatz im LAS-Trigger ausreichend ist und sowohl die Meantimer als auch die Koinzidenzschaltung gemäß den Vorgaben funktionieren. Die auf dem FPGA implementierten Komponenten besitzen ähnliche Eigenschaften wie die bis dato eingesetzten Matrix-Chips des Physikalischen Instituts der Universität Bonn und Meantimer-Module der IPN Orsay.

Darüber hinaus wurde zur Konfiguration des Gesamtsystems eine VME-Schnittstelle und ein Webinterface entwickelt.

Bei der Implementierung der ungetakteten Schaltungen wurde deutlich, dass innerhalb des FPGAs erwartungsgemäß eine sehr hohe Zeitauflösung erreicht wird, diese aber durch Laufzeitschwankungen nicht mehr an den Output-Pins beobachtet werden kann. Je länger das Signal durch den FPGA läuft, desto größer wird dieser Output-Jitter.

Als Ausblick wird daher ein Konzept vorgeschlagen, mit dem sich das zeitliche Auflösungsvermögen der FPGA-basierten Trigger-Elektronik durch Reduktion dieses Output-Jitters noch weiter verbessern lassen sollte. Durch die Kombination von getakteten und ungetakteten Elementen sollte es möglich sein, sowohl eine hohe Zeitauflösung als auch einen geringen Output-Jitter zu erreichen.

Bei der in Abbildung *8.1* dargestellten Schaltung wird das Eingangssignal direkt nach dem Eintritt in den FPGA in eine Carry-Chain gegeben, wobei das Signal nach jedem Carry-Step auch auf einen FlipFlop geführt wird. Alle diese FlipFlops werden durch den gleichen Takt angesteuert, sodass aus diesem Input-Bit-Pattern (IBP) die Differenz des Eingangssignals zum Takt ersichtlich ist. Diese Information wird für das Eingangssignal gespeichert. Die eigentlichen Schaltungen, z.B. die Meantimer- oder die Koinzidenzschaltung werden getaktet ausgeführt. Bevor das Signal über den Output-Pin den FPGA wieder verlässt, wird die bekannte Differenz zum Takt wieder hergestellt. Dazu wird ein Demultiplexer eingesetzt, der in Abhängigkeit des IBPs das Signal auf eine weitere Carry-Chain gibt.

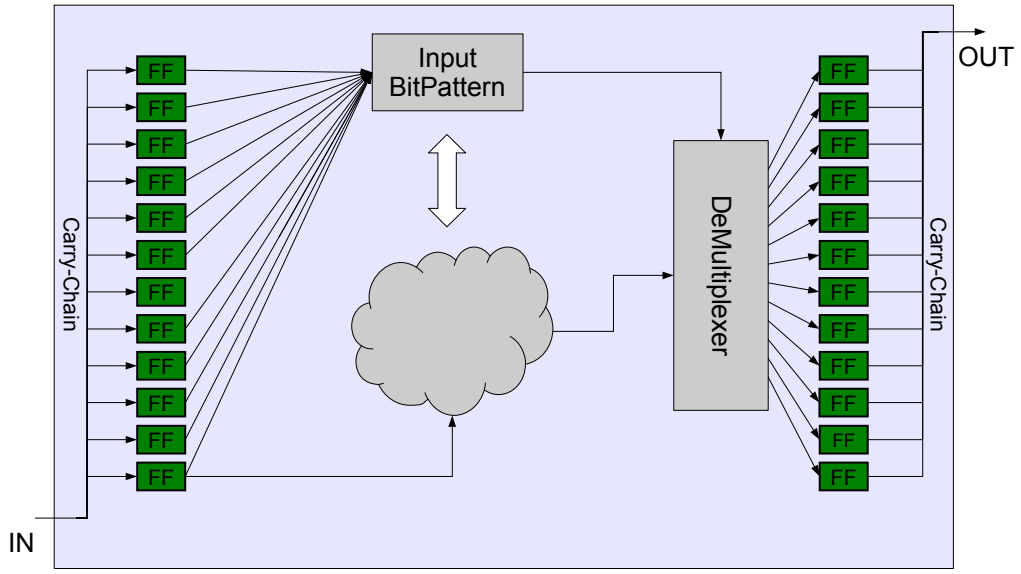


Abbildung 8.1: Eine schematische Darstellung einer taktstabilisierten Schaltung.

Gegenüber der aktuell implementierten Trigger-Elektronik würden sich damit folgende Vorteile ergeben:

- Die Zeitauflösung der Meantimer wird durch die Größe eines Carry-Chain-Steps bestimmt (ca. 100ps), da die meantime berechnet wird und die Anpassung des Ausgangssignals über die Manipulation des IBPs erfolgt.
- Die ungetaktete Laufzeit innerhalb des FPGAs ist auf ein Minimum reduziert.
- Um Eingangssignale zu verzögern, können nun einfach weitere Takt-Zyklen hinzugefügt bzw. das IBP manipuliert werden. Auf diese Weise kann jitterfrei und quasi unbegrenzt verzögert werden. Die obere Grenze der Verzögerung wird nur durch die Größe des eingesetzten Zählers begrenzt.

Es wird daher empfohlen, dieses Konzept zum Gegenstand weiterer Untersuchungen zu machen.

A. Verilog Quellcode des FPGA-Designs

Der Quellcode ist in den folgenden vier Dateien organisiert:

- `fpga_top.v`: Basisstruktur des Designs, vergleichbar mit dem Hauptprogramm bei der Entwicklung von Software.
- `parts.v`: Definiert diverse Hilfs-Module.
- `meanTimer.v`: Definiert die 64 Meantimer-Module
- `matrix.v`: Das Matrix-Modul.

Diese Dateien befinden sich auch auf der beigefügten Daten-CD.

Listing A.1: fpga_top.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company: University of Bonn
4  // Engineer: John Bieling
5  //
6  // Design Name:  Mean-Timer
7  // Module Name: fpga_top
8  // Project Name: COMPASS
9  //
10 //////////////////////////////////////////////////////////////////////////////////
11
12 module fpga_top( CONN_LN, CONN_LP, CONN_RN, CONN_RP,
13                   CONN_OA1, CONN_OA2, CONN_OB1, CONN_OB2,
14                   CONN_IA, CONN_IB,
15                   CLK_40MHZ_VDSP,
16                   VA_Write, VA_Strobe, VA_Ready, VA_Control, VA_uBlaze,
17                   VA_FifoFull, VA_FifoEmpty, VA_Reset, VD);
18
19 //how many MTs are on the board?
20 localparam count = 64;
21
22 input wire [count-1:0] CONN_LN;
23 input wire [count-1:0] CONN_LP;
24 input wire [count-1:0] CONN_RN;
25 input wire [count-1:0] CONN_RP;
26
27 input wire VA_Write;
28 input wire VA_Strobe;
29 output wire VA_Ready;
30 input wire VA_Control;
31 input wire VA_uBlaze;
32 output wire VA_FifoFull;
33 output wire VA_FifoEmpty;
34 input wire VA_Reset;
35 inout wire [31:0] VD;
36 input wire CLK_40MHZ_VDSP;
37
38 input wire CONN_IA;
39 input wire CONN_IB;
40
41 output wire LED_RED;
42 output wire LED_GREEN;
43
44 output wire CONN_OA1;
45 output wire CONN_OA2;
46 output wire CONN_OB1;
47 output wire CONN_OB2;
48
49 //locals
50 wire [count-1:0] CONN_L;
51 wire [count-1:0] CONN_L_Delay;
52 wire [count-1:0] CONN_L_eDelay;
53 wire [count-1:0] CONN_R;
54 wire [count-1:0] CONN_R_Delay;
55 wire [count-1:0] CONN_R_eDelay;
56 wire [count-1:0] CONN_O;
57 wire [31:0] CONN_H1;
58 wire [31:0] CONN_H2;
59
60 wire config_mem_BRAM_Rst;
61 wire config_mem_BRAM_Clk;
62 reg config_mem_BRAM_EN;
63
64 wire [3:0] config_mem_BRAM_WEN;
65 wire [15:0] config_mem_BRAM_Addr;
66 wire [31:0] config_mem_BRAM_Din;
67 wire [31:0] config_mem_BRAM_Dout;
68
69 wire CLK_40MHZ_OUT;
70 wire CLK_120MHZ_OUT;
71 wire CLK_200MHZ_OUT;
72
```

```
73 wire [255:0] FastRegister;
74
75 assign LED_RED = 0;
76 assign LED_GREEN = 0;
77
78 //initialize CPLD-Interface for VME-communication
79 cpld_if #( .GEN_PERFORM_SIMU(0), .GEN_LATENCY(300), .GEN_FRAMEWIDTH(1023),
80                     .GEN_RDM(4'b0010), .GEN_DSP_FIRMW_VERS(29012010))
81
82 cpld_if_1 ( .D (VD), .f_Write(VA_Write), .f_Strobe(VA_Strobe), .f_Ready (VA_Ready),
83                     .f_Control (VA_Control), .f_uBlaze (VA_uBlaze), .f_FifoFull(VA_FifoFull),
84                     .f_FifoEmpty(VA_FifoEmpty), .f_Reset(1'b0),
85                     .CLK_40MHZ_VDSP(CLK_40MHZ_VDSP),
86                     .CLK_40MHZ_OUT (CLK_40MHZ_OUT),
87                     .nCLK_40MHZ_OUT(),
88                     .CLK_120MHZ_OUT(CLK_120MHZ_OUT),
89                     .RST_Startup_1_OUT(), .RST_Startup_2_OUT(), .RST_Startup_3_OUT(),
90                     .SLINK_init_done(1'b1),
91                     .Spy_Din(), .Spy_CLK(), .Spy_WR(), .Spy_RST(), .Spy_Full(), .Spy_Almost_Full(),
92                     .config_mem_BRAM_Rst(1'b0),
93                     .config_mem_BRAM_Clk(CLK_40MHZ_OUT),
94                     .config_mem_BRAM_EN(config_mem_BRAM_EN),
95                     .config_mem_BRAM_WEN(config_mem_BRAM_WEN),
96                     .config_mem_BRAM_Addr(config_mem_BRAM_Addr),
97                     .config_mem_BRAM_Din(config_mem_BRAM_Din),
98                     .config_mem_BRAM_Dout(config_mem_BRAM_Dout),
99                     .FastRegister(FastRegister));
100
101 //store the matrix-pixel information
102 reg [31:0] MatrixEnReg00 = 0;
103 reg [31:0] MatrixEnReg01 = 0;
104 reg [31:0] MatrixEnReg02 = 0;
105 reg [31:0] MatrixEnReg03 = 0;
106 reg [31:0] MatrixEnReg04 = 0;
107 reg [31:0] MatrixEnReg05 = 0;
108 reg [31:0] MatrixEnReg06 = 0;
109 reg [31:0] MatrixEnReg07 = 0;
110 reg [31:0] MatrixEnReg08 = 0;
111 reg [31:0] MatrixEnReg09 = 0;
112 reg [31:0] MatrixEnReg10 = 0;
113 reg [31:0] MatrixEnReg11 = 0;
114 reg [31:0] MatrixEnReg12 = 0;
115 reg [31:0] MatrixEnReg13 = 0;
116 reg [31:0] MatrixEnReg14 = 0;
117 reg [31:0] MatrixEnReg15 = 0;
118 reg [31:0] MatrixEnReg16 = 0;
119 reg [31:0] MatrixEnReg17 = 0;
120 reg [31:0] MatrixEnReg18 = 0;
121 reg [31:0] MatrixEnReg19 = 0;
122 reg [31:0] MatrixEnReg20 = 0;
123 reg [31:0] MatrixEnReg21 = 0;
124 reg [31:0] MatrixEnReg22 = 0;
125 reg [31:0] MatrixEnReg23 = 0;
126 reg [31:0] MatrixEnReg24 = 0;
127 reg [31:0] MatrixEnReg25 = 0;
128 reg [31:0] MatrixEnReg26 = 0;
129 reg [31:0] MatrixEnReg27 = 0;
130 reg [31:0] MatrixEnReg28 = 0;
131 reg [31:0] MatrixEnReg29 = 0;
132 reg [31:0] MatrixEnReg30 = 0;
133 reg [31:0] MatrixEnReg31 = 0;
134
135 //store which ANDs, CAPs and MTs are supposed to be on - all active low
136 reg [63:0] MTEnabled = 0;
137 reg [63:0] AndEnabled_0 = 0;
138 reg [1:0] CapsEnabled_0 = 0;
139
140 //stores information, if the real data or the generated signal should be layed onto the meantimer
141 reg input_choice_1;
142
143 //if choice_1 puts is set to generated, do we want to on all MT-L-R-Inputs the same signals
144 //or do we want to big delays to simulate szintillator-signals at different x-positions
145 //at the same time (done by webinterface)
146 reg input_choice_2;
```

```
147
148 //select signal for each output pin
149 reg [5:0] output_choice_OA1;
150 reg [5:0] output_choice_OA2;
151 reg [5:0] output_choice_OB1;
152 reg [5:0] output_choice_OB2;
153
154 //signals to pass the enable-information of each AND from one MT to the next one
155 wire [53:0] AndEnabled_1;
156 wire [53:0] AndEnabled_2;
157 wire [53:0] AndEnabled_3;
158 wire [53:0] AndEnabled_4;
159 wire [53:0] AndEnabled_5;
160 wire [53:0] AndEnabled_6;
161 wire [53:0] AndEnabled_7;
162 wire [53:0] AndEnabled_8;
163 wire [53:0] AndEnabled_9;
164 wire [53:0] AndEnabled_10;
165 wire [53:0] AndEnabled_11;
166 wire [53:0] AndEnabled_12;
167 wire [53:0] AndEnabled_13;
168 wire [53:0] AndEnabled_14;
169 wire [53:0] AndEnabled_15;
170 wire [53:0] AndEnabled_16;
171 wire [53:0] AndEnabled_17;
172 wire [53:0] AndEnabled_18;
173 wire [53:0] AndEnabled_19;
174 wire [53:0] AndEnabled_20;
175 wire [53:0] AndEnabled_21;
176 wire [53:0] AndEnabled_22;
177 wire [53:0] AndEnabled_23;
178 wire [53:0] AndEnabled_24;
179 wire [53:0] AndEnabled_25;
180 wire [53:0] AndEnabled_26;
181 wire [53:0] AndEnabled_27;
182 wire [53:0] AndEnabled_28;
183 wire [53:0] AndEnabled_29;
184 wire [53:0] AndEnabled_30;
185 wire [53:0] AndEnabled_31;
186 wire [53:0] AndEnabled_32;
187 wire [53:0] AndEnabled_33;
188 wire [53:0] AndEnabled_34;
189 wire [53:0] AndEnabled_35;
190 wire [53:0] AndEnabled_36;
191 wire [53:0] AndEnabled_37;
192 wire [53:0] AndEnabled_38;
193 wire [53:0] AndEnabled_39;
194 wire [53:0] AndEnabled_40;
195 wire [53:0] AndEnabled_41;
196 wire [53:0] AndEnabled_42;
197 wire [53:0] AndEnabled_43;
198 wire [53:0] AndEnabled_44;
199 wire [53:0] AndEnabled_45;
200 wire [53:0] AndEnabled_46;
201 wire [53:0] AndEnabled_47;
202 wire [53:0] AndEnabled_48;
203 wire [53:0] AndEnabled_49;
204 wire [53:0] AndEnabled_50;
205 wire [53:0] AndEnabled_51;
206 wire [53:0] AndEnabled_52;
207 wire [53:0] AndEnabled_53;
208 wire [53:0] AndEnabled_54;
209 wire [53:0] AndEnabled_55;
210 wire [53:0] AndEnabled_56;
211 wire [53:0] AndEnabled_57;
212 wire [53:0] AndEnabled_58;
213 wire [53:0] AndEnabled_59;
214 wire [53:0] AndEnabled_60;
215 wire [53:0] AndEnabled_61;
216 wire [53:0] AndEnabled_62;
217 wire [53:0] AndEnabled_63;
218
219 //signals to pass the enable-information of both CAPs from one MT to the next one
220 wire [1:0] CapsEnabled_1;
```

```
221 wire [1:0] CapsEnabled_2;
222 wire [1:0] CapsEnabled_3;
223 wire [1:0] CapsEnabled_4;
224 wire [1:0] CapsEnabled_5;
225 wire [1:0] CapsEnabled_6;
226 wire [1:0] CapsEnabled_7;
227 wire [1:0] CapsEnabled_8;
228 wire [1:0] CapsEnabled_9;
229 wire [1:0] CapsEnabled_10;
230 wire [1:0] CapsEnabled_11;
231 wire [1:0] CapsEnabled_12;
232 wire [1:0] CapsEnabled_13;
233 wire [1:0] CapsEnabled_14;
234 wire [1:0] CapsEnabled_15;
235 wire [1:0] CapsEnabled_16;
236 wire [1:0] CapsEnabled_17;
237 wire [1:0] CapsEnabled_18;
238 wire [1:0] CapsEnabled_19;
239 wire [1:0] CapsEnabled_20;
240 wire [1:0] CapsEnabled_21;
241 wire [1:0] CapsEnabled_22;
242 wire [1:0] CapsEnabled_23;
243 wire [1:0] CapsEnabled_24;
244 wire [1:0] CapsEnabled_25;
245 wire [1:0] CapsEnabled_26;
246 wire [1:0] CapsEnabled_27;
247 wire [1:0] CapsEnabled_28;
248 wire [1:0] CapsEnabled_29;
249 wire [1:0] CapsEnabled_30;
250 wire [1:0] CapsEnabled_31;
251 wire [1:0] CapsEnabled_32;
252 wire [1:0] CapsEnabled_33;
253 wire [1:0] CapsEnabled_34;
254 wire [1:0] CapsEnabled_35;
255 wire [1:0] CapsEnabled_36;
256 wire [1:0] CapsEnabled_37;
257 wire [1:0] CapsEnabled_38;
258 wire [1:0] CapsEnabled_39;
259 wire [1:0] CapsEnabled_40;
260 wire [1:0] CapsEnabled_41;
261 wire [1:0] CapsEnabled_42;
262 wire [1:0] CapsEnabled_43;
263 wire [1:0] CapsEnabled_44;
264 wire [1:0] CapsEnabled_45;
265 wire [1:0] CapsEnabled_46;
266 wire [1:0] CapsEnabled_47;
267 wire [1:0] CapsEnabled_48;
268 wire [1:0] CapsEnabled_49;
269 wire [1:0] CapsEnabled_50;
270 wire [1:0] CapsEnabled_51;
271 wire [1:0] CapsEnabled_52;
272 wire [1:0] CapsEnabled_53;
273 wire [1:0] CapsEnabled_54;
274 wire [1:0] CapsEnabled_55;
275 wire [1:0] CapsEnabled_56;
276 wire [1:0] CapsEnabled_57;
277 wire [1:0] CapsEnabled_58;
278 wire [1:0] CapsEnabled_59;
279 wire [1:0] CapsEnabled_60;
280 wire [1:0] CapsEnabled_61;
281 wire [1:0] CapsEnabled_62;
282 wire [1:0] CapsEnabled_63;
283
284 //Finite-State-Machine stuff to update local configuration after VME update
285 localparam idle   = 3'b000; // idle state
286 localparam dec_addr  = 3'b010;  // dec addres
287 localparam get_data  = 3'b011; // read Data state
288 localparam inc_prep = 3'b100; // prepare inc delay
289 localparam inc_delay  = 3'b101; // inc delay elements
290
291 localparam mt_delay  = 4'b1000; // 64x32Bit delay information block
292 localparam mt_data  = 4'b0100; // 64x32Bit data block
293 localparam mt_3  = 4'b0010; // 64x32Bit unused
294 localparam mt_4   = 4'b0001; // 64x32Bit unused
```

```
295
296 reg [7:0] current_addr = 0;
297 reg [2:0] current_state = idle;
298 reg [63:0] current_sreg = 0;
299 reg [3:0] current_mtype;
300
301 reg [7:0] next_addr = 0;
302 reg [2:0] next_state = idle;
303 reg [63:0] next_sreg = 0;
304 reg [3:0] next_mtype;
305 reg next_en = 0;
306
307 //delay stuff
308 wire [count-1:0] inc_L;
309 wire [count-1:0] inc_R;
310 wire [count-1:0] inc_M;
311 wire [count-1:0] inc_O;
312 reg [31:0] BRAM_delayinfo = 0;
313 reg [31:0] current_delayinfo = 0;
314 reg [31:0] next_delayinfo = 0;
315 reg delay_rst = 0;
316
317 //the read 32bit (current_delayinfo) for each MT contain 4x8Bit with 6Bit used as a tap-count
318 //each tab-count-information gets reduced and while they are not 0, the corresponding
319 //inc-signal must be high
320 //by simply ORing the 6bits together, the high-signal/inc-signal is obtained
321 //the select_MT-signal is a one_hot register indicating the MT or simply the adress as a 6bit "number"
322 //DeMux1to64 (select_MT,in,out);
323 DeMux1to64 DeMux1to64_M (current_addr[5:0],|(current_delayinfo[5:0]),inc_M);
324 DeMux1to64 DeMux1to64_R (current_addr[5:0],|(current_delayinfo[13:8]),inc_R);
325 DeMux1to64 DeMux1to64_O (current_addr[5:0],|(current_delayinfo[21:16]),inc_O);
326 DeMux1to64 DeMux1to64_L (current_addr[5:0],|(current_delayinfo[29:24]),inc_L);
327
328 //RAM-mapping of the CPLD-interface (Freiburg)
329 //Base = 1000000000011111
330 //A_min = 1100000000 = C00 in VME
331 //A_max = 1100011111
332
333 //additional RAM-mapping
334 //Base = 1110000000011111
335 //A_min = 00000000 = C00 in VME
336 //A_max = 00011111
337
338 //config_mem_BRAM_Addr=16'b1110000000011111;
339 //address-mapping, address is ^^^^^^^^
340 assign config_mem_BRAM_Addr = {3'b111,current_addr,5'b11111};
341
342
343 //auf der posedge werden nur die next-Werte aktualisiert, hier
344 //werden daher keine Veränderungen von Dout erzeugt
345 always@(posedge CLK_40MHZ_OUT)
346 begin
347
348         next_state <= idle;
349         next_en <= 0;
350         delay_rst <= 0;
351         next_delayinfo <= 0;
352         next_addr <= current_addr;
353         next_sreg <= current_sreg;
354         next_mtype <= current_mtype;
355
356         case (current_state)
357
358                 idle : if (FastRegister[16])
359                         begin
360                                 next_addr <= 255; //103; //max:255
361                                 // we use the lower memmory and count down
362                                 //11_111111 64x32Bit unused
363                                 //10_111111 64x32Bit unused
364                                 //01_111111 64x32Bit (40 needed) DataBlock
365                                 //00_111111 64x32Bit Delay Information
366                                 next_state <= get_data;
367                                 next_sreg <= 1;
368                                 next_en <= 1;
```

```
369                                     delay_rst <= 1;  //Set all delays to 0 in the first idle part only
370                                     next_mtype <= mt_4; //read out the highest block first and go down
371                             end
372
373                     inc_prep : begin
374                                     next_state <= inc_delay;
375                                     next_delayinfo <= BRAM_delayinfo;
376                             end
377
378                     inc_delay : if (current_delayinfo == 0) next_state <= dec_addr;
379                             else begin
380                                     //reduce all4 tabs, but not below 0
381                                     if (current_delayinfo[5:0] != 0) next_delayinfo[5:0] <= current_delayinfo[5:0] - 1;
382                                     if (current_delayinfo[13:8] != 0) next_delayinfo[13:8] <= current_delayinfo[13:8] - 1;
383                                     if (current_delayinfo[21:16] != 0) next_delayinfo[21:16] <= current_delayinfo[21:16] - 1;
384                                     if (current_delayinfo[29:24] != 0) next_delayinfo[29:24] <= current_delayinfo[29:24] - 1;
385                                     next_state <= inc_delay;
386                             end
387
388                     get_data : case (current_mtype)
389                                     mt_delay : next_state <= inc_prep;
390                                     default : next_state <= dec_addr;
391                             endcase
392
393                     dec_addr : if (current_addr != 0) //auto-switch to idle if done
394                             begin
395                                     //if the last addr read has 0 in the lowest 6Bits, its a memoryychange
396                                     if (current_addr[5:0] == 0)
397                                     begin
398                                             next_mtype <= {current_mtype[2:0],1'b0};
399                                             next_sreg <= 1; //each memoryblock has its own sreg
400                                     end
401                                     else begin
402                                             next_mtype <= current_mtype;
403                                             next_sreg <= {current_sreg[62:0],1'b0};
404                                     end
405
406                                     next_state <= get_data;
407                                     //The next two statements might be better placed inside an extra
408                                     //always, at least Gabriel is saying so...
409                                     next_addr <= current_addr - 1;
410                                     next_en <= 1;
411                             end
412
413             endcase
414     end
415
416
417
418     //Mit der negativen Flanke werden diese Latches aktualisert
419     //kurz danach liegt die richtige Adresse auf dem DRAM
420     //der EN ist hight (falls so gewählt)
421     //der Ausgang von Dout wird außerdem auf das richtige Register gelegt
422     //Mit der nächten posedge wird das Dout Register aktualisiert (wenn EN high)
423     //Wir können die Daten also erst NACH der nächsten Posedge auslesen
424     always@(negedge CLK_40MHZ_OUT)
425     begin
426                     current_state = next_state;
427                     current_addr = next_addr;
428                     current_sreg = next_sreg;
429                     config_mem_BRAM_EN = next_en;
430                     current_delayinfo = next_delayinfo;
431                     current_mtype = next_mtype;
432                     current_delayinfo = next_delayinfo;
433     end
434
435
436
437     //Die negedge vorher hat auf den BRAM alle Signale angelegt
438     //Die nächste Posedge hat dann das Dout aktualisiert
439     //jetzt können wir die Daten wegschreiben
440     always@(negedge CLK_40MHZ_OUT)
441     begin
442
```

```
443     if (current_state == get_data)
444         case (current_mtype)
445             mt_delay : BRAM_delayinfo = config_mem_BRAM_Dout;
446             mt_data : case(current_sreg)
447                                 64'h8000000000000000: MatrixEnReg00 = config_mem_BRAM_Dout;
448                                 64'h4000000000000000: MatrixEnReg01 = config_mem_BRAM_Dout;
449                                 64'h2000000000000000: MatrixEnReg02 = config_mem_BRAM_Dout;
450                                 64'h1000000000000000: MatrixEnReg03 = config_mem_BRAM_Dout;
451                                 64'h0800000000000000: MatrixEnReg04 = config_mem_BRAM_Dout;
452                                 64'h0400000000000000: MatrixEnReg05 = config_mem_BRAM_Dout;
453                                 64'h0200000000000000: MatrixEnReg06 = config_mem_BRAM_Dout;
454                                 64'h0100000000000000: MatrixEnReg07 = config_mem_BRAM_Dout;
455                                 64'h0080000000000000: MatrixEnReg08 = config_mem_BRAM_Dout;
456                                 64'h0040000000000000: MatrixEnReg09 = config_mem_BRAM_Dout;
457                                 64'h0020000000000000: MatrixEnReg10 = config_mem_BRAM_Dout;
458                                 64'h0010000000000000: MatrixEnReg11 = config_mem_BRAM_Dout;
459                                 64'h0008000000000000: MatrixEnReg12 = config_mem_BRAM_Dout;
460                                 64'h0004000000000000: MatrixEnReg13 = config_mem_BRAM_Dout;
461                                 64'h0002000000000000: MatrixEnReg14 = config_mem_BRAM_Dout;
462                                 64'h0001000000000000: MatrixEnReg15 = config_mem_BRAM_Dout;
463                                 64'h0000800000000000: MatrixEnReg16 = config_mem_BRAM_Dout;
464                                 64'h0000400000000000: MatrixEnReg17 = config_mem_BRAM_Dout;
465                                 64'h0000200000000000: MatrixEnReg18 = config_mem_BRAM_Dout;
466                                 64'h0000100000000000: MatrixEnReg19 = config_mem_BRAM_Dout;
467                                 64'h0000080000000000: MatrixEnReg20 = config_mem_BRAM_Dout;
468                                 64'h0000040000000000: MatrixEnReg21 = config_mem_BRAM_Dout;
469                                 64'h0000020000000000: MatrixEnReg22 = config_mem_BRAM_Dout;
470                                 64'h0000010000000000: MatrixEnReg23 = config_mem_BRAM_Dout;
471                                 64'h0000008000000000: MatrixEnReg24 = config_mem_BRAM_Dout;
472                                 64'h0000004000000000: MatrixEnReg25 = config_mem_BRAM_Dout;
473                                 64'h0000002000000000: MatrixEnReg26 = config_mem_BRAM_Dout;
474                                 64'h0000001000000000: MatrixEnReg27 = config_mem_BRAM_Dout;
475                                 64'h0000000800000000: MatrixEnReg28 = config_mem_BRAM_Dout;
476                                 64'h0000000400000000: MatrixEnReg29 = config_mem_BRAM_Dout;
477                                 64'h0000000200000000: MatrixEnReg30 = config_mem_BRAM_Dout;
478                                 64'h0000000100000000: MatrixEnReg31 = config_mem_BRAM_Dout;
479                                 64'h0000000080000000: AndEnabled_0[31:0] = config_mem_BRAM_Dout;
480                                 64'h0000000040000000: AndEnabled_0[63:32] = config_mem_BRAM_Dout;
481                                 64'h0000000020000000: MTEnabled[31:0] = config_mem_BRAM_Dout;
482                                 64'h0000000010000000: MTEnabled[63:32] = config_mem_BRAM_Dout;
483
484                                 64'h0000000008000000:begin
485                                                         CapsEnabled_0 = config_mem_BRAM_Dout[1:0];
486                                                         input_choice_1 = config_mem_BRAM_Dout[2];
487                                                         input_choice_2 = config_mem_BRAM_Dout[3];
488                                                     end
489                                 64'h0000000004000000:begin
490                                                         output_choice_OA1 = config_mem_BRAM_Dout[5:0];
491                                                         output_choice_OA2 = config_mem_BRAM_Dout[13:8];
492                                                         output_choice_OB1 = config_mem_BRAM_Dout[21:16];
493                                                         output_choice_OB2 = config_mem_BRAM_Dout[29:24];
494                                                     end
495                                 endcase
496         endcase
497 end
498
499 //signal generator
500 reg signal_gen = 0;
501 reg [4:0] signal_state = 0;
502
503 always@(posedge CLK_120MHZ_OUT)
504 begin
505             if (signal_state == 0)
506             begin
507                     signal_gen <= 1;
508                     signal_state <= 31;
509             end
510             else begin
511                     signal_gen <= 0;
512                     signal_state <= signal_state - 1;
513             end
514 end
515
516
```

```
517 //fan-out generated signal (signal_gen)
518 //and delay it as needed
519 wire signal_gen_A;
520 wire signal_gen_B;
521 wire signal_gen_szinti;
522 wire signal_gen_align;
523 wire signal_gen_output;
524 wire signal_gen_choice;
525 leftdelay signal_gen_split_1(signal_gen,signal_gen_A,signal_gen_B);
526 leftdelay signal_gen_split_2(signal_gen_A,signal_gen_szinti,signal_gen_align);
527 leftdelay signal_gen_split_3(signal_gen_B,signal_gen_output,signal_gen_choice);
528
529 wire signal_gen_left;
530 wire signal_gen_right;
531 MOVE_delays szinti_delays(signal_gen_szinti, signal_gen_left, signal_gen_right , CLK_40MHZ_OUT, inc_O[0], delay_rst);
532
533 wire signal_gen_left_type;
534 wire signal_gen_right_type;
535 ChoiceSig ChoiceSig_left (signal_gen_choice, signal_gen_left, input_choice_2, signal_gen_left_type);
536 ChoiceSig ChoiceSig_right (signal_gen_choice, signal_gen_right, input_choice_2, signal_gen_right_type);
537
538
539 //generate MT
540 genvar i;
541 generate
542  for (i=0; i < count; i=i+1) begin : IN
543         //restore LVDS input from differential information
544         IBUFDS #(.DIFF_TERM("TRUE"),.IOSTANDARD("LVDS_25")) IBUFDS_CONN_L ( .O(CONN_L[i]),
545                                                                             .I(CONN_LP[i]),
546                                                                             .IB(CONN_LN[i]) );
547         IBUFDS #(.DIFF_TERM("TRUE"),.IOSTANDARD("LVDS_25")) IBUFDS_CONN_R ( .O(CONN_R[i]),
548                                                                             .I(CONN_RP[i]),
549                                                                             .IB(CONN_RN[i]) );
550
551         //portmapping
552         if ( i==0 || i==1 || i==2 || i==3 || i==4 || i==5 || i==6 || i==7 || i==8 || i==9 || i==10 ||
553                 i==11 || i==12 || i==13 || i==14 || i==15 || i==42 || i==43 || i==44 || i==45 || i==57)
554         begin
555                 //normal mapping
556                 LR_delays R_Delay (CONN_R_Delay[i], CONN_R[i], input_choice_1, signal_gen_right_type,
557                                                 CLK_40MHZ_OUT, inc_R[i], delay_rst);
558                 LR_delays L_Delay (CONN_L_Delay[i], CONN_L[i], input_choice_1, signal_gen_left_type,
559                                                 CLK_40MHZ_OUT, inc_L[i], delay_rst);
560         end else if ( i==16 || i== 31 || i==32 || i==33 || i==34 || i==35 || i==36 || i== 37 ||
561                                 i==38 || i==41 || i==62 || i==63)
562         begin
563                 //extra LUT in path, manual redirect path to reduce diff between L and R
564                 LR_delays R_Delay (CONN_R_eDelay[i], CONN_R[i], input_choice_1, signal_gen_right_type,
565                                                 CLK_40MHZ_OUT, inc_R[i], delay_rst);
566                 LR_delays L_Delay (CONN_L_eDelay[i], CONN_L[i], input_choice_1, signal_gen_left_type,
567                                                 CLK_40MHZ_OUT, inc_L[i], delay_rst);
568                 simpledelay R_eDelay(CONN_R_eDelay[i],CONN_R_Delay[i]);
569                 simpledelay L_eDelay(CONN_L_eDelay[i],CONN_L_Delay[i]);
570         end else
571         begin
572                 //L-R inverted mapping
573                 LR_delays R_Delay (CONN_L_Delay[i], CONN_R[i], input_choice_1, signal_gen_left_type,
574                                                 CLK_40MHZ_OUT, inc_R[i], delay_rst);
575                 LR_delays L_Delay (CONN_R_Delay[i], CONN_L[i], input_choice_1, signal_gen_right_type,
576                                                 CLK_40MHZ_OUT, inc_L[i], delay_rst);
577         end
578
579         if (i == 0) meanTimer_0 MT (CONN_L_Delay[0], CONN_R_Delay[0], CONN_O[i],AndEnabled_0[53:0],
580                                                 MTEnabled[i], CapsEnabled_0,AndEnabled_1,CapsEnabled_1);
581         if (i == 1) meanTimer_1 MT (CONN_L_Delay[1], CONN_R_Delay[1], CONN_O[i],AndEnabled_1,
582                                                 MTEnabled[i], CapsEnabled_1,AndEnabled_2,CapsEnabled_2);
583         if (i == 2) meanTimer_2 MT (CONN_L_Delay[2], CONN_R_Delay[2], CONN_O[i],AndEnabled_2,
584                                                 MTEnabled[i], CapsEnabled_2,AndEnabled_3,CapsEnabled_3);
585         if (i == 3) meanTimer_3 MT (CONN_L_Delay[3], CONN_R_Delay[3], CONN_O[i],AndEnabled_3,
586                                                 MTEnabled[i], CapsEnabled_3,AndEnabled_4,CapsEnabled_4);
587         if (i == 4) meanTimer_4 MT (CONN_L_Delay[4], CONN_R_Delay[4], CONN_O[i],AndEnabled_4,
588                                                 MTEnabled[i], CapsEnabled_4,AndEnabled_5,CapsEnabled_5);
589         if (i == 5) meanTimer_5 MT (CONN_L_Delay[5], CONN_R_Delay[5], CONN_O[i],AndEnabled_5,
590                                                 MTEnabled[i], CapsEnabled_5,AndEnabled_6,CapsEnabled_6);
```

```
591         if (i == 6) meanTimer_6 MT (CONN_L_Delay[6], CONN_R_Delay[6], CONN_O[i],AndEnabled_6,
592                                     MTEnabled[i], CapsEnabled_6,AndEnabled_7,CapsEnabled_7);
593         if (i == 7) meanTimer_7 MT (CONN_L_Delay[7], CONN_R_Delay[7], CONN_O[i],AndEnabled_7,
594                                     MTEnabled[i], CapsEnabled_7,AndEnabled_8,CapsEnabled_8);
595         if (i == 8) meanTimer_8 MT (CONN_L_Delay[8], CONN_R_Delay[8], CONN_O[i],AndEnabled_8,
596                                     MTEnabled[i], CapsEnabled_8,AndEnabled_9,CapsEnabled_9);
597         if (i == 9) meanTimer_9 MT (CONN_L_Delay[9], CONN_R_Delay[9], CONN_O[i],AndEnabled_9,
598                                     MTEnabled[i], CapsEnabled_9,AndEnabled_10,CapsEnabled_10);
599         if (i == 10) meanTimer_10 MT (CONN_L_Delay[10], CONN_R_Delay[10], CONN_O[i],AndEnabled_10,
600                                     MTEnabled[i], CapsEnabled_10,AndEnabled_11,CapsEnabled_11);
601         if (i == 11) meanTimer_11 MT (CONN_L_Delay[11], CONN_R_Delay[11], CONN_O[i],AndEnabled_11,
602                                     MTEnabled[i], CapsEnabled_11,AndEnabled_12,CapsEnabled_12);
603         if (i == 12) meanTimer_12 MT (CONN_L_Delay[12], CONN_R_Delay[12], CONN_O[i],AndEnabled_12,
604                                     MTEnabled[i], CapsEnabled_12,AndEnabled_13,CapsEnabled_13);
605         if (i == 13) meanTimer_13 MT (CONN_L_Delay[13], CONN_R_Delay[13], CONN_O[i],AndEnabled_13,
606                                     MTEnabled[i], CapsEnabled_13,AndEnabled_14,CapsEnabled_14);
607         if (i == 14) meanTimer_14 MT (CONN_L_Delay[14], CONN_R_Delay[14], CONN_O[i],AndEnabled_14,
608                                     MTEnabled[i], CapsEnabled_14,AndEnabled_15,CapsEnabled_15);
609         if (i == 15) meanTimer_15 MT (CONN_L_Delay[15], CONN_R_Delay[15], CONN_O[i],AndEnabled_15,
610                                     MTEnabled[i], CapsEnabled_15,AndEnabled_16,CapsEnabled_16);
611         if (i == 16) meanTimer_16 MT (CONN_L_Delay[16], CONN_R_Delay[16], CONN_O[i],AndEnabled_16,
612                                     MTEnabled[i], CapsEnabled_16,AndEnabled_17,CapsEnabled_17);
613         if (i == 17) meanTimer_17 MT (CONN_L_Delay[17], CONN_R_Delay[17], CONN_O[i],AndEnabled_17,
614                                     MTEnabled[i], CapsEnabled_17,AndEnabled_18,CapsEnabled_18);
615         if (i == 18) meanTimer_18 MT (CONN_L_Delay[18], CONN_R_Delay[18], CONN_O[i],AndEnabled_18,
616                                     MTEnabled[i], CapsEnabled_18,AndEnabled_19,CapsEnabled_19);
617         if (i == 19) meanTimer_19 MT (CONN_L_Delay[19], CONN_R_Delay[19], CONN_O[i],AndEnabled_19,
618                                     MTEnabled[i], CapsEnabled_19,AndEnabled_20,CapsEnabled_20);
619         if (i == 20) meanTimer_20 MT (CONN_L_Delay[20], CONN_R_Delay[20], CONN_O[i],AndEnabled_20,
620                                     MTEnabled[i], CapsEnabled_20,AndEnabled_21,CapsEnabled_21);
621         if (i == 21) meanTimer_21 MT (CONN_L_Delay[21], CONN_R_Delay[21], CONN_O[i],AndEnabled_21,
622                                     MTEnabled[i], CapsEnabled_21,AndEnabled_22,CapsEnabled_22);
623         if (i == 22) meanTimer_22 MT (CONN_L_Delay[22], CONN_R_Delay[22], CONN_O[i],AndEnabled_22,
624                                     MTEnabled[i], CapsEnabled_22,AndEnabled_23,CapsEnabled_23);
625         if (i == 23) meanTimer_23 MT (CONN_L_Delay[23], CONN_R_Delay[23], CONN_O[i],AndEnabled_23,
626                                     MTEnabled[i], CapsEnabled_23,AndEnabled_24,CapsEnabled_24);
627         if (i == 24) meanTimer_24 MT (CONN_L_Delay[24], CONN_R_Delay[24], CONN_O[i],AndEnabled_24,
628                                     MTEnabled[i], CapsEnabled_24,AndEnabled_25,CapsEnabled_25);
629         if (i == 25) meanTimer_25 MT (CONN_L_Delay[25], CONN_R_Delay[25], CONN_O[i],AndEnabled_25,
630                                     MTEnabled[i], CapsEnabled_25,AndEnabled_26,CapsEnabled_26);
631         if (i == 26) meanTimer_26 MT (CONN_L_Delay[26], CONN_R_Delay[26], CONN_O[i],AndEnabled_26,
632                                     MTEnabled[i], CapsEnabled_26,AndEnabled_27,CapsEnabled_27);
633         if (i == 27) meanTimer_27 MT (CONN_L_Delay[27], CONN_R_Delay[27], CONN_O[i],AndEnabled_27,
634                                     MTEnabled[i], CapsEnabled_27,AndEnabled_28,CapsEnabled_28);
635         if (i == 28) meanTimer_28 MT (CONN_L_Delay[28], CONN_R_Delay[28], CONN_O[i],AndEnabled_28,
636                                     MTEnabled[i], CapsEnabled_28,AndEnabled_29,CapsEnabled_29);
637         if (i == 29) meanTimer_29 MT (CONN_L_Delay[29], CONN_R_Delay[29], CONN_O[i],AndEnabled_29,
638                                     MTEnabled[i], CapsEnabled_29,AndEnabled_30,CapsEnabled_30);
639         if (i == 30) meanTimer_30 MT (CONN_L_Delay[30], CONN_R_Delay[30], CONN_O[i],AndEnabled_30,
640                                     MTEnabled[i], CapsEnabled_30,AndEnabled_31,CapsEnabled_31);
641
642         if (i == 31) meanTimer_31 MT (CONN_L_Delay[31], CONN_R_Delay[31], CONN_O[i],AndEnabled_31,
643                                     MTEnabled[i], CapsEnabled_31,,);
644         if (i == 32) meanTimer_32 MT (CONN_L_Delay[32], CONN_R_Delay[32], CONN_O[i],AndEnabled_0[53:0],
645                                     MTEnabled[i], CapsEnabled_0,AndEnabled_33,CapsEnabled_33);
646
647         if (i == 33) meanTimer_33 MT (CONN_L_Delay[33], CONN_R_Delay[33], CONN_O[i],AndEnabled_33,
648                                     MTEnabled[i], CapsEnabled_33,AndEnabled_34,CapsEnabled_34);
649         if (i == 34) meanTimer_34 MT (CONN_L_Delay[34], CONN_R_Delay[34], CONN_O[i],AndEnabled_34,
650                                     MTEnabled[i], CapsEnabled_34,AndEnabled_35,CapsEnabled_35);
651         if (i == 35) meanTimer_35 MT (CONN_L_Delay[35], CONN_R_Delay[35], CONN_O[i],AndEnabled_35,
652                                     MTEnabled[i], CapsEnabled_35,AndEnabled_36,CapsEnabled_36);
653         if (i == 36) meanTimer_36 MT (CONN_L_Delay[36], CONN_R_Delay[36], CONN_O[i],AndEnabled_36,
654                                     MTEnabled[i], CapsEnabled_36,AndEnabled_37,CapsEnabled_37);
655         if (i == 37) meanTimer_37 MT (CONN_L_Delay[37], CONN_R_Delay[37], CONN_O[i],AndEnabled_37,
656                                     MTEnabled[i], CapsEnabled_37,AndEnabled_38,CapsEnabled_38);
657         if (i == 38) meanTimer_38 MT (CONN_L_Delay[38], CONN_R_Delay[38], CONN_O[i],AndEnabled_38,
658                                     MTEnabled[i], CapsEnabled_38,AndEnabled_39,CapsEnabled_39);
659         if (i == 39) meanTimer_39 MT (CONN_L_Delay[39], CONN_R_Delay[39], CONN_O[i],AndEnabled_39,
660                                     MTEnabled[i], CapsEnabled_39,AndEnabled_40,CapsEnabled_40);
661         if (i == 40) meanTimer_40 MT (CONN_L_Delay[40], CONN_R_Delay[40], CONN_O[i],AndEnabled_40,
662                                     MTEnabled[i], CapsEnabled_40,AndEnabled_41,CapsEnabled_41);
663         if (i == 41) meanTimer_41 MT (CONN_L_Delay[41], CONN_R_Delay[41], CONN_O[i],AndEnabled_41,
664                                     MTEnabled[i], CapsEnabled_41,AndEnabled_42,CapsEnabled_42);
```

```
665         if (i == 42) meanTimer_42 MT (CONN_L_Delay[42], CONN_R_Delay[42], CONN_O[i],AndEnabled_42,
666                                                 MTEnabled[i], CapsEnabled_42,AndEnabled_43,CapsEnabled_43);
667         if (i == 43) meanTimer_43 MT (CONN_L_Delay[43], CONN_R_Delay[43], CONN_O[i],AndEnabled_43,
668                                                 MTEnabled[i], CapsEnabled_43,AndEnabled_44,CapsEnabled_44);
669         if (i == 44) meanTimer_44 MT (CONN_L_Delay[44], CONN_R_Delay[44], CONN_O[i],AndEnabled_44,
670                                                 MTEnabled[i], CapsEnabled_44,AndEnabled_45,CapsEnabled_45);
671         if (i == 45) meanTimer_45 MT (CONN_L_Delay[45], CONN_R_Delay[45], CONN_O[i],AndEnabled_45,
672                                                 MTEnabled[i], CapsEnabled_45,AndEnabled_46,CapsEnabled_46);
673         if (i == 46) meanTimer_46 MT (CONN_L_Delay[46], CONN_R_Delay[46], CONN_O[i],AndEnabled_46,
674                                                 MTEnabled[i], CapsEnabled_46,AndEnabled_47,CapsEnabled_47);
675         if (i == 47) meanTimer_47 MT (CONN_L_Delay[47], CONN_R_Delay[47], CONN_O[i],AndEnabled_47,
676                                                 MTEnabled[i], CapsEnabled_47,AndEnabled_48,CapsEnabled_48);
677         if (i == 48) meanTimer_48 MT (CONN_L_Delay[48], CONN_R_Delay[48], CONN_O[i],AndEnabled_48,
678                                                 MTEnabled[i], CapsEnabled_48,AndEnabled_49,CapsEnabled_49);
679         if (i == 49) meanTimer_49 MT (CONN_L_Delay[49], CONN_R_Delay[49], CONN_O[i],AndEnabled_49,
680                                                 MTEnabled[i], CapsEnabled_49,AndEnabled_50,CapsEnabled_50);
681         if (i == 50) meanTimer_50 MT (CONN_L_Delay[50], CONN_R_Delay[50], CONN_O[i],AndEnabled_50,
682                                                 MTEnabled[i], CapsEnabled_50,AndEnabled_51,CapsEnabled_51);
683         if (i == 51) meanTimer_51 MT (CONN_L_Delay[51], CONN_R_Delay[51], CONN_O[i],AndEnabled_51,
684                                                 MTEnabled[i], CapsEnabled_51,AndEnabled_52,CapsEnabled_52);
685         if (i == 52) meanTimer_52 MT (CONN_L_Delay[52], CONN_R_Delay[52], CONN_O[i],AndEnabled_52,
686                                                 MTEnabled[i], CapsEnabled_52,AndEnabled_53,CapsEnabled_53);
687         if (i == 53) meanTimer_53 MT (CONN_L_Delay[53], CONN_R_Delay[53], CONN_O[i],AndEnabled_53,
688                                                 MTEnabled[i], CapsEnabled_53,AndEnabled_54,CapsEnabled_54);
689         if (i == 54) meanTimer_54 MT (CONN_L_Delay[54], CONN_R_Delay[54], CONN_O[i],AndEnabled_54,
690                                                 MTEnabled[i], CapsEnabled_54,AndEnabled_55,CapsEnabled_55);
691         if (i == 55) meanTimer_55 MT (CONN_L_Delay[55], CONN_R_Delay[55], CONN_O[i],AndEnabled_55,
692                                                 MTEnabled[i], CapsEnabled_55,AndEnabled_56,CapsEnabled_56);
693         if (i == 56) meanTimer_56 MT (CONN_L_Delay[56], CONN_R_Delay[56], CONN_O[i],AndEnabled_56,
694                                                 MTEnabled[i], CapsEnabled_56,AndEnabled_57,CapsEnabled_57);
695         if (i == 57) meanTimer_57 MT (CONN_L_Delay[57], CONN_R_Delay[57], CONN_O[i],AndEnabled_57,
696                                                 MTEnabled[i], CapsEnabled_57,AndEnabled_58,CapsEnabled_58);
697         if (i == 58) meanTimer_58 MT (CONN_L_Delay[58], CONN_R_Delay[58], CONN_O[i],AndEnabled_58,
698                                                 MTEnabled[i], CapsEnabled_58,AndEnabled_59,CapsEnabled_59);
699         if (i == 59) meanTimer_59 MT (CONN_L_Delay[59], CONN_R_Delay[59], CONN_O[i],AndEnabled_59,
700                                                 MTEnabled[i], CapsEnabled_59,AndEnabled_60,CapsEnabled_60);
701         if (i == 60) meanTimer_60 MT (CONN_L_Delay[60], CONN_R_Delay[60], CONN_O[i],AndEnabled_60,
702                                                 MTEnabled[i], CapsEnabled_60,AndEnabled_61,CapsEnabled_61);
703         if (i == 61) meanTimer_61 MT (CONN_L_Delay[61], CONN_R_Delay[61], CONN_O[i],AndEnabled_61,
704                                                 MTEnabled[i], CapsEnabled_61,AndEnabled_62,CapsEnabled_62);
705         if (i == 62) meanTimer_62 MT (CONN_L_Delay[62], CONN_R_Delay[62], CONN_O[i],AndEnabled_62,
706                                                 MTEnabled[i], CapsEnabled_62,AndEnabled_63,CapsEnabled_63);
707         if (i == 63) meanTimer_63 MT (CONN_L_Delay[63], CONN_R_Delay[63], CONN_O[i],AndEnabled_63,
708                                                 MTEnabled[i], CapsEnabled_63,,);
709
710  end
711 endgenerate
712
713
714 //prepare Matrix-Inputs
715 SWDELAY SD1_16 (CONN_H1[0] ,CONN_O[16], CLK_40MHZ_OUT, delay_rst, inc_M[16]);
716 SWDELAY SD1_17 (CONN_H1[1] ,CONN_O[17], CLK_40MHZ_OUT, delay_rst, inc_M[17]);
717 SWDELAY SD1_18 (CONN_H1[2] ,CONN_O[18], CLK_40MHZ_OUT, delay_rst, inc_M[18]);
718 SWDELAY SD1_19 (CONN_H1[3] ,CONN_O[19], CLK_40MHZ_OUT, delay_rst, inc_M[19]);
719 SWDELAY SD1_20 (CONN_H1[4] ,CONN_O[20], CLK_40MHZ_OUT, delay_rst, inc_M[20]);
720 SWDELAY SD1_21 (CONN_H1[5] ,CONN_O[21], CLK_40MHZ_OUT, delay_rst, inc_M[21]);
721 SWDELAY SD1_22 (CONN_H1[6] ,CONN_O[22], CLK_40MHZ_OUT, delay_rst, inc_M[22]);
722 SWDELAY SD1_23 (CONN_H1[7] ,CONN_O[23], CLK_40MHZ_OUT, delay_rst, inc_M[23]);
723 SWDELAY SD1_24 (CONN_H1[8] ,CONN_O[24], CLK_40MHZ_OUT, delay_rst, inc_M[24]);
724 SWDELAY SD1_25 (CONN_H1[9] ,CONN_O[25], CLK_40MHZ_OUT, delay_rst, inc_M[25]);
725 SWDELAY SD1_26 (CONN_H1[10],CONN_O[26], CLK_40MHZ_OUT, delay_rst, inc_M[26]);
726 SWDELAY SD1_27 (CONN_H1[11],CONN_O[27], CLK_40MHZ_OUT, delay_rst, inc_M[27]);
727 SWDELAY SD1_28 (CONN_H1[12],CONN_O[28], CLK_40MHZ_OUT, delay_rst, inc_M[28]);
728 SWDELAY SD1_29 (CONN_H1[13],CONN_O[29], CLK_40MHZ_OUT, delay_rst, inc_M[29]);
729 SWDELAY SD1_30 (CONN_H1[14],CONN_O[30], CLK_40MHZ_OUT, delay_rst, inc_M[30]);
730 SWDELAY SD1_31 (CONN_H1[15],CONN_O[31], CLK_40MHZ_OUT, delay_rst, inc_M[31]);
731 SWDELAY SD1_15 (CONN_H1[16],CONN_O[15], CLK_40MHZ_OUT, delay_rst, inc_M[15]);
732 SWDELAY SD1_14 (CONN_H1[17],CONN_O[14], CLK_40MHZ_OUT, delay_rst, inc_M[14]);
733 SWDELAY SD1_13 (CONN_H1[18],CONN_O[13], CLK_40MHZ_OUT, delay_rst, inc_M[13]);
734 SWDELAY SD1_12 (CONN_H1[19],CONN_O[12], CLK_40MHZ_OUT, delay_rst, inc_M[12]);
735 SWDELAY SD1_11 (CONN_H1[20],CONN_O[11], CLK_40MHZ_OUT, delay_rst, inc_M[11]);
736 SWDELAY SD1_10 (CONN_H1[21],CONN_O[10], CLK_40MHZ_OUT, delay_rst, inc_M[10]);
737 SWDELAY SD1_9 (CONN_H1[22],CONN_O[9] , CLK_40MHZ_OUT, delay_rst, inc_M[9]);
738 SWDELAY SD1_8 (CONN_H1[23],CONN_O[8] , CLK_40MHZ_OUT, delay_rst, inc_M[8]);
```

```
739 SWDELAY SD1_7 (CONN_H1[24],CONN_O[7] , CLK_40MHZ_OUT, delay_rst, inc_M[7]);
740 SWDELAY SD1_6 (CONN_H1[25],CONN_O[6] , CLK_40MHZ_OUT, delay_rst, inc_M[6]);
741 SWDELAY SD1_5 (CONN_H1[26],CONN_O[5] , CLK_40MHZ_OUT, delay_rst, inc_M[5]);
742 SWDELAY SD1_4 (CONN_H1[27],CONN_O[4] , CLK_40MHZ_OUT, delay_rst, inc_M[4]);
743 SWDELAY SD1_3 (CONN_H1[28],CONN_O[3] , CLK_40MHZ_OUT, delay_rst, inc_M[3]);
744 SWDELAY SD1_2 (CONN_H1[29],CONN_O[2] , CLK_40MHZ_OUT, delay_rst, inc_M[2]);
745 SWDELAY SD1_1 (CONN_H1[30],CONN_O[1] , CLK_40MHZ_OUT, delay_rst, inc_M[1]);
746 SWDELAY SD1_0 (CONN_H1[31],CONN_O[0] , CLK_40MHZ_OUT, delay_rst, inc_M[0]);
747
748 SWDELAY SD2_34 (CONN_H2[0],CONN_O[34], CLK_40MHZ_OUT, delay_rst, inc_M[34]);
749 SWDELAY SD2_33 (CONN_H2[1],CONN_O[33], CLK_40MHZ_OUT, delay_rst, inc_M[33]);
750 SWDELAY SD2_35 (CONN_H2[2],CONN_O[35], CLK_40MHZ_OUT, delay_rst, inc_M[35]);
751 SWDELAY SD2_32 (CONN_H2[3],CONN_O[32], CLK_40MHZ_OUT, delay_rst, inc_M[32]);
752 SWDELAY SD2_36 (CONN_H2[4],CONN_O[36], CLK_40MHZ_OUT, delay_rst, inc_M[36]);
753 SWDELAY SD2_37 (CONN_H2[5],CONN_O[37], CLK_40MHZ_OUT, delay_rst, inc_M[37]);
754 SWDELAY SD2_38 (CONN_H2[6],CONN_O[38], CLK_40MHZ_OUT, delay_rst, inc_M[38]);
755 SWDELAY SD2_39 (CONN_H2[7],CONN_O[39], CLK_40MHZ_OUT, delay_rst, inc_M[39]);
756 SWDELAY SD2_40 (CONN_H2[8],CONN_O[40], CLK_40MHZ_OUT, delay_rst, inc_M[40]);
757 SWDELAY SD2_41 (CONN_H2[9],CONN_O[41], CLK_40MHZ_OUT, delay_rst, inc_M[41]);
758 SWDELAY SD2_42 (CONN_H2[10],CONN_O[42], CLK_40MHZ_OUT, delay_rst, inc_M[42]);
759 SWDELAY SD2_43 (CONN_H2[11],CONN_O[43], CLK_40MHZ_OUT, delay_rst, inc_M[43]);
760 SWDELAY SD2_44 (CONN_H2[12],CONN_O[44], CLK_40MHZ_OUT, delay_rst, inc_M[44]);
761 SWDELAY SD2_45 (CONN_H2[13],CONN_O[45], CLK_40MHZ_OUT, delay_rst, inc_M[45]);
762 SWDELAY SD2_46 (CONN_H2[14],CONN_O[46], CLK_40MHZ_OUT, delay_rst, inc_M[46]);
763 SWDELAY SD2_47 (CONN_H2[15],CONN_O[47], CLK_40MHZ_OUT, delay_rst, inc_M[47]);
764 SWDELAY SD2_48 (CONN_H2[16],CONN_O[48], CLK_40MHZ_OUT, delay_rst, inc_M[48]);
765 SWDELAY SD2_49 (CONN_H2[17],CONN_O[49], CLK_40MHZ_OUT, delay_rst, inc_M[49]);
766 SWDELAY SD2_50 (CONN_H2[18],CONN_O[50], CLK_40MHZ_OUT, delay_rst, inc_M[50]);
767 SWDELAY SD2_51 (CONN_H2[19],CONN_O[51], CLK_40MHZ_OUT, delay_rst, inc_M[51]);
768 SWDELAY SD2_52 (CONN_H2[20],CONN_O[52], CLK_40MHZ_OUT, delay_rst, inc_M[52]);
769 SWDELAY SD2_53 (CONN_H2[21],CONN_O[53], CLK_40MHZ_OUT, delay_rst, inc_M[53]);
770 SWDELAY SD2_54 (CONN_H2[22],CONN_O[54], CLK_40MHZ_OUT, delay_rst, inc_M[54]);
771 SWDELAY SD2_55 (CONN_H2[23],CONN_O[55], CLK_40MHZ_OUT, delay_rst, inc_M[55]);
772 SWDELAY SD2_56 (CONN_H2[24],CONN_O[56], CLK_40MHZ_OUT, delay_rst, inc_M[56]);
773 SWDELAY SD2_57 (CONN_H2[25],CONN_O[57], CLK_40MHZ_OUT, delay_rst, inc_M[57]);
774 SWDELAY SD2_58 (CONN_H2[26],CONN_O[58], CLK_40MHZ_OUT, delay_rst, inc_M[58]);
775 SWDELAY SD2_59 (CONN_H2[27],CONN_O[59], CLK_40MHZ_OUT, delay_rst, inc_M[59]);
776 SWDELAY SD2_60 (CONN_H2[28],CONN_O[60], CLK_40MHZ_OUT, delay_rst, inc_M[60]);
777 SWDELAY SD2_61 (CONN_H2[29],CONN_O[61], CLK_40MHZ_OUT, delay_rst, inc_M[61]);
778 SWDELAY SD2_62 (CONN_H2[30],CONN_O[62], CLK_40MHZ_OUT, delay_rst, inc_M[62]);
779 SWDELAY SD2_63 (CONN_H2[31],CONN_O[63], CLK_40MHZ_OUT, delay_rst, inc_M[63]);
780
781
782 //Generate Output-Signals
783 wire OUT_MATRIX, OUT_PUREMATRIX, OUT_H1, OUT_H2;
784 wire OUT_H1_Full, OUT_H2_Full;
785 wire OUT_OA1, OUT_OA2, OUT_OB1, OUT_OB2;
786 wire BUF_OA1, BUF_OA2, BUF_OB1, BUF_OB2;
787 wire signal_align;
788
789 matrix matrix_inst(CONN_H1,CONN_H2,OUT_MATRIX,OUT_PUREMATRIX,OUT_H1,OUT_H2,
790         MatrixEnReg00, MatrixEnReg01, MatrixEnReg02, MatrixEnReg03, MatrixEnReg04, MatrixEnReg05,
791         MatrixEnReg06, MatrixEnReg07, MatrixEnReg08, MatrixEnReg09, MatrixEnReg10, MatrixEnReg11,
792         MatrixEnReg12, MatrixEnReg13, MatrixEnReg14, MatrixEnReg15, MatrixEnReg16, MatrixEnReg17,
793         MatrixEnReg18, MatrixEnReg19, MatrixEnReg20, MatrixEnReg21, MatrixEnReg22, MatrixEnReg23,
794         MatrixEnReg24, MatrixEnReg25, MatrixEnReg26, MatrixEnReg27, MatrixEnReg28, MatrixEnReg29,
795         MatrixEnReg30, MatrixEnReg31);
796
797
798 SINGLE_delay H1_full_delay(OUT_H1, OUT_H1_Full, CLK_40MHZ_OUT, inc_O[1], delay_rst);
799 SINGLE_delay H2_full_delay(OUT_H2, OUT_H2_Full, CLK_40MHZ_OUT, inc_O[2], delay_rst);
800 assign OUT_FULL = OUT_H1_Full | OUT_H2_Full;
801
802 ALIGN_delays alignsig_delays(signal_gen_align, signal_align, CLK_40MHZ_OUT, inc_O[3], inc_O[4], delay_rst);
803
804
805 //Those are all possible Output Signals
806 OutDeMux output_selector ( OUT_H1, OUT_H2, OUT_FULL, OUT_MATRIX, OUT_PUREMATRIX,
807                                         signal_gen_output,
808                                         signal_gen_left_type,
809                                         signal_gen_right_type,
810                                         signal_align,
811                                         output_choice_OA1, output_choice_OA2, output_choice_OB1, output_choice_OB2,
812                                         OUT_OA1, OUT_OA2, OUT_OB1, OUT_OB2);
```

```
813
814 SINGLE_delay OA1_delay(OUT_OA1, BUF_OA1, CLK_40MHZ_OUT, inc_O[5], delay_rst);
815 SINGLE_delay OA2_delay(OUT_OA2, BUF_OA2, CLK_40MHZ_OUT, inc_O[6], delay_rst);
816 SINGLE_delay OB1_delay(OUT_OB1, BUF_OB1, CLK_40MHZ_OUT, inc_O[7], delay_rst);
817 SINGLE_delay OB2_delay(OUT_OB2, BUF_OB2, CLK_40MHZ_OUT, inc_O[8], delay_rst);
818
819 simpledelay OA1_buf(BUF_OA1,CONN_OA1);
820 simpledelay OA2_buf(BUF_OA2,CONN_OA2);
821 simpledelay OB1_buf(BUF_OB1,CONN_OB1);
822 simpledelay OB2_buf(BUF_OB2,CONN_OB2);
823
824 // This is needed for the IODELAYs, the pll generates our reference clk
825 // and the IDELAYCTRL is in charge of controlling that clk in terms of temperature changes
826 (* LOC = "IDELAYCTRL_X0Y7" *) IDELAYCTRL CTRL_IODELAY_07_MT (.REFCLK(CLK_200MHZ_OUT),
827                                                 .RST(delay_rst),.RDY());
828 (* LOC = "IDELAYCTRL_X0Y6" *) IDELAYCTRL CTRL_IODELAY_06_MT (.REFCLK(CLK_200MHZ_OUT),
829                                                 .RST(delay_rst),.RDY());
830 (* LOC = "IDELAYCTRL_X0Y5" *) IDELAYCTRL CTRL_IODELAY_05_MT (.REFCLK(CLK_200MHZ_OUT),
831                                                 .RST(delay_rst),.RDY());
832 (* LOC = "IDELAYCTRL_X0Y4" *) IDELAYCTRL CTRL_IODELAY_04_MT (.REFCLK(CLK_200MHZ_OUT),
833                                                 .RST(delay_rst),.RDY());
834 (* LOC = "IDELAYCTRL_X0Y3" *) IDELAYCTRL CTRL_IODELAY_03_MT (.REFCLK(CLK_200MHZ_OUT),
835                                                 .RST(delay_rst),.RDY());
836 (* LOC = "IDELAYCTRL_X0Y2" *) IDELAYCTRL CTRL_IODELAY_02_MT (.REFCLK(CLK_200MHZ_OUT),
837                                                 .RST(delay_rst),.RDY());
838 (* LOC = "IDELAYCTRL_X0Y1" *) IDELAYCTRL CTRL_IODELAY_01_MT (.REFCLK(CLK_200MHZ_OUT),
839                                                 .RST(delay_rst),.RDY());
840 (* LOC = "IDELAYCTRL_X0Y0" *) IDELAYCTRL CTRL_IODELAY_00_MT (.REFCLK(CLK_200MHZ_OUT),
841                                                 .RST(delay_rst),.RDY());
842 (* LOC = "IDELAYCTRL_X2Y6" *) IDELAYCTRL CTRL_IODELAY_26_MT (.REFCLK(CLK_200MHZ_OUT),
843                                                 .RST(delay_rst),.RDY());
844 (* LOC = "IDELAYCTRL_X2Y4" *) IDELAYCTRL CTRL_IODELAY_24_MT (.REFCLK(CLK_200MHZ_OUT),
845                                                 .RST(delay_rst),.RDY());
846 (* LOC = "IDELAYCTRL_X2Y2" *) IDELAYCTRL CTRL_IODELAY_22_MT (.REFCLK(CLK_200MHZ_OUT),
847                                                 .RST(delay_rst),.RDY());
848 (* LOC = "IDELAYCTRL_X2Y1" *) IDELAYCTRL CTRL_IODELAY_21_MT (.REFCLK(CLK_200MHZ_OUT),
849                                                 .RST(delay_rst),.RDY());
850 (* LOC = "IDELAYCTRL_X1Y7" *) IDELAYCTRL CTRL_IODELAY_17_MT (.REFCLK(CLK_200MHZ_OUT),
851                                                 .RST(delay_rst),.RDY());
852
853 //CLK for the IODELAYs
854 delay_pll PLL_200MHZ (
855                           .CLKIN1_IN(CLK_40MHZ_OUT),
856                           .CLKOUT0_OUT(CLK_200MHZ_OUT)
857                           );
858
859 endmodule
```

Listing A.2: parts.v

```
1  `timescale 1ns / 1ps
2
3  //switch an optional wire into the signal path
4  module SwitchDelayElement(In,Out, DOut, DIn, Ctrl);
5
6          (* S="TRUE" *) input wire In, DIn;
7          (* S="TRUE" *) output wire Out, DOut;
8          input wire Ctrl;
9
10         (* LOCK_PINS="ALL" *)
11         LUT6_2 #(.INIT(64'hBBBB_8888_FFFF_0000)) S_1 (
12                 .I0(DIn),
13                 .I1(Ctrl),
14                 .I4(In),
15                 .O5(DOut),
16                 .O6(Out),
17                 .I5(1'b1)
18         );
19
20 endmodule
21
22 //shift register for the SWDELAY
23 module shiftreg (CLK, CLR, INC, OUT);
24
25         input wire CLK,INC,CLR;
26         output wire [20:0] OUT;
27         reg [20:0] SwDCtrl;
28
29         always @(posedge CLK)
30         begin
31                 if (CLR) SwDCtrl = 0;
32                 else if (INC) SwDCtrl = {SwDCtrl[19:0],1'b1};
33         end
34         assign OUT = SwDCtrl;
35
36 endmodule
37
38 //SWDELAY, constructed so that works in the same way as the IODELAYs
39 module SWDELAY (Out,In, CLK, CLR, INC);
40
41         (* S="TRUE" *) input wire In, CLK, CLR, INC;
42         (* S="TRUE" *) output wire Out;
43         wire step_1a,step_2a,step_3a,step_4a,step_5a,step_6a,step_7a,step_8a;
44         wire step_1b,step_2b,step_3b,step_4b,step_5b,step_6b,step_7b,step_8b;
45         (* S="TRUE" *) wire [20:0] Ctrl;
46
47         shiftreg switchreg (CLK, CLR, INC, Ctrl);
48
49         simpledelay SwD_21 (step_21a, step_21b);
50         SwitchDelayElement SwD_20 (step_20a, step_20b, step_21a, step_21b, Ctrl[20]);
51         SwitchDelayElement SwD_19 (step_19a, step_19b, step_20a, step_20b, Ctrl[19]);
52         SwitchDelayElement SwD_18 (step_18a, step_18b, step_19a, step_19b, Ctrl[18]);
53         SwitchDelayElement SwD_17 (step_17a, step_17b, step_18a, step_18b, Ctrl[17]);
54         SwitchDelayElement SwD_16 (step_16a, step_16b, step_17a, step_17b, Ctrl[16]);
55         SwitchDelayElement SwD_15 (step_15a, step_15b, step_16a, step_16b, Ctrl[15]);
56         SwitchDelayElement SwD_14 (step_14a, step_14b, step_15a, step_15b, Ctrl[14]);
57         SwitchDelayElement SwD_13 (step_13a, step_13b, step_14a, step_14b, Ctrl[13]);
58         SwitchDelayElement SwD_12 (step_12a, step_12b, step_13a, step_13b, Ctrl[12]);
59         SwitchDelayElement SwD_11 (step_11a, step_11b, step_12a, step_12b, Ctrl[11]);
60         SwitchDelayElement SwD_10 (step_10a, step_10b, step_11a, step_11b, Ctrl[10]);
61         SwitchDelayElement SwD_9 (step_9a , step_9b , step_10a, step_10b, Ctrl[9]);
62         SwitchDelayElement SwD_8 (step_8a , step_8b , step_9a , step_9b , Ctrl[8]);
63         SwitchDelayElement SwD_7 (step_7a , step_7b , step_8a , step_8b , Ctrl[7]);
64         SwitchDelayElement SwD_6 (step_6a , step_6b , step_7a , step_7b , Ctrl[6]);
65         SwitchDelayElement SwD_5 (step_5a , step_5b , step_6a , step_6b , Ctrl[5]);
66         SwitchDelayElement SwD_4 (step_4a , step_4b , step_5a , step_5b , Ctrl[4]);
67         SwitchDelayElement SwD_3 (step_3a , step_3b , step_4a , step_4b , Ctrl[3]);
68         SwitchDelayElement SwD_2 (step_2a , step_2b , step_3a , step_3b , Ctrl[2]);
69         SwitchDelayElement SwD_1 (step_1a , step_1b , step_2a , step_2b , Ctrl[1]);
70         SwitchDelayElement SwD_0 (In , Out , step_1a , step_1b , Ctrl[0]);
71
72 endmodule
```

```
73
74 //simply puts a LUT in the signalpath, actual delay is done by positioning this LUT
75 module simpledelay(In,Out);
76
77         (* S="TRUE" *) input wire In;
78         (* S="TRUE" *) output wire Out;
79
80         (* LOCK_PINS="ALL" *)
81         LUT6 #(.INIT(64'hF0F0_F0F0_F0F0_F0F0)) S_1 (
82                 .O(Out),
83                 .I2(In)
84         );
85
86 endmodule
87
88 //leftdelay of the STEP-LUT
89 module leftdelay(In,Out,aOut);
90
91         (* S="TRUE" *) input wire In;
92         (* S="TRUE" *) output wire Out;
93         (* S="TRUE" *) output wire aOut;
94
95         (* LOCK_PINS="ALL" *)
96         LUT6_2 #(.INIT(64'hF0F0_F0F0_F0F0_F0F0)) lDelay_LUT (
97                 .O5(aOut),
98                 .O6(Out),
99                 .I5(1'b1),
100                .I2(In)
101        );
102 endmodule
103
104 //rightdelay of the STEP-LUT
105 module rightdelay(In,Out,aOut);
106
107        (* S="TRUE" *) input wire In;
108        (* S="TRUE" *) output wire Out;
109        (* S="TRUE" *) output wire aOut;
110
111        (* LOCK_PINS="ALL" *)
112        LUT6_2 #(.INIT(64'hF0F0_F0F0_F0F0_F0F0)) rDelay_LUT (
113                .O5(aOut),
114                .O6(Out),
115                .I5(1'b1),
116                .I2(In)
117        );
118
119 endmodule
120
121 //STEP-SLICE (orig. name was HOB-CLB, was changed later)
122 module hobclb(andEnable,andEnable_out,lIn,lOut,rIn,rOut,aOut);
123
124        (* S="TRUE" *) input wire andEnable; //active low
125        (* S="TRUE" *) output wire andEnable_out;
126        (* S="TRUE" *) input wire lIn;
127        (* S="TRUE" *) output wire lOut;
128        (* S="TRUE" *) input wire rIn;
129        (* S="TRUE" *) output wire rOut;
130        (* S="TRUE" *) output wire aOut;
131        (* S="TRUE" *) wire rAnd;
132        (* S="TRUE" *) wire lAnd;
133
134        rightdelay rHob (rIn, rOut,rAnd);
135        leftdelay lHob (lIn, lOut,lAnd);
136
137        //And them up
138        (* LOCK_PINS="ALL" *)
139        LUT6_2 #(.INIT(64'h4444_0000_aaaa_aaaa)) AND_LUT (
140                .I5(1'b1),
141                .I1(lAnd),
142                .I4(rAnd),
143                .I0(andEnable),
144                .O6(aOut),
145                .O5(andEnable_out)
146        );
```

```
147
148 endmodule
149
150 //upper range limiter= CAP
151 module urlclb(andEnable, andEnable_out, capEnable, capEnable_out, lIn,rIn,rOut,aOut);
152
153         (* S="TRUE" *) input wire andEnable; //active low
154         (* S="TRUE" *) input wire capEnable; //active low
155         (* S="TRUE" *) input wire lIn;
156         (* S="TRUE" *) input wire rIn;
157         (* S="TRUE" *) output wire rOut;
158         (* S="TRUE" *) output wire aOut;
159         (* S="TRUE" *) wire rAnd;
160         (* S="TRUE" *) wire lAnd;
161         (* S="TRUE" *) wire set_signal;
162         (* S="TRUE" *) wire clear_signal;
163         (* S="TRUE" *) wire status_signal;
164         (* S="TRUE" *) wire lOut;
165         (* S="TRUE" *) wire rPass;
166
167         (* S="TRUE" *) output wire andEnable_out;
168         (* S="TRUE" *) output wire capEnable_out;
169         assign andEnable_out = andEnable;
170         assign capEnable_out = capEnable;
171         wire rIn1, rIn2;
172
173
174         (* LOCK_PINS="ALL" *)
175         LUT6_2 #(.INIT(64'hF0F0_F0F0_F0F0_F0F0)) Split_RLM (
176                 .O5(rIn1),
177                 .O6(rIn2),
178                 .I5(1'b1),
179                 .I2(rIn)
180         );
181
182         //GateLUT
183         (* LOCK_PINS="ALL" *)
184         LUT6_2 #(.INIT(64'hA0A0_F0F0_F0A0_F0F0)) GATE_LUT (
185                 .O5(rPass),
186                 .O6(set_signal),
187                 .I5(1'b1),
188                 .I0(capEnable),
189                 .I2(rIn1),
190                 .I3(status_signal),
191                 .I4(lOut)
192         );
193
194         rightdelay rHob (rPass, rOut,rAnd);
195         leftdelay lHob (lIn, lOut,lAnd);
196
197         //And them up
198         (* LOCK_PINS="ALL" *)
199         LUT6 #(.INIT(64'h4444_0000_4444_0000)) AND_LUT (
200                 .I1(lAnd),
201                 .I4(rAnd),
202                 .I0(andEnable),
203                 .O(aOut)
204         );
205
206         LDCPE #(.INIT(1'b0)) Status_LUT (
207                 .Q(status_signal), // Data
208                 .CLR(clear_signal), // Asyn
209                 .D(1'b0), // Data
210                 .G(1'b0), // Gate
211                 .GE(1'b0), // Gate
212                 .PRE(set_signal) // Asyn
213         );
214
215         FDCPE_1 #(.INIT(1'b1)) TrailingEdge_RLM (
216                 .Q(clear_signal), // Data output
217                 .C(rIn2), // Clock input
218                 .CE(1'b1), // Clock enable input
219                 .CLR(clear_signal), // Asynchronous clear input
220                 .D(1'b1), // Data input
```

```
221                 .PRE(1'b0) // Asynchronous set input
222         );
223
224 endmodule
225
226 //lower range limiter= CAP
227 module lrlclb(andEnable, andEnable_out, capEnable, capEnable_out, lIn,lOut,rIn,aOut);
228
229         (* S="TRUE" *) input wire andEnable; //active low
230         (* S="TRUE" *) input wire capEnable; //active low
231         (* S="TRUE" *) input wire lIn;
232         (* S="TRUE" *) input wire rIn;
233         (* S="TRUE" *) output wire lOut;
234         (* S="TRUE" *) output wire aOut;
235         (* S="TRUE" *) wire rAnd;
236         (* S="TRUE" *) wire lAnd;
237         (* S="TRUE" *) wire set_signal;
238         (* S="TRUE" *) wire clear_signal;
239         (* S="TRUE" *) wire status_signal;
240         (* S="TRUE" *) wire rOut;
241         (* S="TRUE" *) wire lPass;
242
243         (* S="TRUE" *) output wire andEnable_out;
244         (* S="TRUE" *) output wire capEnable_out;
245         assign andEnable_out = andEnable;
246         assign capEnable_out = capEnable;
247
248         wire lIn1,lIn2;
249
250         (* LOCK_PINS="ALL" *)
251         LUT6_2 #(.INIT(64'hF0F0_F0F0_F0F0_F0F0)) Split_RLM (
252                 .O5(lIn1),
253                 .O6(lIn2),
254                 .I5(1'b1),
255                 .I2(lIn)
256         );
257
258         //GateLUT
259         (* LOCK_PINS="ALL" *)
260         LUT6_2 #(.INIT(64'hA0A0_F0F0_F0A0_F0F0)) GATE_LUT (
261                 .O5(lPass),
262                 .O6(set_signal),
263                 .I5(1'b1),
264                 .I0(capEnable),
265                 .I2(lIn1),
266                 .I3(status_signal),
267                 .I4(rOut)
268         );
269
270         rightdelay rHob (rIn, rOut,rAnd);
271         leftdelay lHob (lPass, lOut,lAnd);
272
273         //And them up
274         (* LOCK_PINS="ALL" *)
275         LUT6 #(.INIT(64'h4444_0000_4444_0000)) AND_LUT (
276                 .I1(lAnd),
277                 .I4(rAnd),
278                 .I0(andEnable),
279                 .O(aOut)
280         );
281
282         LDCPE #(.INIT(1'b0)) Status_LUT ( //name it _LUT so it will be placed inside the CLB
283                 .Q(status_signal), // Data
284                 .CLR(clear_signal), // Asyn
285                 .D(1'b0), // Data
286                 .G(1'b0), // Gate
287                 .GE(1'b0), // Gate
288                 .PRE(set_signal) // Asyn
289         );
290
291         FDCPE_1 #(.INIT(1'b1)) TrailingEdge_RLM ( //name it _LUT so it will be placed inside the CLB
292                 .Q(clear_signal), // Data output
293                 .C(lIn2), // Clock input
294                 .CE(1'b1), // Clock enable input
```

```
295                 .CLR(clear_signal), // Asynchronous clear input
296                 .D(1'b1), // Data input
297                 .PRE(1'b0) // Asynchronous set input
298         );
299
300 endmodule
301
302 // ----------------------------------------------------------------------
303 // OUTPUT-SELECTOR
304 // ----------------------------------------------------------------------
305
306 module AndOrSig (SIG_A, SIG_B, OUT_AND , OUT_OR);
307         input wire SIG_A,SIG_B;
308         output wire OUT_AND,OUT_OR;
309         assign OUT_AND = SIG_A & SIG_B;
310         assign OUT_OR = SIG_A | SIG_B;
311 endmodule
312
313 module ChoiceSig (SIG_A, SIG_B, CTRL, OUT);
314         input wire SIG_A,SIG_B, CTRL;
315         output wire OUT;
316         assign OUT = (CTRL == 1'b0) ? SIG_A : SIG_B;
317 endmodule
318
319 module ActiveSig ( SIG, CTRL, OUT);
320         input wire SIG, CTRL;
321         output wire OUT;
322         assign OUT = SIG & CTRL;
323 endmodule
324
325 //Output-Selector
326 module OutDeMux ( H1,
327                                 H2,
328                                 FULL,
329                                 MATRIX,
330                                 PUREMATRIX,
331                                 signal_gen,
332                                 signal_gen_left,
333                                 signal_gen_right ,
334                                 signal_align,
335                                 output_choice_OA1,
336                                 output_choice_OA2,
337                                 output_choice_OB1,
338                                 output_choice_OB2,
339                                 OA1,
340                                 OA2,
341                                 OB1,
342                                 OB2);
343
344         input wire H1, H2, FULL, MATRIX, PUREMATRIX;
345         input wire signal_gen, signal_gen_left, signal_gen_right, signal_align;
346         input wire [5:0] output_choice_OA1;
347         input wire [5:0] output_choice_OA2;
348         input wire [5:0] output_choice_OB1;
349         input wire [5:0] output_choice_OB2;
350         output wire OA1,OA2,OB1,OB2;
351
352         wire [5:0] OA1_select;
353         wire [5:0] OA2_select;
354         wire [5:0] OB1_select;
355         wire [5:0] OB2_select;
356
357         ActiveSig ActiveSig_OA1_0 (H1    , output_choice_OA1[0], OA1_select[0]);
358         ActiveSig ActiveSig_OA1_1 (H2    , output_choice_OA1[1], OA1_select[1]);
359         ActiveSig ActiveSig_OA1_2 (FULL   , output_choice_OA1[2], OA1_select[2]);
360         ActiveSig ActiveSig_OA1_3 (signal_gen_left  , output_choice_OA1[3], OA1_select[3]);
361         ActiveSig ActiveSig_OA1_4 (signal_gen_right  , output_choice_OA1[4], OA1_select[4]);
362         ActiveSig ActiveSig_OA1_5 (signal_gen  , output_choice_OA1[5], OA1_select[5]);
363
364         ActiveSig ActiveSig_OA2_0 (H1    , output_choice_OA2[0], OA2_select[0]);
365         ActiveSig ActiveSig_OA2_1 (H2    , output_choice_OA2[1], OA2_select[1]);
366         ActiveSig ActiveSig_OA2_2 (FULL   , output_choice_OA2[2], OA2_select[2]);
367         ActiveSig ActiveSig_OA2_3 (signal_gen_left  , output_choice_OA2[3], OA2_select[3]);
368         ActiveSig ActiveSig_OA2_4 (signal_gen_right  , output_choice_OA2[4], OA2_select[4]);
```

```
369         ActiveSig ActiveSig_OA2_5 (signal_gen   , output_choice_OA2[5], OA2_select[5]);
370
371         ActiveSig ActiveSig_OB1_0 (H1     , output_choice_OB1[0], OB1_select[0]);
372         ActiveSig ActiveSig_OB1_1 (H2     , output_choice_OB1[1], OB1_select[1]);
373         ActiveSig ActiveSig_OB1_2 (FULL    , output_choice_OB1[2], OB1_select[2]);
374         ActiveSig ActiveSig_OB1_3 (MATRIX  , output_choice_OB1[3], OB1_select[3]);
375         ActiveSig ActiveSig_OB1_4 (PUREMATRIX  , output_choice_OB1[4], OB1_select[4]);
376         ActiveSig ActiveSig_OB1_5 (signal_gen   , output_choice_OB1[5], OB1_select[5]);
377
378         wire OB1_select_done;
379         wire OB2_select_done;
380         wire OA1_select_done;
381         wire OA2_select_done;
382
383         assign OA1_select_done = (|OA1_select);
384         assign OA2_select_done = (|OA2_select);
385         assign OB1_select_done = (|OB1_select);
386
387         wire signal_and, signal_or;
388         AndOrSig AndOrSig_OB1(OB1_select_done, signal_align, signal_and, signal_or);
389
390         ActiveSig ActiveSig_OB2_0 (H1      , output_choice_OB2[0], OB2_select[0]);
391         ActiveSig ActiveSig_OB2_1 (H2      , output_choice_OB2[1], OB2_select[1]);
392         ActiveSig ActiveSig_OB2_2 (FULL     , output_choice_OB2[2], OB2_select[2]);
393         ActiveSig ActiveSig_OB2_3 (MATRIX    , output_choice_OB2[3], OB2_select[3]);
394         ActiveSig ActiveSig_OB2_4 (signal_and    , output_choice_OB2[4], OB2_select[4]);
395         ActiveSig ActiveSig_OB2_5 (signal_or   , output_choice_OB2[5], OB2_select[5]);
396
397 // assign OB2_select_done = (signal_and | signal_or);
398         assign OB2_select_done = (|OB2_select);
399
400         assign OA1 = ~OA1_select_done;
401         assign OA2 = ~OA2_select_done;
402         assign OB1 = ~OB1_select_done;
403         assign OB2 = ~OB2_select_done;
404
405 endmodule
406
407 module Mux256( select, d, q );
408
409         input[7:0] select; //there are 256 32Bit-Registers
410         input[255:0] d;
411         output q;
412
413         wire q;
414         wire[7:0] select;
415         wire[255:0] d;
416
417         assign q = d[select];
418
419 endmodule
420
421 module DeMux1to64 (select,in,q);
422
423         input wire [5:0] select;
424         input wire in;
425         output [63:0] q;
426
427         reg q;
428
429         always @(select)
430         begin
431                 q = 0;
432                 case (select)
433                         0: q[0] = in;
434                         1: q[1] = in;
435                         2: q[2] = in;
436                         3: q[3] = in;
437                         4: q[4] = in;
438                         5: q[5] = in;
439                         6: q[6] = in;
440                         7: q[7] = in;
441                         8: q[8] = in;
442                         9: q[9] = in;
```

```
443                         10: q[10] = in;
444                         11: q[11] = in;
445                         12: q[12] = in;
446                         13: q[13] = in;
447                         14: q[14] = in;
448                         15: q[15] = in;
449                         16: q[16] = in;
450                         17: q[17] = in;
451                         18: q[18] = in;
452                         19: q[19] = in;
453                         20: q[20] = in;
454                         21: q[21] = in;
455                         22: q[22] = in;
456                         23: q[23] = in;
457                         24: q[24] = in;
458                         25: q[25] = in;
459                         26: q[26] = in;
460                         27: q[27] = in;
461                         28: q[28] = in;
462                         29: q[29] = in;
463                         30: q[30] = in;
464                         31: q[31] = in;
465                         32: q[32] = in;
466                         33: q[33] = in;
467                         34: q[34] = in;
468                         35: q[35] = in;
469                         36: q[36] = in;
470                         37: q[37] = in;
471                         38: q[38] = in;
472                         39: q[39] = in;
473                         40: q[40] = in;
474                         41: q[41] = in;
475                         42: q[42] = in;
476                         43: q[43] = in;
477                         44: q[44] = in;
478                         45: q[45] = in;
479                         46: q[46] = in;
480                         47: q[47] = in;
481                         48: q[48] = in;
482                         49: q[49] = in;
483                         50: q[50] = in;
484                         51: q[51] = in;
485                         52: q[52] = in;
486                         53: q[53] = in;
487                         54: q[54] = in;
488                         55: q[55] = in;
489                         56: q[56] = in;
490                         57: q[57] = in;
491                         58: q[58] = in;
492                         59: q[59] = in;
493                         60: q[60] = in;
494                         61: q[61] = in;
495                         62: q[62] = in;
496                         63: q[63] = in;
497                 endcase
498         end
499
500 endmodule
501
502 // ----------------------------------------------------------------------
503 // IODELAY-STUFF
504 // ----------------------------------------------------------------------
505
506 module LR_delays(out, in, signal_choice, in_gen, CLK, inc, rst);
507
508         input wire CLK;
509         input wire in;
510         input wire in_gen;
511         input wire signal_choice;
512
513         output wire out;
514
515         input wire inc;
516         input wire rst;
```

```
517
518         wire CONN_STEP;
519
520         LUT6 #(.INIT(64'hFFFF_0F0F_F0F0_0000)) choice (
521                 .I5(in),
522                 .I4(in_gen),
523                 .I2(signal_choice),
524                 .O(CONN_STEP)
525         );
526
527         IODELAY #(
528                         .IDELAY_TYPE ("VARIABLE"),
529                         .DELAY_SRC ("DATAIN"),
530                         .IDELAY_VALUE (0),
531                         .REFCLK_FREQUENCY (200.0),
532                         .SIGNAL_PATTERN ("CLOCK"),
533                         .HIGH_PERFORMANCE_MODE ("TRUE")
534                         ) D_1 (
535                                 .DATAIN (CONN_STEP),
536                                 .C (CLK),
537                                 .CE (inc),
538                                 .INC (1'b1),
539                                 .RST (rst),
540                                 .DATAOUT (out)
541                         );
542
543 endmodule
544
545 module SINGLE_delay(in, out, CLK, inc, rst);
546
547         input wire CLK;
548         input wire in;
549         output wire out;
550
551         input wire inc;
552         input wire rst;
553
554         IODELAY #(
555                         .IDELAY_TYPE ("VARIABLE"),
556                         .DELAY_SRC ("DATAIN"),
557                         .IDELAY_VALUE (0),
558                         .REFCLK_FREQUENCY (200.0),
559                         .SIGNAL_PATTERN ("CLOCK"),
560                         .HIGH_PERFORMANCE_MODE ("TRUE")
561                         ) D (
562                                 .DATAIN (in),
563                                 .C (CLK),
564                                 .CE (inc),
565                                 .INC (1'b1),      //inc
566                                 .RST (rst),
567                                 .DATAOUT (out)
568                         );
569 endmodule
570
571 //very large delay by adding multiple IODELAYs - bad, cause lots of jitter
572 module ALIGN_delays (in, out, CLK, inc, inc_fein, rst);
573
574         input wire CLK;
575         input wire in;
576         output wire out;
577
578         input wire inc,inc_fein;
579         input wire rst;
580
581         wire step_0, step_1, step_2, step_3, step_4, step_5, step_6, step_7, step_8, step_9;
582         wire step_10, step_11, step_12, step_13, step_14, step_15, step_16, step_17, step_18;
583
584         SINGLE_delay F_0 (in , step_0 , CLK, inc_fein, rst);
585         SINGLE_delay B_0 (step_0 , step_1 , CLK, inc , rst);
586         SINGLE_delay B_1 (step_1 , step_2 , CLK, inc , rst);
587         SINGLE_delay B_2 (step_2 , step_3 , CLK, inc , rst);
588         SINGLE_delay B_3 (step_3 , step_4 , CLK, inc , rst);
589         SINGLE_delay B_4 (step_4 , step_5 , CLK, inc , rst);
590         SINGLE_delay B_5 (step_5 , step_6 , CLK, inc , rst);
```

```
591         SINGLE_delay B_6 (step_6 , step_7 , CLK, inc , rst);
592         SINGLE_delay B_7 (step_7 , step_8 , CLK, inc , rst);
593         SINGLE_delay B_8 (step_8 , step_9 , CLK, inc , rst);
594         SINGLE_delay B_9 (step_9 , step_10, CLK, inc , rst);
595         SINGLE_delay B_10 (step_10, step_11, CLK, inc , rst);
596         SINGLE_delay B_11 (step_11, step_12, CLK, inc , rst);
597         SINGLE_delay B_12 (step_12, step_13, CLK, inc , rst);
598         SINGLE_delay B_13 (step_13, step_14, CLK, inc , rst);
599         SINGLE_delay B_14 (step_14, step_15, CLK, inc , rst);
600         SINGLE_delay B_15 (step_15, step_16, CLK, inc , rst);
601         SINGLE_delay B_16 (step_16, step_17, CLK, inc , rst);
602         SINGLE_delay B_17 (step_17, step_18, CLK, inc , rst);
603         SINGLE_delay B_18 (step_18, out , CLK, inc , rst);
604
605 endmodule
606
607 //very large delay by adding multiple IODELAYs - bad, cause lots of jitter
608 //this one produces two output signals, which simulate szinit-signals
609 module MOVE_delays(in,left_out,right_out,CLK, inc, rst);
610
611         ...
612         vollständiger Quellcode befindet sich auf der beigefügten CD
613         ...
614
615 endmodule
616
617 // ----------------------------------------------------------------------
618 // All possible pin-combinations for a "2-to-1-OR", so the
619 // permutation-script just needs to change the name of module
620 // ----------------------------------------------------------------------
621
622         ...
623         vollständiger Quellcode befindet sich auf der beigefügten CD
624         ...
625
626 module OrLut51(orSig0,orSig1,orOut);
627         (* S="TRUE" *) input wire orSig0;
628         (* S="TRUE" *) input wire orSig1;
629         (* S="TRUE" *) output wire orOut;
630         (* LOCK_PINS="ALL" *)
631         LUT6 #(.INIT(64'hFFFF_FFFF_CCCC_CCCC)) OR_LUT (
632                 .I5(orSig0),
633                 .I1(orSig1),
634                 .O(orOut)
635         );
636 endmodule
637
638 module OrLut32(orSig0,orSig1,orOut);
639         (* S="TRUE" *) input wire orSig0;
640         (* S="TRUE" *) input wire orSig1;
641         (* S="TRUE" *) output wire orOut;
642         (* LOCK_PINS="ALL" *)
643         LUT6 #(.INIT(64'hFFF0_FFF0_FFF0_FFF0)) OR_LUT (
644                 .I3(orSig0),
645                 .I2(orSig1),
646                 .O(orOut)
647         );
648 endmodule
649
650         ...
651         vollständiger Quellcode befindet sich auf der beigefügten CD
652         ...
```

Listing A.3: meanTimer.v

```
1  `timescale 1ns / 1ps
2
3  module meanTimer_0(lIn, rIn, MTOut, AndEnabled, MTEnabled, CapsEnabled, AndEnabled_out, CapsEnabled_out);
4
5          (* S="TRUE" *) input wire lIn, rIn;
6          (* S="TRUE" *) output wire MTOut;
7
8          (* S="TRUE" *) wire [52:0] lStep, rStep;
9          (* S="TRUE" *) wire [53:0] l1Sig;
10         (* S="TRUE" *) wire [27:0] l2Sig;
11         (* S="TRUE" *) wire [13:0] l3Sig;
12         (* S="TRUE" *) wire [6:0] l4Sig;
13         (* S="TRUE" *) wire [3:0] l5Sig;
14         (* S="TRUE" *) wire [1:0] l6Sig;
15         (* S="TRUE" *) wire orOut;
16
17         (* S="TRUE" *) input wire [53:0] AndEnabled;
18         (* S="TRUE" *) input wire [1:0] CapsEnabled;
19         (* S="TRUE" *) output wire [53:0] AndEnabled_out;
20         (* S="TRUE" *) output wire [1:0] CapsEnabled_out;
21         (* S="TRUE" *) input wire MTEnabled;
22
23         urlclb CLB_53(AndEnabled[53], AndEnabled_out[53], CapsEnabled[1],
24                                 CapsEnabled_out[1], lStep[52], rIn , rStep[0] , l1Sig[53]);
25         hobclb CLB_52(AndEnabled[52], AndEnabled_out[52], lStep[51], lStep[52], rStep[0] , rStep[1] , l1Sig[52]);
26         hobclb CLB_51(AndEnabled[51], AndEnabled_out[51], lStep[50], lStep[51], rStep[1] , rStep[2] , l1Sig[51]);
27         hobclb CLB_50(AndEnabled[50], AndEnabled_out[50], lStep[49], lStep[50], rStep[2] , rStep[3] , l1Sig[50]);
28         hobclb CLB_49(AndEnabled[49], AndEnabled_out[49], lStep[48], lStep[49], rStep[3] , rStep[4] , l1Sig[49]);
29         hobclb CLB_48(AndEnabled[48], AndEnabled_out[48], lStep[47], lStep[48], rStep[4] , rStep[5] , l1Sig[48]);
30         hobclb CLB_47(AndEnabled[47], AndEnabled_out[47], lStep[46], lStep[47], rStep[5] , rStep[6] , l1Sig[47]);
31         hobclb CLB_46(AndEnabled[46], AndEnabled_out[46], lStep[45], lStep[46], rStep[6] , rStep[7] , l1Sig[46]);
32         hobclb CLB_45(AndEnabled[45], AndEnabled_out[45], lStep[44], lStep[45], rStep[7] , rStep[8] , l1Sig[45]);
33         hobclb CLB_44(AndEnabled[44], AndEnabled_out[44], lStep[43], lStep[44], rStep[8] , rStep[9] , l1Sig[44]);
34         hobclb CLB_43(AndEnabled[43], AndEnabled_out[43], lStep[42], lStep[43], rStep[9] , rStep[10], l1Sig[43]);
35         hobclb CLB_42(AndEnabled[42], AndEnabled_out[42], lStep[41], lStep[42], rStep[10], rStep[11], l1Sig[42]);
36         hobclb CLB_41(AndEnabled[41], AndEnabled_out[41], lStep[40], lStep[41], rStep[11], rStep[12], l1Sig[41]);
37         hobclb CLB_40(AndEnabled[40], AndEnabled_out[40], lStep[39], lStep[40], rStep[12], rStep[13], l1Sig[40]);
38         hobclb CLB_39(AndEnabled[39], AndEnabled_out[39], lStep[38], lStep[39], rStep[13], rStep[14], l1Sig[39]);
39         hobclb CLB_38(AndEnabled[38], AndEnabled_out[38], lStep[37], lStep[38], rStep[14], rStep[15], l1Sig[38]);
40         hobclb CLB_37(AndEnabled[37], AndEnabled_out[37], lStep[36], lStep[37], rStep[15], rStep[16], l1Sig[37]);
41         hobclb CLB_36(AndEnabled[36], AndEnabled_out[36], lStep[35], lStep[36], rStep[16], rStep[17], l1Sig[36]);
42         hobclb CLB_35(AndEnabled[35], AndEnabled_out[35], lStep[34], lStep[35], rStep[17], rStep[18], l1Sig[35]);
43         hobclb CLB_34(AndEnabled[34], AndEnabled_out[34], lStep[33], lStep[34], rStep[18], rStep[19], l1Sig[34]);
44         hobclb CLB_33(AndEnabled[33], AndEnabled_out[33], lStep[32], lStep[33], rStep[19], rStep[20], l1Sig[33]);
45         hobclb CLB_32(AndEnabled[32], AndEnabled_out[32], lStep[31], lStep[32], rStep[20], rStep[21], l1Sig[32]);
46         hobclb CLB_31(AndEnabled[31], AndEnabled_out[31], lStep[30], lStep[31], rStep[21], rStep[22], l1Sig[31]);
47         hobclb CLB_30(AndEnabled[30], AndEnabled_out[30], lStep[29], lStep[30], rStep[22], rStep[23], l1Sig[30]);
48         hobclb CLB_29(AndEnabled[29], AndEnabled_out[29], lStep[28], lStep[29], rStep[23], rStep[24], l1Sig[29]);
49         hobclb CLB_28(AndEnabled[28], AndEnabled_out[28], lStep[27], lStep[28], rStep[24], rStep[25], l1Sig[28]);
50         hobclb CLB_27(AndEnabled[27], AndEnabled_out[27], lStep[26], lStep[27], rStep[25], rStep[26], l1Sig[27]);
51         hobclb CLB_26(AndEnabled[26], AndEnabled_out[26], lStep[25], lStep[26], rStep[26], rStep[27], l1Sig[26]);
52         hobclb CLB_25(AndEnabled[25], AndEnabled_out[25], lStep[24], lStep[25], rStep[27], rStep[28], l1Sig[25]);
53         hobclb CLB_24(AndEnabled[24], AndEnabled_out[24], lStep[23], lStep[24], rStep[28], rStep[29], l1Sig[24]);
54         hobclb CLB_23(AndEnabled[23], AndEnabled_out[23], lStep[22], lStep[23], rStep[29], rStep[30], l1Sig[23]);
55         hobclb CLB_22(AndEnabled[22], AndEnabled_out[22], lStep[21], lStep[22], rStep[30], rStep[31], l1Sig[22]);
56         hobclb CLB_21(AndEnabled[21], AndEnabled_out[21], lStep[20], lStep[21], rStep[31], rStep[32], l1Sig[21]);
57         hobclb CLB_20(AndEnabled[20], AndEnabled_out[20], lStep[19], lStep[20], rStep[32], rStep[33], l1Sig[20]);
58         hobclb CLB_19(AndEnabled[19], AndEnabled_out[19], lStep[18], lStep[19], rStep[33], rStep[34], l1Sig[19]);
59         hobclb CLB_18(AndEnabled[18], AndEnabled_out[18], lStep[17], lStep[18], rStep[34], rStep[35], l1Sig[18]);
60         hobclb CLB_17(AndEnabled[17], AndEnabled_out[17], lStep[16], lStep[17], rStep[35], rStep[36], l1Sig[17]);
61         hobclb CLB_16(AndEnabled[16], AndEnabled_out[16], lStep[15], lStep[16], rStep[36], rStep[37], l1Sig[16]);
62         hobclb CLB_15(AndEnabled[15], AndEnabled_out[15], lStep[14], lStep[15], rStep[37], rStep[38], l1Sig[15]);
63         hobclb CLB_14(AndEnabled[14], AndEnabled_out[14], lStep[13], lStep[14], rStep[38], rStep[39], l1Sig[14]);
64         hobclb CLB_13(AndEnabled[13], AndEnabled_out[13], lStep[12], lStep[13], rStep[39], rStep[40], l1Sig[13]);
65         hobclb CLB_12(AndEnabled[12], AndEnabled_out[12], lStep[11], lStep[12], rStep[40], rStep[41], l1Sig[12]);
66         hobclb CLB_11(AndEnabled[11], AndEnabled_out[11], lStep[10], lStep[11], rStep[41], rStep[42], l1Sig[11]);
67         hobclb CLB_10(AndEnabled[10], AndEnabled_out[10], lStep[9] , lStep[10], rStep[42], rStep[43], l1Sig[10]);
68         hobclb CLB_09(AndEnabled[9], AndEnabled_out[9], lStep[8] , lStep[9] , rStep[43], rStep[44], l1Sig[9]);
69         hobclb CLB_08(AndEnabled[8], AndEnabled_out[8], lStep[7] , lStep[8] , rStep[44], rStep[45], l1Sig[8]);
70         hobclb CLB_07(AndEnabled[7], AndEnabled_out[7], lStep[6] , lStep[7] , rStep[45], rStep[46], l1Sig[7]);
71         hobclb CLB_06(AndEnabled[6], AndEnabled_out[6], lStep[5] , lStep[6] , rStep[46], rStep[47], l1Sig[6]);
72         hobclb CLB_05(AndEnabled[5], AndEnabled_out[5], lStep[4] , lStep[5] , rStep[47], rStep[48], l1Sig[5]);
```

```
73          hobclb CLB_04(AndEnabled[4], AndEnabled_out[4], IStep[3] , IStep[4] , rStep[48], rStep[49], I1Sig[4]);
74          hobclb CLB_03(AndEnabled[3], AndEnabled_out[3], IStep[2] , IStep[3] , rStep[49], rStep[50], I1Sig[3]);
75          hobclb CLB_02(AndEnabled[2], AndEnabled_out[2], IStep[1] , IStep[2] , rStep[50], rStep[51], I1Sig[2]);
76          hobclb CLB_01(AndEnabled[1], AndEnabled_out[1], IStep[0] , IStep[1] , rStep[51], rStep[52], I1Sig[1]);
77          lrlclb CLB_00(AndEnabled[0], AndEnabled_out[0], CapsEnabled[0],
78                                      CapsEnabled_out[0], IIn , IStep[0] , rStep[52], I1Sig[0]);
79
80          OrLut25 ORL1_27 (I1Sig[53], , I2Sig[27]);
81          OrLut25 ORL1_26 (I1Sig[51], I1Sig[52], I2Sig[26]);
82          OrLut25 ORL1_25 (I1Sig[49], I1Sig[50], I2Sig[25]);
83          OrLut25 ORL1_24 (I1Sig[47], I1Sig[48], I2Sig[24]);
84          OrLut25 ORL1_23 (I1Sig[45], I1Sig[46], I2Sig[23]);
85          OrLut25 ORL1_22 (I1Sig[43], I1Sig[44], I2Sig[22]);
86          OrLut25 ORL1_21 (I1Sig[41], I1Sig[42], I2Sig[21]);
87          OrLut25 ORL1_20 (I1Sig[39], I1Sig[40], I2Sig[20]);
88          OrLut25 ORL1_19 (I1Sig[37], I1Sig[38], I2Sig[19]);
89          OrLut25 ORL1_18 (I1Sig[35], I1Sig[36], I2Sig[18]);
90          OrLut25 ORL1_17 (I1Sig[33], I1Sig[34], I2Sig[17]);
91          OrLut25 ORL1_16 (I1Sig[31], I1Sig[32], I2Sig[16]);
92          OrLut25 ORL1_15 (I1Sig[29], I1Sig[30], I2Sig[15]);
93          OrLut25 ORL1_14 (I1Sig[27], I1Sig[28], I2Sig[14]);
94          OrLut25 ORL1_13 (I1Sig[25], I1Sig[26], I2Sig[13]);
95          OrLut25 ORL1_12 (I1Sig[23], I1Sig[24], I2Sig[12]);
96          OrLut25 ORL1_11 (I1Sig[21], I1Sig[22], I2Sig[11]);
97          OrLut25 ORL1_10 (I1Sig[19], I1Sig[20], I2Sig[10]);
98          OrLut25 ORL1_09 (I1Sig[17], I1Sig[18], I2Sig[9]);
99          OrLut25 ORL1_08 (I1Sig[15], I1Sig[16], I2Sig[8]);
100         OrLut25 ORL1_07 (I1Sig[13], I1Sig[14], I2Sig[7]);
101         OrLut25 ORL1_06 (I1Sig[11], I1Sig[12], I2Sig[6]);
102         OrLut25 ORL1_05 (I1Sig[9] , I1Sig[10], I2Sig[5]);
103         OrLut25 ORL1_04 (I1Sig[7] , I1Sig[8] , I2Sig[4]);
104         OrLut25 ORL1_03 (I1Sig[5] , I1Sig[6] , I2Sig[3]);
105         OrLut25 ORL1_02 (I1Sig[3] , I1Sig[4] , I2Sig[2]);
106         OrLut25 ORL1_01 (I1Sig[1] , I1Sig[2] , I2Sig[1]);
107         OrLut25 ORL1_00 ( , I1Sig[0], I2Sig[0]);
108         OrLut34 ORL2_13 (I2Sig[26], I2Sig[27], I3Sig[13]);
109         OrLut34 ORL2_12 (I2Sig[24], I2Sig[25], I3Sig[12]);
110         OrLut34 ORL2_11 (I2Sig[22], I2Sig[23], I3Sig[11]);
111         OrLut34 ORL2_10 (I2Sig[20], I2Sig[21], I3Sig[10]);
112         OrLut34 ORL2_09 (I2Sig[18], I2Sig[19], I3Sig[9]);
113         OrLut34 ORL2_08 (I2Sig[16], I2Sig[17], I3Sig[8]);
114         OrLut34 ORL2_07 (I2Sig[14], I2Sig[15], I3Sig[7]);
115         OrLut34 ORL2_06 (I2Sig[12], I2Sig[13], I3Sig[6]);
116         OrLut34 ORL2_05 (I2Sig[10], I2Sig[11], I3Sig[5]);
117         OrLut34 ORL2_04 (I2Sig[8] , I2Sig[9] , I3Sig[4]);
118         OrLut34 ORL2_03 (I2Sig[6] , I2Sig[7] , I3Sig[3]);
119         OrLut34 ORL2_02 (I2Sig[4] , I2Sig[5] , I3Sig[2]);
120         OrLut34 ORL2_01 (I2Sig[2] , I2Sig[3] , I3Sig[1]);
121         OrLut34 ORL2_00 (I2Sig[0] , I2Sig[1] , I3Sig[0]);
122         OrLut34 ORL3_06 (I3Sig[12], I3Sig[13], I4Sig[6]);
123         OrLut34 ORL3_05 (I3Sig[10], I3Sig[11], I4Sig[5]);
124         OrLut34 ORL3_04 (I3Sig[8] , I3Sig[9] , I4Sig[4]);
125         OrLut34 ORL3_03 (I3Sig[6] , I3Sig[7] , I4Sig[3]);
126         OrLut34 ORL3_02 (I3Sig[4] , I3Sig[5] , I4Sig[2]);
127         OrLut34 ORL3_01 (I3Sig[2] , I3Sig[3] , I4Sig[1]);
128         OrLut34 ORL3_00 (I3Sig[0] , I3Sig[1] , I4Sig[0]);
129         OrLut41 ORL4_03 (I4Sig[5] , I4Sig[6] , I5Sig[3]);
130         OrLut41 ORL4_02 (I4Sig[3] , I4Sig[4] , I5Sig[2]);
131         OrLut41 ORL4_01 (I4Sig[1] , I4Sig[2] , I5Sig[1]);
132         OrLut41 ORL4_00 ( , I4Sig[0] , I5Sig[0]);
133         OrLut24 ORL5_01 (I5Sig[2],I5Sig[3],I6Sig[1]);
134         OrLut24 ORL5_00 (I5Sig[0],I5Sig[1],I6Sig[0]);
135         OrLut54 ORL6_00 (I6Sig[0],I6Sig[1],orOut);
136         OrOutEn OutEnableHi (orOut,MTEnabled,MTOut);
137 endmodule
138
139         ...
140         Quellcode der anderen 63 Meantimer befindet
141         sich auf der beigefügten CD
142         ...
```

Listing A.4: matrix.v

```
1  `timescale 1ns / 1ps
2
3  module matrix_row_clb (IN, OR_IN, OR_OUT);
4
5          (* S="TRUE" *) input wire IN;
6          (* S="TRUE" *) input wire OR_IN;
7          (* S="TRUE" *) output wire OR_OUT;
8          (* S="TRUE" *) wire leap;
9
10         (* LOCK_PINS="ALL" *)
11         LUT6 #(.INIT(64'hFFFF_FFFF_F0F0_F0F0)) H2_LUT (
12                 .O(OR_OUT),
13                 .I2(OR_IN),
14                 .I5(leap)
15         );
16
17         (* LOCK_PINS="ALL" *)
18         LUT6 #(.INIT(64'hF0F0_F0F0_F0F0_F0F0)) H1_LUT (
19                 .O(leap),
20                 .I2(IN)
21         );
22
23 endmodule
24
25
26 module matrix_col_clb (IN, OR_IN, OR_OUT);
27
28         (* S="TRUE" *) input wire IN;
29         (* S="TRUE" *) input wire OR_IN;
30         (* S="TRUE" *) output wire OR_OUT;
31         (* S="TRUE" *) wire leap;
32
33         (* LOCK_PINS="ALL" *)
34         LUT6 #(.INIT(64'hFFFF_F0F0_FFFF_F0F0)) H1_LUT (
35                 .O(OR_OUT),
36                 .I2(OR_IN),
37                 .I4(leap)
38         );
39
40         (* LOCK_PINS="ALL" *)
41         LUT6 #(.INIT(64'hF0F0_F0F0_F0F0_F0F0)) H2_LUT (
42                 .O(leap),
43                 .I2(IN)
44         );
45
46 endmodule
47
48
49 module matrix_clb(h1_In,h1_Out,h2_In,h2_Out,trigger_in,trigger_out, MatrixEnReg);
50
51         (* S="TRUE" *) input wire h1_In;
52         (* S="TRUE" *) output wire h1_Out;
53
54         (* S="TRUE" *) input wire h2_In;
55         (* S="TRUE" *) output wire h2_Out;
56
57         (* S="TRUE" *) wire h1_and;
58         (* S="TRUE" *) wire h2_and;
59         (* S="TRUE" *) input wire trigger_in;
60         (* S="TRUE" *) output wire trigger_out;
61
62         input wire MatrixEnReg;
63
64         (* LOCK_PINS="ALL" *)
65         LUT6_2 #(.INIT(64'hF0F0_F0F0_F0F0_F0F0)) H1_LUT (
66                 .O6(h1_Out),
67                 .O5(h1_and),
68                 .I2(h1_In),
69                 .I5(1'b1)
70         );
71
72         (* LOCK_PINS="ALL" *)
```

```
73          LUT6_2 #(.INIT(64'hF0F0_F0F0_F0F0_F0F0)) H2_LUT (
74                  .O6(h2_Out),
75                  .O5(h2_and),
76                  .I2(h2_In),
77                  .I5(1'b1)
78          );
79
80          (* LOCK_PINS="ALL" *)
81          LUT6 #(.INIT(64'hFFFF_CC00_FFFF_0000)) K_LUT ( // => I4 | (I3 & I5 & I1)
82                  .O(trigger_out),
83                  .I5(h2_and),
84                  .I4(trigger_in),
85                  .I3(MatrixEnReg),
86                  .I1(h1_and)
87          );
88
89  endmodule
90
91
92  module matrix_row(IN_H1,OUT_H1,IN_H2,OUT_H2,IN_TRIGGER,OUT_TRIGGER,MatrixEnReg);
93
94          input wire [31:0] MatrixEnReg;
95
96          input wire [31:0] IN_H1;
97          output wire [31:0] OUT_H1;
98
99          input wire IN_H2;
100         output wire OUT_H2;
101
102         input wire [31:0] IN_TRIGGER;
103         output wire [31:0] OUT_TRIGGER;
104
105         wire [30:0] h2;
106
107         matrix_clb matrix_clb_0 (IN_H1[0], OUT_H1[0], IN_H2, h2[0], IN_TRIGGER[0],
108                                                 OUT_TRIGGER[0], MatrixEnReg[0]);
109         matrix_clb matrix_clb_1 (IN_H1[1], OUT_H1[1], h2[0], h2[1], IN_TRIGGER[1],
110                                                 OUT_TRIGGER[1], MatrixEnReg[1]);
111         matrix_clb matrix_clb_2 (IN_H1[2], OUT_H1[2], h2[1], h2[2], IN_TRIGGER[2],
112                                                 OUT_TRIGGER[2], MatrixEnReg[2]);
113         matrix_clb matrix_clb_3 (IN_H1[3], OUT_H1[3], h2[2], h2[3], IN_TRIGGER[3],
114                                                 OUT_TRIGGER[3], MatrixEnReg[3]);
115         matrix_clb matrix_clb_4 (IN_H1[4], OUT_H1[4], h2[3], h2[4], IN_TRIGGER[4],
116                                                 OUT_TRIGGER[4], MatrixEnReg[4]);
117         matrix_clb matrix_clb_5 (IN_H1[5], OUT_H1[5], h2[4], h2[5], IN_TRIGGER[5],
118                                                 OUT_TRIGGER[5], MatrixEnReg[5]);
119         matrix_clb matrix_clb_6 (IN_H1[6], OUT_H1[6], h2[5], h2[6], IN_TRIGGER[6],
120                                                 OUT_TRIGGER[6], MatrixEnReg[6]);
121         matrix_clb matrix_clb_7 (IN_H1[7], OUT_H1[7], h2[6], h2[7], IN_TRIGGER[7],
122                                                 OUT_TRIGGER[7], MatrixEnReg[7]);
123         matrix_clb matrix_clb_8 (IN_H1[8], OUT_H1[8], h2[7], h2[8], IN_TRIGGER[8],
124                                                 OUT_TRIGGER[8], MatrixEnReg[8]);
125         matrix_clb matrix_clb_9 (IN_H1[9], OUT_H1[9], h2[8], h2[9], IN_TRIGGER[9],
126                                                 OUT_TRIGGER[9], MatrixEnReg[9]);
127         matrix_clb matrix_clb_10(IN_H1[10], OUT_H1[10], h2[9], h2[10], IN_TRIGGER[10],
128                                                 OUT_TRIGGER[10], MatrixEnReg[10]);
129         matrix_clb matrix_clb_11(IN_H1[11], OUT_H1[11], h2[10], h2[11], IN_TRIGGER[11],
130                                                 OUT_TRIGGER[11], MatrixEnReg[11]);
131         matrix_clb matrix_clb_12(IN_H1[12], OUT_H1[12], h2[11], h2[12], IN_TRIGGER[12],
132                                                 OUT_TRIGGER[12], MatrixEnReg[12]);
133         matrix_clb matrix_clb_13(IN_H1[13], OUT_H1[13], h2[12], h2[13], IN_TRIGGER[13],
134                                                 OUT_TRIGGER[13], MatrixEnReg[13]);
135         matrix_clb matrix_clb_14(IN_H1[14], OUT_H1[14], h2[13], h2[14], IN_TRIGGER[14],
136                                                 OUT_TRIGGER[14], MatrixEnReg[14]);
137         matrix_clb matrix_clb_15(IN_H1[15], OUT_H1[15], h2[14], h2[15], IN_TRIGGER[15],
138                                                 OUT_TRIGGER[15], MatrixEnReg[15]);
139         matrix_clb matrix_clb_16(IN_H1[16], OUT_H1[16], h2[15], h2[16], IN_TRIGGER[16],
140                                                 OUT_TRIGGER[16], MatrixEnReg[16]);
141         matrix_clb matrix_clb_17(IN_H1[17], OUT_H1[17], h2[16], h2[17], IN_TRIGGER[17],
142                                                 OUT_TRIGGER[17], MatrixEnReg[17]);
143         matrix_clb matrix_clb_18(IN_H1[18], OUT_H1[18], h2[17], h2[18], IN_TRIGGER[18],
144                                                 OUT_TRIGGER[18], MatrixEnReg[18]);
145         matrix_clb matrix_clb_19(IN_H1[19], OUT_H1[19], h2[18], h2[19], IN_TRIGGER[19],
146                                                 OUT_TRIGGER[19], MatrixEnReg[19]);
```

```
147         matrix_clb matrix_clb_20(IN_H1[20], OUT_H1[20], h2[19], h2[20], IN_TRIGGER[20],
148                                                 OUT_TRIGGER[20], MatrixEnReg[20]);
149         matrix_clb matrix_clb_21(IN_H1[21], OUT_H1[21], h2[20], h2[21], IN_TRIGGER[21],
150                                                 OUT_TRIGGER[21], MatrixEnReg[21]);
151         matrix_clb matrix_clb_22(IN_H1[22], OUT_H1[22], h2[21], h2[22], IN_TRIGGER[22],
152                                                 OUT_TRIGGER[22], MatrixEnReg[22]);
153         matrix_clb matrix_clb_23(IN_H1[23], OUT_H1[23], h2[22], h2[23], IN_TRIGGER[23],
154                                                 OUT_TRIGGER[23], MatrixEnReg[23]);
155         matrix_clb matrix_clb_24(IN_H1[24], OUT_H1[24], h2[23], h2[24], IN_TRIGGER[24],
156                                                 OUT_TRIGGER[24], MatrixEnReg[24]);
157         matrix_clb matrix_clb_25(IN_H1[25], OUT_H1[25], h2[24], h2[25], IN_TRIGGER[25],
158                                                 OUT_TRIGGER[25], MatrixEnReg[25]);
159         matrix_clb matrix_clb_26(IN_H1[26], OUT_H1[26], h2[25], h2[26], IN_TRIGGER[26],
160                                                 OUT_TRIGGER[26], MatrixEnReg[26]);
161         matrix_clb matrix_clb_27(IN_H1[27], OUT_H1[27], h2[26], h2[27], IN_TRIGGER[27],
162                                                 OUT_TRIGGER[27], MatrixEnReg[27]);
163         matrix_clb matrix_clb_28(IN_H1[28], OUT_H1[28], h2[27], h2[28], IN_TRIGGER[28],
164                                                 OUT_TRIGGER[28], MatrixEnReg[28]);
165         matrix_clb matrix_clb_29(IN_H1[29], OUT_H1[29], h2[28], h2[29], IN_TRIGGER[29],
166                                                 OUT_TRIGGER[29], MatrixEnReg[29]);
167         matrix_clb matrix_clb_30(IN_H1[30], OUT_H1[30], h2[29], h2[30], IN_TRIGGER[30],
168                                                 OUT_TRIGGER[30], MatrixEnReg[30]);
169         matrix_clb matrix_clb_31(IN_H1[31], OUT_H1[31], h2[30], OUT_H2, IN_TRIGGER[31],
170                                                 OUT_TRIGGER[31], MatrixEnReg[31]);
171
172 endmodule
173
174
175
176 module matrix_or_row(IN,OUT);
177
178         input wire [31:0] IN;
179         output wire OUT;
180
181         wire [30:0] h2;
182
183         matrix_row_clb matrix_clb_0 (IN[0],  , h2[0]);
184         matrix_row_clb matrix_clb_1 (IN[1], h2[0], h2[1]);
185         matrix_row_clb matrix_clb_2 (IN[2], h2[1], h2[2]);
186         matrix_row_clb matrix_clb_3 (IN[3], h2[2], h2[3]);
187         matrix_row_clb matrix_clb_4 (IN[4], h2[3], h2[4]);
188         matrix_row_clb matrix_clb_5 (IN[5], h2[4], h2[5]);
189         matrix_row_clb matrix_clb_6 (IN[6], h2[5], h2[6]);
190         matrix_row_clb matrix_clb_7 (IN[7], h2[6], h2[7]);
191         matrix_row_clb matrix_clb_8 (IN[8], h2[7], h2[8]);
192         matrix_row_clb matrix_clb_9 (IN[9], h2[8], h2[9]);
193         matrix_row_clb matrix_clb_10(IN[10], h2[9], h2[10]);
194         matrix_row_clb matrix_clb_11(IN[11], h2[10], h2[11]);
195         matrix_row_clb matrix_clb_12(IN[12], h2[11], h2[12]);
196         matrix_row_clb matrix_clb_13(IN[13], h2[12], h2[13]);
197         matrix_row_clb matrix_clb_14(IN[14], h2[13], h2[14]);
198         matrix_row_clb matrix_clb_15(IN[15], h2[14], h2[15]);
199         matrix_row_clb matrix_clb_16(IN[16], h2[15], h2[16]);
200         matrix_row_clb matrix_clb_17(IN[17], h2[16], h2[17]);
201         matrix_row_clb matrix_clb_18(IN[18], h2[17], h2[18]);
202         matrix_row_clb matrix_clb_19(IN[19], h2[18], h2[19]);
203         matrix_row_clb matrix_clb_20(IN[20], h2[19], h2[20]);
204         matrix_row_clb matrix_clb_21(IN[21], h2[20], h2[21]);
205         matrix_row_clb matrix_clb_22(IN[22], h2[21], h2[22]);
206         matrix_row_clb matrix_clb_23(IN[23], h2[22], h2[23]);
207         matrix_row_clb matrix_clb_24(IN[24], h2[23], h2[24]);
208         matrix_row_clb matrix_clb_25(IN[25], h2[24], h2[25]);
209         matrix_row_clb matrix_clb_26(IN[26], h2[25], h2[26]);
210         matrix_row_clb matrix_clb_27(IN[27], h2[26], h2[27]);
211         matrix_row_clb matrix_clb_28(IN[28], h2[27], h2[28]);
212         matrix_row_clb matrix_clb_29(IN[29], h2[28], h2[29]);
213         matrix_row_clb matrix_clb_30(IN[30], h2[29], h2[30]);
214         matrix_row_clb matrix_clb_31(IN[31], h2[30], OUT);
215
216 endmodule
217
218
219 module matrix_or_col (IN,OUT);
220
```

```
221         input wire [31:0] IN;
222         output wire OUT;
223
224         wire [30:0] h1;
225
226         matrix_col_clb matrix_col_31(IN[31], h1[30], OUT);
227         matrix_col_clb matrix_col_30(IN[30], h1[29], h1[30]);
228         matrix_col_clb matrix_col_29(IN[29], h1[28], h1[29]);
229         matrix_col_clb matrix_col_28(IN[28], h1[27], h1[28]);
230         matrix_col_clb matrix_col_27(IN[27], h1[26], h1[27]);
231         matrix_col_clb matrix_col_26(IN[26], h1[25], h1[26]);
232         matrix_col_clb matrix_col_25(IN[25], h1[24], h1[25]);
233         matrix_col_clb matrix_col_24(IN[24], h1[23], h1[24]);
234         matrix_col_clb matrix_col_23(IN[23], h1[22], h1[23]);
235         matrix_col_clb matrix_col_22(IN[22], h1[21], h1[22]);
236         matrix_col_clb matrix_col_21(IN[21], h1[20], h1[21]);
237         matrix_col_clb matrix_col_20(IN[20], h1[19], h1[20]);
238         matrix_col_clb matrix_col_19(IN[19], h1[18], h1[19]);
239         matrix_col_clb matrix_col_18(IN[18], h1[17], h1[18]);
240         matrix_col_clb matrix_col_17(IN[17], h1[16], h1[17]);
241         matrix_col_clb matrix_col_16(IN[16], h1[15], h1[16]);
242         matrix_col_clb matrix_col_15(IN[15], h1[14], h1[15]);
243         matrix_col_clb matrix_col_14(IN[14], h1[13], h1[14]);
244         matrix_col_clb matrix_col_13(IN[13], h1[12], h1[13]);
245         matrix_col_clb matrix_col_12(IN[12], h1[11], h1[12]);
246         matrix_col_clb matrix_col_11(IN[11], h1[10], h1[11]);
247         matrix_col_clb matrix_col_10(IN[10], h1[9], h1[10]);
248         matrix_col_clb matrix_col_9 (IN[9], h1[8], h1[9]);
249         matrix_col_clb matrix_col_8 (IN[8], h1[7], h1[8]);
250         matrix_col_clb matrix_col_7 (IN[7], h1[6], h1[7]);
251         matrix_col_clb matrix_col_6 (IN[6], h1[5], h1[6]);
252         matrix_col_clb matrix_col_5 (IN[5], h1[4], h1[5]);
253         matrix_col_clb matrix_col_4 (IN[4], h1[3], h1[4]);
254         matrix_col_clb matrix_col_3 (IN[3], h1[2], h1[3]);
255         matrix_col_clb matrix_col_2 (IN[2], h1[1], h1[2]);
256         matrix_col_clb matrix_col_1 (IN[1], h1[0], h1[1]);
257         matrix_col_clb matrix_col_0 (IN[0], , h1[0]);
258
259 endmodule
260
261
262
263 module matrix (IN_H1, IN_H2, TRG_MATRIX, PURE_TG, TRG_H1, TRG_H2,
264                             MatrixEnReg00, MatrixEnReg01, MatrixEnReg02, MatrixEnReg03, MatrixEnReg04,
265                             MatrixEnReg05, MatrixEnReg06, MatrixEnReg07, MatrixEnReg08, MatrixEnReg09,
266                             MatrixEnReg10, MatrixEnReg11, MatrixEnReg12, MatrixEnReg13, MatrixEnReg14,
267                             MatrixEnReg15, MatrixEnReg16, MatrixEnReg17, MatrixEnReg18, MatrixEnReg19,
268                             MatrixEnReg20, MatrixEnReg21, MatrixEnReg22, MatrixEnReg23, MatrixEnReg24,
269                             MatrixEnReg25, MatrixEnReg26, MatrixEnReg27, MatrixEnReg28, MatrixEnReg29,
270                             MatrixEnReg30, MatrixEnReg31);
271
272         input wire [31:0] MatrixEnReg00;
273         input wire [31:0] MatrixEnReg01;
274         input wire [31:0] MatrixEnReg02;
275         input wire [31:0] MatrixEnReg03;
276         input wire [31:0] MatrixEnReg04;
277         input wire [31:0] MatrixEnReg05;
278         input wire [31:0] MatrixEnReg06;
279         input wire [31:0] MatrixEnReg07;
280         input wire [31:0] MatrixEnReg08;
281         input wire [31:0] MatrixEnReg09;
282         input wire [31:0] MatrixEnReg10;
283         input wire [31:0] MatrixEnReg11;
284         input wire [31:0] MatrixEnReg12;
285         input wire [31:0] MatrixEnReg13;
286         input wire [31:0] MatrixEnReg14;
287         input wire [31:0] MatrixEnReg15;
288         input wire [31:0] MatrixEnReg16;
289         input wire [31:0] MatrixEnReg17;
290         input wire [31:0] MatrixEnReg18;
291         input wire [31:0] MatrixEnReg19;
292         input wire [31:0] MatrixEnReg20;
293         input wire [31:0] MatrixEnReg21;
294         input wire [31:0] MatrixEnReg22;
```

```
295         input wire [31:0] MatrixEnReg23;
296         input wire [31:0] MatrixEnReg24;
297         input wire [31:0] MatrixEnReg25;
298         input wire [31:0] MatrixEnReg26;
299         input wire [31:0] MatrixEnReg27;
300         input wire [31:0] MatrixEnReg28;
301         input wire [31:0] MatrixEnReg29;
302         input wire [31:0] MatrixEnReg30;
303         input wire [31:0] MatrixEnReg31;
304
305         input wire [31:0] IN_H1, IN_H2;
306         output wire TRG_MATRIX, PURE_TG, TRG_H1, TRG_H2;
307
308         wire [31:0] h1_31;
309         wire [31:0] h1_30;
310         wire [31:0] h1_29;
311         wire [31:0] h1_28;
312         wire [31:0] h1_27;
313         wire [31:0] h1_26;
314         wire [31:0] h1_25;
315         wire [31:0] h1_24;
316         wire [31:0] h1_23;
317         wire [31:0] h1_22;
318         wire [31:0] h1_21;
319         wire [31:0] h1_20;
320         wire [31:0] h1_19;
321         wire [31:0] h1_18;
322         wire [31:0] h1_17;
323         wire [31:0] h1_16;
324         wire [31:0] h1_15;
325         wire [31:0] h1_14;
326         wire [31:0] h1_13;
327         wire [31:0] h1_12;
328         wire [31:0] h1_11;
329         wire [31:0] h1_10;
330         wire [31:0] h1_9;
331         wire [31:0] h1_8;
332         wire [31:0] h1_7;
333         wire [31:0] h1_6;
334         wire [31:0] h1_5;
335         wire [31:0] h1_4;
336         wire [31:0] h1_3;
337         wire [31:0] h1_2;
338         wire [31:0] h1_1;
339         wire [31:0] h1_0;
340
341         wire [31:0] tg_31;
342         wire [31:0] tg_30;
343         wire [31:0] tg_29;
344         wire [31:0] tg_28;
345         wire [31:0] tg_27;
346         wire [31:0] tg_26;
347         wire [31:0] tg_25;
348         wire [31:0] tg_24;
349         wire [31:0] tg_23;
350         wire [31:0] tg_22;
351         wire [31:0] tg_21;
352         wire [31:0] tg_20;
353         wire [31:0] tg_19;
354         wire [31:0] tg_18;
355         wire [31:0] tg_17;
356         wire [31:0] tg_16;
357         wire [31:0] tg_15;
358         wire [31:0] tg_14;
359         wire [31:0] tg_13;
360         wire [31:0] tg_12;
361         wire [31:0] tg_11;
362         wire [31:0] tg_10;
363         wire [31:0] tg_9;
364         wire [31:0] tg_8;
365         wire [31:0] tg_7;
366         wire [31:0] tg_6;
367         wire [31:0] tg_5;
368         wire [31:0] tg_4;
```

```
369         wire [31:0] tg_3;
370         wire [31:0] tg_2;
371         wire [31:0] tg_1;
372         wire [31:0] tg_0;
373
374         wire [31:0] OUT_H2;
375
376         (* S="TRUE" *) wire PURE_H1;
377
378         matrix_or_row matrix_row_33(tg_31,PURE_TG);
379         matrix_or_row matrix_row_32(h1_31,PURE_H1);
380
381         matrix_row matrix_row_31 (h1_30, h1_31, IN_H2[31], OUT_H2[31], tg_30, tg_31, MatrixEnReg31);
382         matrix_row matrix_row_30 (h1_29, h1_30, IN_H2[30], OUT_H2[30], tg_29, tg_30, MatrixEnReg30);
383         matrix_row matrix_row_29 (h1_28, h1_29, IN_H2[29], OUT_H2[29], tg_28, tg_29, MatrixEnReg29);
384         matrix_row matrix_row_28 (h1_27, h1_28, IN_H2[28], OUT_H2[28], tg_27, tg_28, MatrixEnReg28);
385         matrix_row matrix_row_27 (h1_26, h1_27, IN_H2[27], OUT_H2[27], tg_26, tg_27, MatrixEnReg27);
386         matrix_row matrix_row_26 (h1_25, h1_26, IN_H2[26], OUT_H2[26], tg_25, tg_26, MatrixEnReg26);
387         matrix_row matrix_row_25 (h1_24, h1_25, IN_H2[25], OUT_H2[25], tg_24, tg_25, MatrixEnReg25);
388         matrix_row matrix_row_24 (h1_23, h1_24, IN_H2[24], OUT_H2[24], tg_23, tg_24, MatrixEnReg24);
389         matrix_row matrix_row_23 (h1_22, h1_23, IN_H2[23], OUT_H2[23], tg_22, tg_23, MatrixEnReg23);
390         matrix_row matrix_row_22 (h1_21, h1_22, IN_H2[22], OUT_H2[22], tg_21, tg_22, MatrixEnReg22);
391         matrix_row matrix_row_21 (h1_20, h1_21, IN_H2[21], OUT_H2[21], tg_20, tg_21, MatrixEnReg21);
392         matrix_row matrix_row_20 (h1_19, h1_20, IN_H2[20], OUT_H2[20], tg_19, tg_20, MatrixEnReg20);
393         matrix_row matrix_row_19 (h1_18, h1_19, IN_H2[19], OUT_H2[19], tg_18, tg_19, MatrixEnReg19);
394         matrix_row matrix_row_18 (h1_17, h1_18, IN_H2[18], OUT_H2[18], tg_17, tg_18, MatrixEnReg18);
395         matrix_row matrix_row_17 (h1_16, h1_17, IN_H2[17], OUT_H2[17], tg_16, tg_17, MatrixEnReg17);
396         matrix_row matrix_row_16 (h1_15, h1_16, IN_H2[16], OUT_H2[16], tg_15, tg_16, MatrixEnReg16);
397         matrix_row matrix_row_15 (h1_14, h1_15, IN_H2[15], OUT_H2[15], tg_14, tg_15, MatrixEnReg15);
398         matrix_row matrix_row_14 (h1_13, h1_14, IN_H2[14], OUT_H2[14], tg_13, tg_14, MatrixEnReg14);
399         matrix_row matrix_row_13 (h1_12, h1_13, IN_H2[13], OUT_H2[13], tg_12, tg_13, MatrixEnReg13);
400         matrix_row matrix_row_12 (h1_11, h1_12, IN_H2[12], OUT_H2[12], tg_11, tg_12, MatrixEnReg12);
401         matrix_row matrix_row_11 (h1_10, h1_11, IN_H2[11], OUT_H2[11], tg_10, tg_11, MatrixEnReg11);
402         matrix_row matrix_row_10 (h1_9, h1_10, IN_H2[10], OUT_H2[10], tg_9, tg_10, MatrixEnReg10);
403         matrix_row matrix_row_9 (h1_8, h1_9, IN_H2[9], OUT_H2[9], tg_8, tg_9, MatrixEnReg09);
404         matrix_row matrix_row_8 (h1_7, h1_8, IN_H2[8], OUT_H2[8], tg_7, tg_8, MatrixEnReg08);
405         matrix_row matrix_row_7 (h1_6, h1_7, IN_H2[7], OUT_H2[7], tg_6, tg_7, MatrixEnReg07);
406         matrix_row matrix_row_6 (h1_5, h1_6, IN_H2[6], OUT_H2[6], tg_5, tg_6, MatrixEnReg06);
407         matrix_row matrix_row_5 (h1_4, h1_5, IN_H2[5], OUT_H2[5], tg_4, tg_5, MatrixEnReg05);
408         matrix_row matrix_row_4 (h1_3, h1_4, IN_H2[4], OUT_H2[4], tg_3, tg_4, MatrixEnReg04);
409         matrix_row matrix_row_3 (h1_2, h1_3, IN_H2[3], OUT_H2[3], tg_2, tg_3, MatrixEnReg03);
410         matrix_row matrix_row_2 (h1_1, h1_2, IN_H2[2], OUT_H2[2], tg_1, tg_2, MatrixEnReg02);
411         matrix_row matrix_row_1 (h1_0, h1_1, IN_H2[1], OUT_H2[1], tg_0, tg_1, MatrixEnReg01);
412         matrix_row matrix_row_0 (IN_H1, h1_0, IN_H2[0], OUT_H2[0], , tg_0, MatrixEnReg00);
413
414         matrix_or_col matrix_col (OUT_H2, TRG_H2);
415
416         (* S="TRUE" *) wire h1step_1;
417         (* S="TRUE" *) wire h1step_2;
418         (* S="TRUE" *) wire [9:0] trg_delay;
419         (* S="TRUE" *) wire trg_reset;
420
421
422         (* LOCK_PINS="ALL" *)
423         LUT6_2 #(.INIT(64'hF0F0_F0F0_F0F0_F0F0)) MATRIX_LUT_H1_0 (
424                 .O6(TRG_H1),
425                 .O5(h1step_1),
426                 .I2(PURE_H1),
427                 .I5(1'b1)
428         );
429
430         (* LOCK_PINS="ALL" *)
431         LUT6 #(.INIT(64'hF0F0_F0F0_F0F0_F0F0)) MATRIX_LUT_H1_1 (
432                 .O(h1step_2),
433                 .I2(h1step_1)
434         );
435
436
437         (* LOCK_PINS="ALL" *)
438         LUT6 #(.INIT(64'hF0F0_F0F0_0000_0000)) MATRIX_LUT_TG (
439                 .O(TRG_MATRIX), //trg_delay[0]),
440                 .I2(h1step_2),
441                 .I5(PURE_TG)
442         );
```

```
443
444
445         FDCPE_1 #(.INIT(1'b0)) TRG_SIGNAL_GEN (
446                 .Q(trg_delay[1]),
447                 .C(trg_delay[0]),
448                 .CE(1'b1),
449                 .CLR(trg_reset),
450                 .D(1'b1),
451                 .PRE(1'b0)
452         );
453
454         (* LOCK_PINS="ALL" *)
455         LUT6 #(.INIT(64'hAAAA_AAAA_AAAA_AAAA)) TRG_DELAY_12 (
456                 .I0(trg_delay[1]),
457                 .O(trg_delay[2])
458         );
459
460         (* LOCK_PINS="ALL" *)
461         LUT6 #(.INIT(64'hAAAA_AAAA_AAAA_AAAA)) TRG_DELAY_23 (
462                 .I0(trg_delay[2]),
463                 .O(trg_delay[3])
464         );
465
466         (* LOCK_PINS="ALL" *)
467         LUT6 #(.INIT(64'hAAAA_AAAA_AAAA_AAAA)) TRG_DELAY_34 (
468                 .I0(trg_delay[3]),
469                 .O(trg_delay[4])
470         );
471
472         (* LOCK_PINS="ALL" *)
473         LUT6 #(.INIT(64'hAAAA_AAAA_AAAA_AAAA)) TRG_DELAY_45 (
474                 .I0(trg_delay[4]),
475                 .O(trg_delay[5])
476         );
477
478         (* LOCK_PINS="ALL" *)
479         LUT6 #(.INIT(64'hAAAA_AAAA_AAAA_AAAA)) TRG_DELAY_56 (
480                 .I0(trg_delay[5]),
481                 .O(trg_delay[6])
482         );
483
484         (* LOCK_PINS="ALL" *)
485         LUT6 #(.INIT(64'hAAAA_AAAA_AAAA_AAAA)) TRG_DELAY_67 (
486                 .I0(trg_delay[6]),
487                 .O(trg_delay[7])
488         );
489
490 endmodule
```

B. User Constraints

Mit den User Constraints werden die Positionen der LUTs auf dem FPGA und das Routing der Signalpfade definiert. Die Constraints sind in den folgenden fünf Dateien organisiert:

- `GandalfBoard.ucf`: Definiert die Input- und Output-Pins des FPGAs entsprechend dem GANDALF-Layout.
- `MT_common.ucf`: Definiert die relativen Positionen aller Elemente der Meantimer
- `handroutet.ucf`: Definiert die Verbindungen von den Input-Pads zu den Meantimer-Eingängen, damit die Differenz zwischen den linken und rechten Signalen minimal ist.
- `matrix.ucf`: Definiert die Positionen aller Matrix-Elemente und die entsprechenden Signalpfade
- `generated.ucf`: Diese Datei wird durch die automatisierte Laufzeitanalyse erzeugt und enthält die kritischen Pfade.

Diese Dateien befinden sich auch auf der beigefügten Daten-CD.

Listing B.1: GandalfBoard.ucf

```
#these constraints define the input and output pins of the fpga
#they are all taken from Freiburg (GANDALF engineers)

#Default Port Definition
NET "CONN_LN[*]" DIFF_TERM = TRUE | IOBDELAY = NONE | IOSTANDARD = LVDS_25;
NET "CONN_LP[*]" DIFF_TERM = TRUE | IOBDELAY = NONE | IOSTANDARD = LVDS_25;
NET "CONN_RN[*]" DIFF_TERM = TRUE | IOBDELAY = NONE | IOSTANDARD = LVDS_25;
NET "CONN_RP[*]" DIFF_TERM = TRUE | IOBDELAY = NONE | IOSTANDARD = LVDS_25;

#CLOCKS
NET "CLK_40MHZ_VDSP" LOC = AH27;
NET "CLK_40MHZ_VDSP" IOSTANDARD = LVTTL | CLOCK_DEDICATED_ROUTE = FALSE;

#NIM Ports
NET "CONN_IA" LOC = AC28 | IOSTANDARD = LVTTL | SLOW;
NET "CONN_OA1" LOC = AF28 | IOSTANDARD = LVTTL | SLOW; #TRG_OUT1_A
NET "CONN_OA2" LOC = AA28 | IOSTANDARD = LVTTL | SLOW; #TRG_OUT2_A

NET "CONN_IB" LOC = AD24 | IOSTANDARD = LVTTL | SLOW;
NET "CONN_OB1" LOC = AF24 | IOSTANDARD = LVTTL | SLOW; #TRG_OUT1_B
NET "CONN_OB2" LOC = AD25 | IOSTANDARD = LVTTL | SLOW; #TRG_OUT2_B

###DSP <> CPLD transfer
NET "VA_Write" LOC = AD19;
NET "VA_Strobe" LOC = AE19;
NET "VA_Ready" LOC = AE17;
NET "VA_Control" LOC = AF16;
NET "VA_uBlaze" LOC = AD20;
NET "VA_FifoFull" LOC = AE21;
NET "VA_FifoEmpty" LOC = AE16;
NET "VA_Reset" LOC = AF15 |IOSTANDARD = LVTTL;
NET "VA_*" IOSTANDARD = "LVTTL";

NET "VD[0]" LOC = L21;
NET "VD[1]" LOC = L20;
NET "VD[2]" LOC = L15;
NET "VD[3]" LOC = L16;
NET "VD[4]" LOC = J22;
NET "VD[5]" LOC = K21;
NET "VD[6]" LOC = K16;
NET "VD[7]" LOC = J15;
NET "VD[8]" LOC = G22;
NET "VD[9]" LOC = H22;
NET "VD[10]" LOC = L14;
NET "VD[11]" LOC = K14;
NET "VD[12]" LOC = K23;
NET "VD[13]" LOC = K22;
NET "VD[14]" LOC = J12;
NET "VD[15]" LOC = H12;
NET "VD[16]" LOC = G23;
NET "VD[17]" LOC = H23;
NET "VD[18]" LOC = K13;
NET "VD[19]" LOC = K12;
NET "VD[20]" LOC = AE13;
NET "VD[21]" LOC = AE12;
NET "VD[22]" LOC = AF23;
NET "VD[23]" LOC = AG23;
NET "VD[24]" LOC = AF13;
NET "VD[25]" LOC = AG12;
NET "VD[26]" LOC = AE22;
NET "VD[27]" LOC = AE23;
NET "VD[28]" LOC = AE14;
NET "VD[29]" LOC = AF14;
NET "VD[30]" LOC = AF20;
NET "VD[31]" LOC = AF21;
NET "VD[*]" IOSTANDARD = "LVTTL";
```

Listing B.2: MT_common.ucf

```
#these constraints are common to all MTs
#this is possible due to relative location constraints (RLOC)
#each MT sets its own origin

INST "AndEnabled_0*" AREA_GROUP=MostLeft;
INST "CapsEnabled_0*" AREA_GROUP=MostLeft;
INST "MTEnabled_*" AREA_GROUP=MostLeft;
AREA_GROUP "MostLeft" RANGE=SLICE_X0Y69:SLICE_X1Y96;

INST "output_choice_*" AREA_GROUP=VME;
INST "input_choice_*" AREA_GROUP=VME;
INST "current_*" AREA_GROUP=VME;
INST "next_*" AREA_GROUP=VME;
INST "signal_state_*" AREA_GROUP=VME;
INST "delay_rst_*" AREA_GROUP=VME;
INST "config_mem_BRAM_EN" AREA_GROUP=VME;
INST "BRAM_delayinfo_*" AREA_GROUP=VME;
INST "*/FastReg*" AREA_GROUP=VME;
AREA_GROUP "VME" RANGE=SLICE_X2Y67:SLICE_X7Y98;

INST "MatrixEnReg*" AREA_GROUP=MatrixEnReg;
INST "cpld_if_1/*" AREA_GROUP=MatrixEnReg;
INST "*/SwDCtrl*" AREA_GROUP=MatrixEnReg;
AREA_GROUP "MatrixEnReg" RANGE=SLICE_X14Y65:SLICE_X77Y97;

INST "signal_gen_*" AREA_GROUP=OUTPUT;
INST "ChoiceSig_*" AREA_GROUP=OUTPUT;
INST "output_selector*" AREA_GROUP=OUTPUT;
AREA_GROUP "OUTPUT" RANGE=SLICE_X2Y25:SLICE_X3Y33;


#common position inside an CLB
INST "*/rDelay_LUT" BEL = D6LUT;
INST "*/IDelay_LUT" BEL = A6LUT;
INST "*/AND_LUT" BEL = B6LUT;
INST "*/EN_LUT" BEL = C6LUT;


#default relative positions of MT-Elements (in one col)
#delaychain and AND
INST "IN[*]..MT/CLB_53/*_RLM" RLOC = X-1Y53;
INST "IN[*]..MT/CLB_53/*_LUT" RLOC = X0Y53;
INST "IN[*]..MT/CLB_52/*_LUT" RLOC = X0Y52;
INST "IN[*]..MT/CLB_51/*_LUT" RLOC = X0Y51;
INST "IN[*]..MT/CLB_50/*_LUT" RLOC = X0Y50;
INST "IN[*]..MT/CLB_49/*_LUT" RLOC = X0Y49;
INST "IN[*]..MT/CLB_48/*_LUT" RLOC = X0Y48;
INST "IN[*]..MT/CLB_47/*_LUT" RLOC = X0Y47;
INST "IN[*]..MT/CLB_46/*_LUT" RLOC = X0Y46;
INST "IN[*]..MT/CLB_45/*_LUT" RLOC = X0Y45;
INST "IN[*]..MT/CLB_44/*_LUT" RLOC = X0Y44;
INST "IN[*]..MT/CLB_43/*_LUT" RLOC = X0Y43;
INST "IN[*]..MT/CLB_42/*_LUT" RLOC = X0Y42;
INST "IN[*]..MT/CLB_41/*_LUT" RLOC = X0Y41;
INST "IN[*]..MT/CLB_40/*_LUT" RLOC = X0Y40;
INST "IN[*]..MT/CLB_39/*_LUT" RLOC = X0Y39;
INST "IN[*]..MT/CLB_38/*_LUT" RLOC = X0Y38;
INST "IN[*]..MT/CLB_37/*_LUT" RLOC = X0Y37;
INST "IN[*]..MT/CLB_36/*_LUT" RLOC = X0Y36;
INST "IN[*]..MT/CLB_35/*_LUT" RLOC = X0Y35;
INST "IN[*]..MT/CLB_34/*_LUT" RLOC = X0Y34;
INST "IN[*]..MT/CLB_33/*_LUT" RLOC = X0Y33;
INST "IN[*]..MT/CLB_32/*_LUT" RLOC = X0Y32;
INST "IN[*]..MT/CLB_31/*_LUT" RLOC = X0Y31;
INST "IN[*]..MT/CLB_30/*_LUT" RLOC = X0Y30;
INST "IN[*]..MT/CLB_29/*_LUT" RLOC = X0Y29;
INST "IN[*]..MT/CLB_28/*_LUT" RLOC = X0Y28;
INST "IN[*]..MT/CLB_27/*_LUT" RLOC = X0Y27;
INST "IN[*]..MT/CLB_26/*_LUT" RLOC = X0Y26;
INST "IN[*]..MT/CLB_25/*_LUT" RLOC = X0Y25;
INST "IN[*]..MT/CLB_24/*_LUT" RLOC = X0Y24;
INST "IN[*]..MT/CLB_23/*_LUT" RLOC = X0Y23;
```

```
INST "IN[*]..MT/CLB_22/*_LUT" RLOC = X0Y22;
INST "IN[*]..MT/CLB_21/*_LUT" RLOC = X0Y21;
INST "IN[*]..MT/CLB_20/*_LUT" RLOC = X0Y20;
INST "IN[*]..MT/CLB_19/*_LUT" RLOC = X0Y19;
INST "IN[*]..MT/CLB_18/*_LUT" RLOC = X0Y18;
INST "IN[*]..MT/CLB_17/*_LUT" RLOC = X0Y17;
INST "IN[*]..MT/CLB_16/*_LUT" RLOC = X0Y16;
INST "IN[*]..MT/CLB_15/*_LUT" RLOC = X0Y15;
INST "IN[*]..MT/CLB_14/*_LUT" RLOC = X0Y14;
INST "IN[*]..MT/CLB_13/*_LUT" RLOC = X0Y13;
INST "IN[*]..MT/CLB_12/*_LUT" RLOC = X0Y12;
INST "IN[*]..MT/CLB_11/*_LUT" RLOC = X0Y11;
INST "IN[*]..MT/CLB_10/*_LUT" RLOC = X0Y10;
INST "IN[*]..MT/CLB_09/*_LUT" RLOC = X0Y9;
INST "IN[*]..MT/CLB_08/*_LUT" RLOC = X0Y8;
INST "IN[*]..MT/CLB_07/*_LUT" RLOC = X0Y7;
INST "IN[*]..MT/CLB_06/*_LUT" RLOC = X0Y6;
INST "IN[*]..MT/CLB_05/*_LUT" RLOC = X0Y5;
INST "IN[*]..MT/CLB_04/*_LUT" RLOC = X0Y4;
INST "IN[*]..MT/CLB_03/*_LUT" RLOC = X0Y3;
INST "IN[*]..MT/CLB_02/*_LUT" RLOC = X0Y2;
INST "IN[*]..MT/CLB_01/*_LUT" RLOC = X0Y1;
INST "IN[*]..MT/CLB_00/*_LUT" RLOC = X0Y0;
INST "IN[*]..MT/CLB_00/*_RLM" RLOC = X-1Y0;

#SwitchDelays
INST "SD1_0/*" U_SET = SD_0; INST "SD1_0/SwD_0" RLOC_ORIGIN = X4Y57;
INST "SD1_1/*" U_SET = SD_1; INST "SD1_1/SwD_0" RLOC_ORIGIN = X8Y57;
INST "SD1_2/*" U_SET = SD_2; INST "SD1_2/SwD_0" RLOC_ORIGIN = X12Y57;
INST "SD1_3/*" U_SET = SD_3; INST "SD1_3/SwD_0" RLOC_ORIGIN = X14Y57;
INST "SD1_4/*" U_SET = SD_4; INST "SD1_4/SwD_0" RLOC_ORIGIN = X16Y57;
INST "SD1_5/*" U_SET = SD_5; INST "SD1_5/SwD_0" RLOC_ORIGIN = X18Y57;
INST "SD1_6/*" U_SET = SD_6; INST "SD1_6/SwD_0" RLOC_ORIGIN = X20Y57;
INST "SD1_7/*" U_SET = SD_7; INST "SD1_7/SwD_0" RLOC_ORIGIN = X24Y57;
INST "SD1_8/*" U_SET = SD_8; INST "SD1_8/SwD_0" RLOC_ORIGIN = X26Y57;
INST "SD1_9/*" U_SET = SD_9; INST "SD1_9/SwD_0" RLOC_ORIGIN = X28Y57;
INST "SD1_10/*" U_SET = SD_10; INST "SD1_10/SwD_0" RLOC_ORIGIN = X30Y57;
INST "SD1_11/*" U_SET = SD_11; INST "SD1_11/SwD_0" RLOC_ORIGIN = X32Y57;
INST "SD1_12/*" U_SET = SD_12; INST "SD1_12/SwD_0" RLOC_ORIGIN = X36Y57;
INST "SD1_13/*" U_SET = SD_13; INST "SD1_13/SwD_0" RLOC_ORIGIN = X38Y57;
INST "SD1_14/*" U_SET = SD_14; INST "SD1_14/SwD_0" RLOC_ORIGIN = X40Y57;
INST "SD1_15/*" U_SET = SD_15; INST "SD1_15/SwD_0" RLOC_ORIGIN = X42Y57;
INST "SD1_16/*" U_SET = SD_16; INST "SD1_16/SwD_0" RLOC_ORIGIN = X44Y57;
INST "SD1_17/*" U_SET = SD_17; INST "SD1_17/SwD_0" RLOC_ORIGIN = X48Y57;
INST "SD1_18/*" U_SET = SD_18; INST "SD1_18/SwD_0" RLOC_ORIGIN = X52Y57;
INST "SD1_19/*" U_SET = SD_19; INST "SD1_19/SwD_0" RLOC_ORIGIN = X54Y57;
INST "SD1_20/*" U_SET = SD_20; INST "SD1_20/SwD_0" RLOC_ORIGIN = X56Y57;
INST "SD1_21/*" U_SET = SD_21; INST "SD1_21/SwD_0" RLOC_ORIGIN = X58Y57;
INST "SD1_22/*" U_SET = SD_22; INST "SD1_22/SwD_0" RLOC_ORIGIN = X60Y57;
INST "SD1_23/*" U_SET = SD_23; INST "SD1_23/SwD_0" RLOC_ORIGIN = X64Y57;
INST "SD1_24/*" U_SET = SD_24; INST "SD1_24/SwD_0" RLOC_ORIGIN = X66Y57;
INST "SD1_25/*" U_SET = SD_25; INST "SD1_25/SwD_0" RLOC_ORIGIN = X68Y57;
INST "SD1_26/*" U_SET = SD_26; INST "SD1_26/SwD_0" RLOC_ORIGIN = X70Y57;
INST "SD1_27/*" U_SET = SD_27; INST "SD1_27/SwD_0" RLOC_ORIGIN = X72Y57;
INST "SD1_28/*" U_SET = SD_28; INST "SD1_28/SwD_0" RLOC_ORIGIN = X76Y57;
INST "SD1_29/*" U_SET = SD_29; INST "SD1_29/SwD_0" RLOC_ORIGIN = X80Y57;
INST "SD1_30/*" U_SET = SD_30; INST "SD1_30/SwD_0" RLOC_ORIGIN = X84Y57;
INST "SD1_31/*" U_SET = SD_31; INST "SD1_31/SwD_0" RLOC_ORIGIN = X88Y57;

INST "SD2_32/*" U_SET = SD_32; INST "SD2_32/SwD_0" RLOC_ORIGIN = X4Y101;
INST "SD2_33/*" U_SET = SD_33; INST "SD2_33/SwD_0" RLOC_ORIGIN = X8Y101;
INST "SD2_34/*" U_SET = SD_34; INST "SD2_34/SwD_0" RLOC_ORIGIN = X12Y101;
INST "SD2_35/*" U_SET = SD_35; INST "SD2_35/SwD_0" RLOC_ORIGIN = X14Y101;
INST "SD2_36/*" U_SET = SD_36; INST "SD2_36/SwD_0" RLOC_ORIGIN = X16Y101;
INST "SD2_37/*" U_SET = SD_37; INST "SD2_37/SwD_0" RLOC_ORIGIN = X18Y101;
INST "SD2_38/*" U_SET = SD_38; INST "SD2_38/SwD_0" RLOC_ORIGIN = X20Y101;
INST "SD2_39/*" U_SET = SD_39; INST "SD2_39/SwD_0" RLOC_ORIGIN = X24Y101;
INST "SD2_40/*" U_SET = SD_40; INST "SD2_40/SwD_0" RLOC_ORIGIN = X26Y101;
INST "SD2_41/*" U_SET = SD_41; INST "SD2_41/SwD_0" RLOC_ORIGIN = X28Y101;
INST "SD2_42/*" U_SET = SD_42; INST "SD2_42/SwD_0" RLOC_ORIGIN = X30Y101;
INST "SD2_43/*" U_SET = SD_43; INST "SD2_43/SwD_0" RLOC_ORIGIN = X32Y101;
INST "SD2_44/*" U_SET = SD_44; INST "SD2_44/SwD_0" RLOC_ORIGIN = X36Y101;
INST "SD2_45/*" U_SET = SD_45; INST "SD2_45/SwD_0" RLOC_ORIGIN = X38Y101;
INST "SD2_46/*" U_SET = SD_46; INST "SD2_46/SwD_0" RLOC_ORIGIN = X40Y101;
```

```
147 INST "SD2_47/*" U_SET = SD_47; INST "SD2_47/SwD_0" RLOC_ORIGIN = X42Y101;
148 INST "SD2_48/*" U_SET = SD_48; INST "SD2_48/SwD_0" RLOC_ORIGIN = X44Y101;
149 INST "SD2_49/*" U_SET = SD_49; INST "SD2_49/SwD_0" RLOC_ORIGIN = X48Y101;
150 INST "SD2_50/*" U_SET = SD_50; INST "SD2_50/SwD_0" RLOC_ORIGIN = X52Y101;
151 INST "SD2_51/*" U_SET = SD_51; INST "SD2_51/SwD_0" RLOC_ORIGIN = X54Y101;
152 INST "SD2_52/*" U_SET = SD_52; INST "SD2_52/SwD_0" RLOC_ORIGIN = X56Y101;
153 INST "SD2_53/*" U_SET = SD_53; INST "SD2_53/SwD_0" RLOC_ORIGIN = X58Y101;
154 INST "SD2_54/*" U_SET = SD_54; INST "SD2_54/SwD_0" RLOC_ORIGIN = X60Y101;
155 INST "SD2_55/*" U_SET = SD_55; INST "SD2_55/SwD_0" RLOC_ORIGIN = X64Y101;
156 INST "SD2_56/*" U_SET = SD_56; INST "SD2_56/SwD_0" RLOC_ORIGIN = X66Y101;
157 INST "SD2_57/*" U_SET = SD_57; INST "SD2_57/SwD_0" RLOC_ORIGIN = X68Y101;
158 INST "SD2_58/*" U_SET = SD_58; INST "SD2_58/SwD_0" RLOC_ORIGIN = X70Y101;
159 INST "SD2_59/*" U_SET = SD_59; INST "SD2_59/SwD_0" RLOC_ORIGIN = X72Y101;
160 INST "SD2_60/*" U_SET = SD_60; INST "SD2_60/SwD_0" RLOC_ORIGIN = X76Y101;
161 INST "SD2_61/*" U_SET = SD_61; INST "SD2_61/SwD_0" RLOC_ORIGIN = X80Y101;
162 INST "SD2_62/*" U_SET = SD_62; INST "SD2_62/SwD_0" RLOC_ORIGIN = X84Y101;
163 INST "SD2_63/*" U_SET = SD_63; INST "SD2_63/SwD_0" RLOC_ORIGIN = X88Y101;
164
165 INST "SD2_*/SwD_0/S_1" RLOC = X0Y0;
166 INST "SD2_*/SwD_1/S_1" RLOC = X0Y-1;
167 INST "SD2_*/SwD_2/S_1" RLOC = X0Y-2;
168 INST "SD2_*/SwD_3/S_1" RLOC = X0Y-3;
169 INST "SD2_*/SwD_4/S_1" RLOC = X0Y-4;
170 INST "SD2_*/SwD_5/S_1" RLOC = X0Y-5;
171 INST "SD2_*/SwD_6/S_1" RLOC = X0Y-6;
172 INST "SD2_*/SwD_7/S_1" RLOC = X0Y-7;
173 INST "SD2_*/SwD_8/S_1" RLOC = X0Y-8;
174 INST "SD2_*/SwD_9/S_1" RLOC = X0Y-9;
175 INST "SD2_*/SwD_10/S_1" RLOC = X0Y-10;
176 INST "SD2_*/SwD_11/S_1" RLOC = X0Y-11;
177 INST "SD2_*/SwD_12/S_1" RLOC = X0Y-12;
178 INST "SD2_*/SwD_13/S_1" RLOC = X0Y-13;
179 INST "SD2_*/SwD_14/S_1" RLOC = X0Y-14;
180 INST "SD2_*/SwD_15/S_1" RLOC = X0Y-15;
181 INST "SD2_*/SwD_16/S_1" RLOC = X0Y-16;
182 INST "SD2_*/SwD_17/S_1" RLOC = X0Y-17;
183 INST "SD2_*/SwD_18/S_1" RLOC = X0Y-18;
184 INST "SD2_*/SwD_19/S_1" RLOC = X0Y-19;
185 INST "SD2_*/SwD_20/S_1" RLOC = X0Y-20;
186 INST "SD2_*/SwD_21/S_1" RLOC = X0Y-21;
187
188 INST "SD1_*/SwD_0/S_1" RLOC = X0Y0;
189 INST "SD1_*/SwD_1/S_1" RLOC = X0Y1;
190 INST "SD1_*/SwD_2/S_1" RLOC = X0Y2;
191 INST "SD1_*/SwD_3/S_1" RLOC = X0Y3;
192 INST "SD1_*/SwD_4/S_1" RLOC = X0Y4;
193 INST "SD1_*/SwD_5/S_1" RLOC = X0Y5;
194 INST "SD1_*/SwD_6/S_1" RLOC = X0Y6;
195 INST "SD1_*/SwD_7/S_1" RLOC = X0Y7;
196 INST "SD1_*/SwD_8/S_1" RLOC = X0Y8;
197 INST "SD1_*/SwD_9/S_1" RLOC = X0Y9;
198 INST "SD1_*/SwD_10/S_1" RLOC = X0Y10;
199 INST "SD1_*/SwD_11/S_1" RLOC = X0Y11;
200 INST "SD1_*/SwD_12/S_1" RLOC = X0Y12;
201 INST "SD1_*/SwD_13/S_1" RLOC = X0Y13;
202 INST "SD1_*/SwD_14/S_1" RLOC = X0Y14;
203 INST "SD1_*/SwD_15/S_1" RLOC = X0Y15;
204 INST "SD1_*/SwD_16/S_1" RLOC = X0Y16;
205 INST "SD1_*/SwD_17/S_1" RLOC = X0Y17;
206 INST "SD1_*/SwD_18/S_1" RLOC = X0Y18;
207 INST "SD1_*/SwD_19/S_1" RLOC = X0Y19;
208 INST "SD1_*/SwD_20/S_1" RLOC = X0Y20;
209 INST "SD1_*/SwD_21/S_1" RLOC = X0Y21;
```

Listing B.3: handroutet.ucf

```
#a few LUTs where put into the signalpath to be able to redirect the path, thus making it longer
#this was done, to minimize the differenze between the left and right input signals, so the 5ns
#of the IODELAYs are not wasted

INST "H2_full_delay/D" LOC = IODELAY_X0Y60;
INST "H1_full_delay/D" LOC = IODELAY_X0Y62;

INST "szinti_delays/R_8" LOC = IODELAY_X0Y79;
INST "szinti_delays/R_7" LOC = IODELAY_X0Y78;
INST "szinti_delays/R_6" LOC = IODELAY_X0Y77;
INST "szinti_delays/R_5" LOC = IODELAY_X0Y76;
INST "szinti_delays/R_4" LOC = IODELAY_X0Y75;
INST "szinti_delays/R_3" LOC = IODELAY_X0Y74;
INST "szinti_delays/R_2" LOC = IODELAY_X0Y73;
INST "szinti_delays/R_1" LOC = IODELAY_X0Y72;

INST "szinti_delays/L_8" LOC = IODELAY_X0Y51;
INST "szinti_delays/L_7" LOC = IODELAY_X0Y50;
INST "szinti_delays/L_6" LOC = IODELAY_X0Y49;
INST "szinti_delays/L_5" LOC = IODELAY_X0Y48;
INST "szinti_delays/L_4" LOC = IODELAY_X0Y47;
INST "szinti_delays/L_3" LOC = IODELAY_X0Y46;
INST "szinti_delays/L_2" LOC = IODELAY_X0Y45;
INST "szinti_delays/L_1" LOC = IODELAY_X0Y44;

INST "OA1_delay/D" LOC = IODELAY_X0Y64;
INST "OA2_delay/D" LOC = IODELAY_X0Y68;
INST "OB1_delay/D" LOC = IODELAY_X0Y53;
INST "OB2_delay/D" LOC = IODELAY_X0Y42;

INST "OA1_buf/S_1" LOC = SLICE_X0Y32 | BEL = A6LUT;
INST "OA2_buf/S_1" LOC = SLICE_X0Y34 | BEL = A6LUT;
INST "OB1_buf/S_1" LOC = SLICE_X0Y26 | BEL = A6LUT;
INST "OB2_buf/S_1" LOC = SLICE_X0Y21 | BEL = A6LUT;

INST "alignsig_delays/B_18/D" LOC = IODELAY_X0Y33;
INST "alignsig_delays/B_17/D" LOC = IODELAY_X0Y32;
INST "alignsig_delays/B_16/D" LOC = IODELAY_X0Y31;
INST "alignsig_delays/B_15/D" LOC = IODELAY_X0Y30;
INST "alignsig_delays/B_14/D" LOC = IODELAY_X0Y29;
INST "alignsig_delays/B_13/D" LOC = IODELAY_X0Y28;
INST "alignsig_delays/B_12/D" LOC = IODELAY_X0Y27;
INST "alignsig_delays/B_11/D" LOC = IODELAY_X0Y26;
INST "alignsig_delays/B_10/D" LOC = IODELAY_X0Y25;
INST "alignsig_delays/B_9/D" LOC = IODELAY_X0Y24;
INST "alignsig_delays/B_8/D" LOC = IODELAY_X0Y23;
INST "alignsig_delays/B_7/D" LOC = IODELAY_X0Y22;
INST "alignsig_delays/B_6/D" LOC = IODELAY_X0Y21;
INST "alignsig_delays/B_5/D" LOC = IODELAY_X0Y20;
INST "alignsig_delays/B_4/D" LOC = IODELAY_X0Y19;
INST "alignsig_delays/B_3/D" LOC = IODELAY_X0Y18;
INST "alignsig_delays/B_2/D" LOC = IODELAY_X0Y17;
INST "alignsig_delays/B_1/D" LOC = IODELAY_X0Y16;
INST "alignsig_delays/B_0/D" LOC = IODELAY_X0Y15;
INST "alignsig_delays/F_0/D" LOC = IODELAY_X0Y14;

#inputs
INST "*/choice" BEL = D6LUT;

INST "IN[38]..L_Delay/D_1" LOC = IODELAY_X2Y160;
INST "IN[47]..L_Delay/D_1" LOC = IODELAY_X2Y76;
INST "IN[53]..L_Delay/D_1" LOC = IODELAY_X2Y184;
INST "IN[61]..L_Delay/D_1" LOC = IODELAY_X1Y292;
INST "IN[21]..R_Delay/D_1" LOC = IODELAY_X0Y80;
INST "IN[20]..R_Delay/D_1" LOC = IODELAY_X0Y82;
INST "IN[17]..R_Delay/D_1" LOC = IODELAY_X0Y38;
INST "IN[5]..L_Delay/D_1" LOC = IODELAY_X0Y104;
INST "IN[12]..L_Delay/D_1" LOC = IODELAY_X0Y138;
INST "IN[30]..R_Delay/choice" LOC = SLICE_X0Y116;
INST "IN[27]..R_Delay/choice" LOC = SLICE_X0Y117;
INST "IN[21]..R_Delay/choice" LOC = SLICE_X0Y40;
INST "IN[16]..R_Delay/choice" LOC = SLICE_X0Y17;
```

```
73  INST "IN[7]..L_Delay/choice" LOC = SLICE_X0Y51;
74  INST "IN[11]..L_Delay/choice" LOC = SLICE_X0Y67;
75  INST "IN[49]..R_Delay/choice" LOC = SLICE_X0Y0;
76  INST "IN[16]..R_Delay/D_1" LOC = IODELAY_X0Y34;
77  INST "IN[8]..L_Delay/choice" LOC = SLICE_X0Y70;
78  INST "IN[9]..L_Delay/D_1" LOC = IODELAY_X0Y142;
79  INST "IN[48]..R_Delay/choice" LOC = SLICE_X0Y1;
80  INST "IN[49]..R_Delay/D_1" LOC = IODELAY_X0Y0;
81  INST "IN[18]..R_Delay/choice" LOC = SLICE_X0Y43;
82  INST "IN[7]..L_Delay/D_1" LOC = IODELAY_X0Y102;
83  INST "IN[15]..L_Delay/choice" LOC = SLICE_X0Y62;
84  INST "IN[14]..L_Delay/D_1" LOC = IODELAY_X0Y126;
85  INST "IN[0]..R_Delay/choice" LOC = SLICE_X0Y87;
86  INST "IN[2]..R_Delay/D_1" LOC = IODELAY_X0Y178;
87  INST "IN[1]..R_Delay/choice" LOC = SLICE_X0Y86;
88  INST "IN[24]..R_Delay/D_1" LOC = IODELAY_X0Y170;
89  INST "IN[0]..L_Delay/choice" LOC = SLICE_X0Y59;
90  INST "IN[15]..L_Delay/D_1" LOC = IODELAY_X0Y124;
91  INST "IN[2]..R_Delay/choice" LOC = SLICE_X0Y89;
92  INST "IN[3]..R_Delay/D_1" LOC = IODELAY_X0Y176;
93  INST "IN[23]..R_Delay/choice" LOC = SLICE_X0Y105;
94  INST "IN[22]..R_Delay/D_1" LOC = IODELAY_X0Y212;
95  INST "IN[3]..L_Delay/choice" LOC = SLICE_X0Y54;
96  INST "IN[6]..L_Delay/D_1" LOC = IODELAY_X0Y110;
97  INST "IN[4]..R_Delay/choice" LOC = SLICE_X0Y90;
98  INST "IN[5]..R_Delay/D_1" LOC = IODELAY_X0Y182;
99  INST "IN[12]..L_Delay/choice" LOC = SLICE_X0Y69;
100 INST "IN[8]..L_Delay/D_1" LOC = IODELAY_X0Y140;
101 INST "IN[3]..R_Delay/choice" LOC = SLICE_X0Y88;
102 INST "IN[4]..R_Delay/D_1" LOC = IODELAY_X0Y180;
103 INST "IN[19]..R_Delay/choice" LOC = SLICE_X0Y64;
104 INST "IN[10]..L_Delay/D_1" LOC = IODELAY_X0Y130;
105 INST "IN[24]..R_Delay/choice" LOC = SLICE_X0Y85;
106 INST "IN[0]..R_Delay/D_1" LOC = IODELAY_X0Y174;
107 INST "IN[26]..R_Delay/choice" LOC = SLICE_X0Y113;
108 INST "IN[28]..R_Delay/D_1" LOC = IODELAY_X0Y228;
109 INST "IN[10]..L_Delay/choice" LOC = SLICE_X0Y65;
110 INST "IN[13]..L_Delay/D_1" LOC = IODELAY_X0Y132;
111 INST "IN[20]..L_Delay/choice" LOC = SLICE_X0Y138;
112 INST "IN[18]..L_Delay/D_1" LOC = IODELAY_X0Y278;
113 INST "IN[31]..L_Delay/choice" LOC = SLICE_X0Y135;
114 INST "IN[12]..R_Delay/D_1" LOC = IODELAY_X0Y272;
115 INST "IN[23]..L_Delay/choice" LOC = SLICE_X0Y130;
116 INST "IN[27]..L_Delay/D_1" LOC = IODELAY_X0Y264;
117 INST "IN[7]..R_Delay/choice" LOC = SLICE_X0Y92;
118 INST "IN[6]..R_Delay/D_1" LOC = IODELAY_X0Y190;
119 INST "IN[6]..R_Delay/choice" LOC = SLICE_X0Y95;
120 INST "IN[23]..R_Delay/D_1" LOC = IODELAY_X0Y210;
121 INST "IN[22]..L_Delay/choice" LOC = SLICE_X0Y129;
122 INST "IN[23]..L_Delay/D_1" LOC = IODELAY_X0Y260;
123 INST "IN[63]..L_Delay/choice" LOC = SLICE_X0Y159;
124 INST "IN[10]..R_Delay/choice" LOC = SLICE_X0Y111;
125 INST "IN[8]..R_Delay/D_1" LOC = IODELAY_X0Y224;
126 INST "IN[8]..R_Delay/choice" LOC = SLICE_X0Y112;
127 INST "IN[26]..R_Delay/D_1" LOC = IODELAY_X0Y226;
128 INST "IN[27]..L_Delay/choice" LOC = SLICE_X0Y132;
129 INST "IN[30]..L_Delay/D_1" LOC = IODELAY_X0Y266;
130 INST "IN[39]..R_Delay/choice" LOC = SLICE_X0Y152;
131 INST "IN[63]..L_Delay/D_1" LOC = IODELAY_X0Y318;
132 INST "IN[17]..L_Delay/choice" LOC = SLICE_X0Y121;
133 INST "IN[16]..L_Delay/D_1" LOC = IODELAY_X0Y244;
134 INST "IN[19]..L_Delay/choice" LOC = SLICE_X0Y137;
135 INST "IN[20]..L_Delay/D_1" LOC = IODELAY_X0Y276;
136 INST "IN[13]..R_Delay/choice" LOC = SLICE_X0Y102;
137 INST "IN[10]..R_Delay/D_1" LOC = IODELAY_X0Y222;
138 INST "IN[31]..R_Delay/choice" LOC = SLICE_X0Y119;
139 INST "IN[14]..R_Delay/D_1" LOC = IODELAY_X0Y240;
140 INST "IN[32]..R_Delay/choice" LOC = SLICE_X0Y147;
141 INST "IN[37]..R_Delay/D_1" LOC = IODELAY_X0Y296;
142 INST "IN[25]..L_Delay/choice" LOC = SLICE_X0Y141;
143 INST "IN[24]..L_Delay/D_1" LOC = IODELAY_X0Y284;
144 INST "IN[37]..R_Delay/choice" LOC = SLICE_X0Y148;
145 INST "IN[34]..R_Delay/D_1" LOC = IODELAY_X0Y298;
146 INST "IN[36]..R_Delay/choice" LOC = SLICE_X0Y150;
```

```
147 INST "IN[38]..R_Delay/D_1" LOC = IODELAY_X0Y302;
148 INST "IN[35]..R_Delay/choice" LOC = SLICE_X0Y146;
149 INST "IN[32]..R_Delay/D_1" LOC = IODELAY_X0Y294;
150 INST "IN[2]..L_Delay/choice" LOC = SLICE_X0Y58;
151 INST "IN[0]..L_Delay/D_1" LOC = IODELAY_X0Y118;
152 INST "IN[17]..R_Delay/choice" LOC = SLICE_X0Y19;
153 INST "IN[4]..L_Delay/choice" LOC = SLICE_X0Y56;
154 INST "IN[1]..L_Delay/D_1" LOC = IODELAY_X0Y114;
155 INST "IN[1]..L_Delay/choice" LOC = SLICE_X0Y57;
156 INST "IN[2]..L_Delay/D_1" LOC = IODELAY_X0Y116;
157 INST "IN[20]..R_Delay/choice" LOC = SLICE_X0Y41;
158 INST "IN[18]..R_Delay/D_1" LOC = IODELAY_X0Y86;
159 INST "IN[5]..R_Delay/choice" LOC = SLICE_X0Y91;
160 INST "IN[7]..R_Delay/D_1" LOC = IODELAY_X0Y184;
161 INST "IN[6]..L_Delay/choice" LOC = SLICE_X0Y55;
162 INST "IN[4]..L_Delay/D_1" LOC = IODELAY_X0Y112;
163 INST "IN[5]..L_Delay/choice" LOC = SLICE_X0Y52;
164 INST "IN[3]..L_Delay/D_1" LOC = IODELAY_X0Y108;
165 INST "IN[14]..L_Delay/choice" LOC = SLICE_X0Y63;
166 INST "IN[19]..R_Delay/D_1" LOC = IODELAY_X0Y128;
167 INST "IN[22]..R_Delay/choice" LOC = SLICE_X0Y106;
168 INST "IN[15]..R_Delay/D_1" LOC = IODELAY_X0Y200;
169 INST "IN[13]..L_Delay/choice" LOC = SLICE_X0Y66;
170 INST "IN[11]..L_Delay/D_1" LOC = IODELAY_X0Y134;
171 INST "IN[13]..R_Delay/D_1" LOC = IODELAY_X0Y204;
172 INST "IN[27]..R_Delay/D_1" LOC = IODELAY_X0Y234;
173 INST "IN[29]..R_Delay/D_1" LOC = IODELAY_X0Y236;
174 INST "IN[25]..R_Delay/choice" LOC = SLICE_X0Y84;
175 INST "IN[1]..R_Delay/D_1" LOC = IODELAY_X0Y172;
176 INST "IN[28]..R_Delay/choice" LOC = SLICE_X0Y114;
177 INST "IN[9]..R_Delay/D_1" LOC = IODELAY_X0Y230;
178 INST "IN[9]..L_Delay/choice" LOC = SLICE_X0Y71;
179 INST "IN[25]..R_Delay/D_1" LOC = IODELAY_X0Y168;
180 INST "IN[29]..R_Delay/choice" LOC = SLICE_X0Y118;
181 INST "IN[31]..R_Delay/D_1" LOC = IODELAY_X0Y238;
182 INST "IN[24]..L_Delay/choice" LOC = SLICE_X0Y142;
183 INST "IN[33]..R_Delay/D_1" LOC = IODELAY_X0Y290;
184 INST "IN[11]..R_Delay/choice" LOC = SLICE_X0Y101;
185 INST "IN[9]..R_Delay/choice" LOC = SLICE_X0Y115;
186 INST "IN[30]..R_Delay/D_1" LOC = IODELAY_X0Y232;
187 INST "IN[30]..L_Delay/choice" LOC = SLICE_X0Y133;
188 INST "IN[28]..L_Delay/D_1" LOC = IODELAY_X0Y268;
189 INST "IN[28]..L_Delay/choice" LOC = SLICE_X0Y134;
190 INST "IN[31]..L_Delay/D_1" LOC = IODELAY_X0Y270;
191 INST "IN[15]..R_Delay/choice" LOC = SLICE_X0Y100;
192 INST "IN[11]..R_Delay/D_1" LOC = IODELAY_X0Y202;
193 INST "IN[12]..R_Delay/choice" LOC = SLICE_X0Y136;
194 INST "IN[19]..L_Delay/D_1" LOC = IODELAY_X0Y274;
195 INST "IN[33]..R_Delay/choice" LOC = SLICE_X0Y145;
196 INST "IN[35]..R_Delay/D_1" LOC = IODELAY_X0Y292;
197 INST "IN[38]..R_Delay/choice" LOC = SLICE_X0Y151;
198 INST "IN[39]..R_Delay/D_1" LOC = IODELAY_X0Y304;
199 INST "IN[34]..R_Delay/choice" LOC = SLICE_X0Y149;
200 INST "IN[36]..R_Delay/D_1" LOC = IODELAY_X0Y300;
201 INST "IN[14]..R_Delay/choice" LOC = SLICE_X0Y120;
202 INST "IN[17]..L_Delay/D_1" LOC = IODELAY_X0Y242;
203 INST "IN[21]..L_Delay/choice" LOC = SLICE_X0Y128;
204 INST "IN[22]..L_Delay/D_1" LOC = IODELAY_X0Y258;
205 INST "IN[16]..L_Delay/choice" LOC = SLICE_X0Y122;
206 INST "IN[21]..L_Delay/D_1" LOC = IODELAY_X0Y256;
207 INST "IN[18]..L_Delay/choice" LOC = SLICE_X0Y139;
208 INST "IN[25]..L_Delay/D_1" LOC = IODELAY_X0Y282;
209 INST "IN[59]..L_Delay/choice" LOC = SLICE_X47Y147;
210 INST "IN[26]..L_Delay/D_1" LOC = IODELAY_X1Y296;
211 INST "IN[58]..L_Delay/choice" LOC = SLICE_X47Y145;
212 INST "IN[59]..L_Delay/D_1" LOC = IODELAY_X1Y294;
213 INST "IN[60]..L_Delay/choice" LOC = SLICE_X47Y155;
214 INST "IN[41]..L_Delay/D_1" LOC = IODELAY_X2Y100;
215 INST "IN[57]..L_Delay/choice" LOC = SLICE_X47Y149;
216 INST "IN[29]..L_Delay/D_1" LOC = IODELAY_X1Y302;
217 INST "IN[56]..L_Delay/choice" LOC = SLICE_X47Y143;
218 INST "IN[58]..L_Delay/D_1" LOC = IODELAY_X1Y290;
219 INST "IN[29]..L_Delay/choice" LOC = SLICE_X47Y151;
220 INST "IN[62]..L_Delay/D_1" LOC = IODELAY_X1Y306;
```

```
221 INST "IN[57]..R_Delay/choice" LOC = SLICE_X47Y141;
222 INST "IN[56]..L_Delay/D_1" LOC = IODELAY_X1Y286;
223 INST "IN[62]..L_Delay/choice" LOC = SLICE_X47Y153;
224 INST "IN[60]..L_Delay/D_1" LOC = IODELAY_X1Y310;
225 INST "IN[26]..L_Delay/choice" LOC = SLICE_X47Y148;
226 INST "IN[57]..L_Delay/D_1" LOC = IODELAY_X1Y298;
227 INST "IN[61]..L_Delay/choice" LOC = SLICE_X47Y146;
228 INST "IN[48]..R_Delay/D_1" LOC = IODELAY_X0Y2;
229 INST "IN[51]..R_Delay/choice" LOC = SLICE_X83Y28;
230 INST "IN[42]..L_Delay/D_1" LOC = IODELAY_X2Y54;
231 INST "IN[45]..L_Delay/choice" LOC = SLICE_X83Y32;
232 INST "IN[43]..L_Delay/D_1" LOC = IODELAY_X2Y60;
233 INST "IN[46]..L_Delay/choice" LOC = SLICE_X83Y36;
234 INST "IN[44]..L_Delay/D_1" LOC = IODELAY_X2Y68;
235 INST "IN[43]..R_Delay/choice" LOC = SLICE_X83Y58;
236 INST "IN[42]..R_Delay/D_1" LOC = IODELAY_X2Y114;
237 INST "IN[55]..R_Delay/choice" LOC = SLICE_X83Y22;
238 INST "IN[54]..R_Delay/D_1" LOC = IODELAY_X2Y42;
239 INST "IN[41]..L_Delay/choice" LOC = SLICE_X83Y50;
240 INST "IN[40]..L_Delay/D_1" LOC = IODELAY_X2Y98;
241 INST "IN[44]..L_Delay/choice" LOC = SLICE_X83Y34;
242 INST "IN[45]..L_Delay/D_1" LOC = IODELAY_X2Y64;
243 INST "IN[50]..R_Delay/choice" LOC = SLICE_X83Y37;
244 INST "IN[46]..L_Delay/D_1" LOC = IODELAY_X2Y72;
245 INST "IN[44]..R_Delay/choice" LOC = SLICE_X83Y59;
246 INST "IN[43]..R_Delay/D_1" LOC = IODELAY_X2Y116;
247 INST "IN[45]..R_Delay/choice" LOC = SLICE_X83Y54;
248 INST "IN[54]..R_Delay/choice" LOC = SLICE_X83Y21;
249 INST "IN[53]..R_Delay/D_1" LOC = IODELAY_X2Y40;
250 INST "IN[42]..R_Delay/choice" LOC = SLICE_X83Y57;
251 INST "IN[47]..R_Delay/D_1" LOC = IODELAY_X2Y112;
252 INST "IN[38]..L_Delay/choice" LOC = SLICE_X83Y80;
253 INST "IN[49]..L_Delay/D_1" LOC = IODELAY_X2Y162;
254 INST "IN[47]..R_Delay/choice" LOC = SLICE_X83Y56;
255 INST "IN[45]..R_Delay/D_1" LOC = IODELAY_X2Y108;
256 INST "IN[41]..R_Delay/choice" LOC = SLICE_X83Y53;
257 INST "IN[46]..R_Delay/D_1" LOC = IODELAY_X2Y104;
258 INST "IN[52]..R_Delay/choice" LOC = SLICE_X83Y25;
259 INST "IN[55]..R_Delay/D_1" LOC = IODELAY_X2Y44;
260 INST "IN[47]..L_Delay/choice" LOC = SLICE_X83Y38;
261 INST "IN[50]..R_Delay/D_1" LOC = IODELAY_X2Y74;
262 INST "IN[46]..R_Delay/choice" LOC = SLICE_X83Y52;
263 INST "IN[40]..R_Delay/D_1" LOC = IODELAY_X2Y102;
264 INST "IN[36]..L_Delay/choice" LOC = SLICE_X83Y82;
265 INST "IN[48]..L_Delay/D_1" LOC = IODELAY_X2Y166;
266 INST "IN[49]..L_Delay/choice" LOC = SLICE_X83Y81;
267 INST "IN[36]..L_Delay/D_1" LOC = IODELAY_X2Y164;
268 INST "IN[52]..L_Delay/choice" LOC = SLICE_X83Y90;
269 INST "IN[63]..R_Delay/D_1" LOC = IODELAY_X2Y182;
270 INST "IN[32]..L_Delay/choice" LOC = SLICE_X83Y121;
271 INST "IN[34]..L_Delay/D_1" LOC = IODELAY_X2Y246;
272 INST "IN[40]..L_Delay/choice" LOC = SLICE_X83Y49;
273 INST "IN[57]..R_Delay/D_1" LOC = IODELAY_X1Y282;
274 INST "IN[56]..R_Delay/choice" LOC = SLICE_X83Y131;
275 INST "IN[60]..R_Delay/D_1" LOC = IODELAY_X2Y264;
276 INST "IN[63]..R_Delay/choice" LOC = SLICE_X83Y91;
277 INST "IN[35]..L_Delay/choice" LOC = SLICE_X83Y86;
278 INST "IN[50]..L_Delay/D_1" LOC = IODELAY_X2Y174;
279 INST "IN[33]..L_Delay/choice" LOC = SLICE_X83Y125;
280 INST "IN[61]..R_Delay/D_1" LOC = IODELAY_X2Y258;
281 INST "IN[51]..L_Delay/choice" LOC = SLICE_X83Y89;
282 INST "IN[52]..L_Delay/D_1" LOC = IODELAY_X2Y180;
283 INST "IN[48]..L_Delay/choice" LOC = SLICE_X83Y83;
284 INST "IN[54]..L_Delay/D_1" LOC = IODELAY_X2Y170;
285 INST "IN[60]..R_Delay/choice" LOC = SLICE_X83Y132;
286 INST "IN[59]..R_Delay/D_1" LOC = IODELAY_X2Y268;
287 INST "IN[61]..R_Delay/choice" LOC = SLICE_X83Y129;
288 INST "IN[56]..R_Delay/D_1" LOC = IODELAY_X2Y262;
289 INST "IN[39]..L_Delay/choice" LOC = SLICE_X83Y136;
290 INST "IN[44]..R_Delay/D_1" LOC = IODELAY_X2Y118;
291 INST "IN[55]..L_Delay/choice" LOC = SLICE_X83Y98;
292 INST "IN[32]..L_Delay/D_1" LOC = IODELAY_X2Y242;
293 INST "IN[37]..L_Delay/choice" LOC = SLICE_X83Y84;
294 INST "IN[53]..L_Delay/choice" LOC = SLICE_X83Y92;
```

```
295 INST "IN[55]..L_Delay/D_1" LOC = IODELAY_X2Y196;
296 INST "IN[34]..L_Delay/choice" LOC = SLICE_X83Y123;
297 INST "IN[33]..L_Delay/D_1" LOC = IODELAY_X2Y250;
298 INST "IN[59]..R_Delay/choice" LOC = SLICE_X83Y134;
299 INST "IN[58]..R_Delay/D_1" LOC = IODELAY_X2Y270;
300 INST "IN[58]..R_Delay/choice" LOC = SLICE_X83Y135;
301 INST "IN[39]..L_Delay/D_1" LOC = IODELAY_X2Y272;
302 INST "IN[42]..L_Delay/choice" LOC = SLICE_X83Y27;
303 INST "IN[52]..R_Delay/D_1" LOC = IODELAY_X2Y50;
304 INST "IN[40]..R_Delay/choice" LOC = SLICE_X83Y51;
305 INST "IN[41]..R_Delay/D_1" LOC = IODELAY_X2Y106;
306 INST "IN[43]..L_Delay/choice" LOC = SLICE_X83Y30;
307 INST "IN[51]..R_Delay/D_1" LOC = IODELAY_X2Y56;
308 INST "IN[53]..R_Delay/choice" LOC = SLICE_X83Y20;
309 INST "IN[37]..L_Delay/D_1" LOC = IODELAY_X2Y168;
310 INST "IN[62]..R_Delay/choice" LOC = SLICE_X83Y88;
311 INST "IN[51]..L_Delay/D_1" LOC = IODELAY_X2Y178;
312 INST "IN[50]..L_Delay/choice" LOC = SLICE_X83Y87;
313 INST "IN[62]..R_Delay/D_1" LOC = IODELAY_X2Y176;
314 INST "IN[54]..L_Delay/choice" LOC = SLICE_X83Y85;
315 INST "IN[35]..L_Delay/D_1" LOC = IODELAY_X2Y172;
316
317
318
319 #allign Matrix
320 INST "IN[0]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X5Y55;
321 INST "IN[1]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X9Y55;
322 INST "IN[2]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X13Y55;
323 INST "IN[3]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X15Y55;
324 INST "IN[4]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X17Y55;
325 INST "IN[5]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X19Y55;
326 INST "IN[6]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X21Y55;
327 INST "IN[7]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X25Y55;
328 INST "IN[8]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X27Y55;
329 INST "IN[9]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X29Y55;
330 INST "IN[10]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X31Y55;
331 INST "IN[11]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X33Y55;
332 INST "IN[12]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X37Y55;
333 INST "IN[13]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X39Y55;
334 INST "IN[14]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X41Y55;
335 INST "IN[15]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X43Y55;
336 INST "IN[16]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X45Y55;
337 INST "IN[17]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X49Y55;
338 INST "IN[18]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X53Y55;
339 INST "IN[19]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X55Y55;
340 INST "IN[20]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X57Y55;
341 INST "IN[21]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X59Y55;
342 INST "IN[22]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X61Y55;
343 INST "IN[23]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X65Y55;
344 INST "IN[24]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X67Y55;
345 INST "IN[25]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X69Y55;
346 INST "IN[26]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X71Y55;
347 INST "IN[27]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X73Y55;
348 INST "IN[28]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X77Y55;
349 INST "IN[29]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X81Y55;
350 INST "IN[30]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X85Y55;
351 INST "IN[31]..MT/OutEnableHi/EN_LUT" LOC = SLICE_X89Y55;
352
353 INST "IN[32]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X5Y104;
354 INST "IN[33]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X9Y104;
355 INST "IN[34]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X13Y104;
356 INST "IN[35]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X15Y104;
357 INST "IN[36]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X17Y104;
358 INST "IN[37]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X19Y104;
359 INST "IN[38]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X21Y104;
360 INST "IN[39]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X25Y104;
361 INST "IN[40]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X27Y104;
362 INST "IN[41]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X29Y104;
363 INST "IN[42]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X31Y104;
364 INST "IN[43]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X33Y104;
365 INST "IN[44]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X37Y104;
366 INST "IN[45]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X39Y104;
367 INST "IN[46]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X41Y104;
368 INST "IN[47]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X43Y104;
```

```
INST "IN[48]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X45Y104;
INST "IN[49]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X49Y104;
INST "IN[50]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X53Y104;
INST "IN[51]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X55Y104;
INST "IN[52]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X57Y104;
INST "IN[53]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X59Y104;
INST "IN[54]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X61Y104;
INST "IN[55]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X65Y104;
INST "IN[56]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X67Y104;
INST "IN[57]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X69Y104;
INST "IN[58]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X71Y104;
INST "IN[59]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X73Y104;
INST "IN[60]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X77Y104;
INST "IN[61]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X81Y104;
INST "IN[62]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X85Y104;
INST "IN[63]..MT/OutEnableLo/EN_LUT" LOC = SLICE_X89Y104;


#align L<->R, some ports need special delays
INST "IN[31]..L_eDelay/S_1" LOC = SLICE_X78Y13 | BEL = D6LUT;
INST "IN[31]..R_eDelay/S_1" LOC = SLICE_X0Y31 | BEL = D6LUT;
INST "IN[16]..L_eDelay/S_1" LOC = SLICE_X44Y3 | BEL = C6LUT;
INST "IN[16]..R_eDelay/S_1" LOC = SLICE_X22Y85 | BEL = A6LUT;
INST "IN[41]..L_eDelay/S_1" LOC = SLICE_X18Y115 | BEL = C6LUT;
INST "IN[41]..R_eDelay/S_1" LOC = SLICE_X55Y102 | BEL = B6LUT;
INST "IN[38]..L_eDelay/S_1" LOC = SLICE_X20Y103 | BEL = C6LUT;
INST "IN[38]..R_eDelay/S_1" LOC = SLICE_X11Y122 | BEL = D6LUT;
INST "IN[62]..L_eDelay/S_1" LOC = SLICE_X84Y103 | BEL = C6LUT;
INST "IN[62]..R_eDelay/S_1" LOC = SLICE_X84Y120 | BEL = B6LUT;
INST "IN[63]..L_eDelay/S_1" LOC = SLICE_X88Y103 | BEL = c6LUT;
INST "IN[63]..R_eDelay/S_1" LOC = SLICE_X68Y123 | BEL = A6LUT;
INST "IN[32]..R_eDelay/S_1" LOC = SLICE_X46Y151 | BEL = A6LUT;
INST "IN[33]..R_eDelay/S_1" LOC = SLICE_X46Y151 | BEL = B6LUT;
INST "IN[34]..R_eDelay/S_1" LOC = SLICE_X44Y151 | BEL = C6LUT;
INST "IN[35]..R_eDelay/S_1" LOC = SLICE_X44Y151 | BEL = D6LUT;
INST "IN[36]..R_eDelay/S_1" LOC = SLICE_X44Y151 | BEL = A6LUT;
INST "IN[37]..R_eDelay/S_1" LOC = SLICE_X44Y151 | BEL = B6LUT;
INST "IN[32]..L_eDelay/S_1" LOC = SLICE_X4Y103 | BEL = C6LUT;
INST "IN[33]..L_eDelay/S_1" LOC = SLICE_X8Y103 | BEL = C6LUT;
INST "IN[34]..L_eDelay/S_1" LOC = SLICE_X12Y103 | BEL = C6LUT;
INST "IN[35]..L_eDelay/S_1" LOC = SLICE_X14Y103 | BEL = C6LUT;
INST "IN[36]..L_eDelay/S_1" LOC = SLICE_X16Y103 | BEL = C6LUT;
INST "IN[37]..L_eDelay/S_1" LOC = SLICE_X18Y103 | BEL = C6LUT;


// 04/03 @ 23:52:56
NET "CONN_L_Delay<0>"
ROUTE="{3;1;5vsx95tff1136;d281a9be!-1;-148488;-66848;S!0;-48;72!1;-32;"
"120!2;-143;-712!3;-452;0!4;-2460;2896!5;-218;-28356!6;0;-58872!7;0;"
"-58872!8;5509;-37745!9;6537;-2383!10;2788;-1996!11;127;1157!12;277;1611!"
"13;683;-88;L!}";
NET "CONN_R_Delay<0>"
ROUTE="{3;1;5vsx95tff1136;8a388ec3!-1;-148488;24272;S!0;-48;72!1;-32;120!"
"2;-143;-712!3;-452;0!4;-1780;-2512!5;-692;-2424!6;2980;-36!7;1898;-3900!"
"8;-868;-32836!9;5445;-37745!10;8368;-9848!11;-4547;-4331!12;2764;-5744!"
"13;-1008;-1040!14;683;-288;L!}";

        ...
        die weiteren ROUTE-Constraints befinden
        sich auf der beigefügten CD
        ...
```

Listing B.4: matrix.ucf

```
#these constraints define the matrix positions and routes
#timing constraints are commented out, cause the routes force them.
#that can be verified but it takes the ISE Suite two days to place&route
#the project, if timing constraints are present

INST "matrix_inst/matrix_col/*" U_SET = MCOL;
INST "matrix_inst/matrix_row_33/*" U_SET = MR33;
INST "matrix_inst/matrix_row_32/*" U_SET = MR32;
INST "matrix_inst/matrix_row_31/*" U_SET = MR31;
INST "matrix_inst/matrix_row_30/*" U_SET = MR30;
INST "matrix_inst/matrix_row_29/*" U_SET = MR29;
INST "matrix_inst/matrix_row_28/*" U_SET = MR28;
INST "matrix_inst/matrix_row_27/*" U_SET = MR27;
INST "matrix_inst/matrix_row_26/*" U_SET = MR26;
INST "matrix_inst/matrix_row_25/*" U_SET = MR25;
INST "matrix_inst/matrix_row_24/*" U_SET = MR24;
INST "matrix_inst/matrix_row_23/*" U_SET = MR23;
INST "matrix_inst/matrix_row_22/*" U_SET = MR22;
INST "matrix_inst/matrix_row_21/*" U_SET = MR21;
INST "matrix_inst/matrix_row_20/*" U_SET = MR20;
INST "matrix_inst/matrix_row_19/*" U_SET = MR19;
INST "matrix_inst/matrix_row_18/*" U_SET = MR18;
INST "matrix_inst/matrix_row_17/*" U_SET = MR17;
INST "matrix_inst/matrix_row_16/*" U_SET = MR16;
INST "matrix_inst/matrix_row_15/*" U_SET = MR15;
INST "matrix_inst/matrix_row_14/*" U_SET = MR14;
INST "matrix_inst/matrix_row_13/*" U_SET = MR13;
INST "matrix_inst/matrix_row_12/*" U_SET = MR12;
INST "matrix_inst/matrix_row_11/*" U_SET = MR11;
INST "matrix_inst/matrix_row_10/*" U_SET = MR10;
INST "matrix_inst/matrix_row_9/*" U_SET = MR9;
INST "matrix_inst/matrix_row_8/*" U_SET = MR8;
INST "matrix_inst/matrix_row_7/*" U_SET = MR7;
INST "matrix_inst/matrix_row_6/*" U_SET = MR6;
INST "matrix_inst/matrix_row_5/*" U_SET = MR5;
INST "matrix_inst/matrix_row_4/*" U_SET = MR4;
INST "matrix_inst/matrix_row_3/*" U_SET = MR3;
INST "matrix_inst/matrix_row_2/*" U_SET = MR2;
INST "matrix_inst/matrix_row_1/*" U_SET = MR1;
INST "matrix_inst/matrix_row_0/*" U_SET = MR0;

INST "matrix_inst/matrix_row_33/matrix_clb_0" RLOC_ORIGIN = X13Y97;
INST "matrix_inst/matrix_row_32/matrix_clb_0" RLOC_ORIGIN = X13Y96;
INST "matrix_inst/matrix_row_31/matrix_clb_0" RLOC_ORIGIN = X13Y95;
INST "matrix_inst/matrix_row_30/matrix_clb_0" RLOC_ORIGIN = X13Y94;
INST "matrix_inst/matrix_row_29/matrix_clb_0" RLOC_ORIGIN = X13Y93;
INST "matrix_inst/matrix_row_28/matrix_clb_0" RLOC_ORIGIN = X13Y92;
INST "matrix_inst/matrix_row_27/matrix_clb_0" RLOC_ORIGIN = X13Y91;
INST "matrix_inst/matrix_row_26/matrix_clb_0" RLOC_ORIGIN = X13Y90;
INST "matrix_inst/matrix_row_25/matrix_clb_0" RLOC_ORIGIN = X13Y89;
INST "matrix_inst/matrix_row_24/matrix_clb_0" RLOC_ORIGIN = X13Y88;
INST "matrix_inst/matrix_row_23/matrix_clb_0" RLOC_ORIGIN = X13Y87;
INST "matrix_inst/matrix_row_22/matrix_clb_0" RLOC_ORIGIN = X13Y86;
INST "matrix_inst/matrix_row_21/matrix_clb_0" RLOC_ORIGIN = X13Y85;
INST "matrix_inst/matrix_row_20/matrix_clb_0" RLOC_ORIGIN = X13Y84;
INST "matrix_inst/matrix_row_19/matrix_clb_0" RLOC_ORIGIN = X13Y83;
INST "matrix_inst/matrix_row_18/matrix_clb_0" RLOC_ORIGIN = X13Y82;
INST "matrix_inst/matrix_row_17/matrix_clb_0" RLOC_ORIGIN = X13Y81;
INST "matrix_inst/matrix_row_16/matrix_clb_0" RLOC_ORIGIN = X13Y80;
INST "matrix_inst/matrix_row_15/matrix_clb_0" RLOC_ORIGIN = X13Y79;
INST "matrix_inst/matrix_row_14/matrix_clb_0" RLOC_ORIGIN = X13Y78;
INST "matrix_inst/matrix_row_13/matrix_clb_0" RLOC_ORIGIN = X13Y77;
INST "matrix_inst/matrix_row_12/matrix_clb_0" RLOC_ORIGIN = X13Y76;
INST "matrix_inst/matrix_row_11/matrix_clb_0" RLOC_ORIGIN = X13Y75;
INST "matrix_inst/matrix_row_10/matrix_clb_0" RLOC_ORIGIN = X13Y74;
INST "matrix_inst/matrix_row_9/matrix_clb_0" RLOC_ORIGIN = X13Y73;
INST "matrix_inst/matrix_row_8/matrix_clb_0" RLOC_ORIGIN = X13Y72;
INST "matrix_inst/matrix_row_7/matrix_clb_0" RLOC_ORIGIN = X13Y71;
INST "matrix_inst/matrix_row_6/matrix_clb_0" RLOC_ORIGIN = X13Y70;
INST "matrix_inst/matrix_row_5/matrix_clb_0" RLOC_ORIGIN = X13Y69;
INST "matrix_inst/matrix_row_4/matrix_clb_0" RLOC_ORIGIN = X13Y68;
INST "matrix_inst/matrix_row_3/matrix_clb_0" RLOC_ORIGIN = X13Y67;
```

```
73 INST "matrix_inst/matrix_row_2/matrix_clb_0" RLOC_ORIGIN = X13Y66;
74 INST "matrix_inst/matrix_row_1/matrix_clb_0" RLOC_ORIGIN = X13Y65;
75 INST "matrix_inst/matrix_row_0/matrix_clb_0" RLOC_ORIGIN = X13Y64;
76 INST "matrix_inst/matrix_col/matrix_col_0" RLOC_ORIGIN = X77Y64;
77
78 INST "*/matrix_clb_0/*_LUT" RLOC = X0Y0;
79 INST "*/matrix_clb_1/*_LUT" RLOC = X2Y0;
80 INST "*/matrix_clb_2/*_LUT" RLOC = X4Y0;
81 INST "*/matrix_clb_3/*_LUT" RLOC = X6Y0;
82 INST "*/matrix_clb_4/*_LUT" RLOC = X8Y0;
83 INST "*/matrix_clb_5/*_LUT" RLOC = X10Y0;
84 INST "*/matrix_clb_6/*_LUT" RLOC = X12Y0;
85 INST "*/matrix_clb_7/*_LUT" RLOC = X14Y0;
86 INST "*/matrix_clb_8/*_LUT" RLOC = X16Y0;
87 INST "*/matrix_clb_9/*_LUT" RLOC = X18Y0;
88 INST "*/matrix_clb_10/*_LUT" RLOC = X20Y0;
89 INST "*/matrix_clb_11/*_LUT" RLOC = X22Y0;
90 INST "*/matrix_clb_12/*_LUT" RLOC = X24Y0;
91 INST "*/matrix_clb_13/*_LUT" RLOC = X26Y0;
92 INST "*/matrix_clb_14/*_LUT" RLOC = X28Y0;
93 INST "*/matrix_clb_15/*_LUT" RLOC = X30Y0;
94 INST "*/matrix_clb_16/*_LUT" RLOC = X32Y0;
95 INST "*/matrix_clb_17/*_LUT" RLOC = X34Y0;
96 INST "*/matrix_clb_18/*_LUT" RLOC = X36Y0;
97 INST "*/matrix_clb_19/*_LUT" RLOC = X38Y0;
98 INST "*/matrix_clb_20/*_LUT" RLOC = X40Y0;
99 INST "*/matrix_clb_21/*_LUT" RLOC = X42Y0;
100 INST "*/matrix_clb_22/*_LUT" RLOC = X44Y0;
101 INST "*/matrix_clb_23/*_LUT" RLOC = X46Y0;
102 INST "*/matrix_clb_24/*_LUT" RLOC = X48Y0;
103 INST "*/matrix_clb_25/*_LUT" RLOC = X50Y0;
104 INST "*/matrix_clb_26/*_LUT" RLOC = X52Y0;
105 INST "*/matrix_clb_27/*_LUT" RLOC = X54Y0;
106 INST "*/matrix_clb_28/*_LUT" RLOC = X56Y0;
107 INST "*/matrix_clb_29/*_LUT" RLOC = X58Y0;
108 INST "*/matrix_clb_30/*_LUT" RLOC = X60Y0;
109 INST "*/matrix_clb_31/*_LUT" RLOC = X62Y0;
110
111 INST "*/matrix_col_0/*_LUT" RLOC = X0Y0;
112 INST "*/matrix_col_1/*_LUT" RLOC = X0Y1;
113 INST "*/matrix_col_2/*_LUT" RLOC = X0Y2;
114 INST "*/matrix_col_3/*_LUT" RLOC = X0Y3;
115 INST "*/matrix_col_4/*_LUT" RLOC = X0Y4;
116 INST "*/matrix_col_5/*_LUT" RLOC = X0Y5;
117 INST "*/matrix_col_6/*_LUT" RLOC = X0Y6;
118 INST "*/matrix_col_7/*_LUT" RLOC = X0Y7;
119 INST "*/matrix_col_8/*_LUT" RLOC = X0Y8;
120 INST "*/matrix_col_9/*_LUT" RLOC = X0Y9;
121 INST "*/matrix_col_10/*_LUT" RLOC = X0Y10;
122 INST "*/matrix_col_11/*_LUT" RLOC = X0Y11;
123 INST "*/matrix_col_12/*_LUT" RLOC = X0Y12;
124 INST "*/matrix_col_13/*_LUT" RLOC = X0Y13;
125 INST "*/matrix_col_14/*_LUT" RLOC = X0Y14;
126 INST "*/matrix_col_15/*_LUT" RLOC = X0Y15;
127 INST "*/matrix_col_16/*_LUT" RLOC = X0Y16;
128 INST "*/matrix_col_17/*_LUT" RLOC = X0Y17;
129 INST "*/matrix_col_18/*_LUT" RLOC = X0Y18;
130 INST "*/matrix_col_19/*_LUT" RLOC = X0Y19;
131 INST "*/matrix_col_20/*_LUT" RLOC = X0Y20;
132 INST "*/matrix_col_21/*_LUT" RLOC = X0Y21;
133 INST "*/matrix_col_22/*_LUT" RLOC = X0Y22;
134 INST "*/matrix_col_23/*_LUT" RLOC = X0Y23;
135 INST "*/matrix_col_24/*_LUT" RLOC = X0Y24;
136 INST "*/matrix_col_25/*_LUT" RLOC = X0Y25;
137 INST "*/matrix_col_26/*_LUT" RLOC = X0Y26;
138 INST "*/matrix_col_27/*_LUT" RLOC = X0Y27;
139 INST "*/matrix_col_28/*_LUT" RLOC = X0Y28;
140 INST "*/matrix_col_29/*_LUT" RLOC = X0Y29;
141 INST "*/matrix_col_30/*_LUT" RLOC = X0Y30;
142 INST "*/matrix_col_31/*_LUT" RLOC = X0Y31;
143
144 INST "*/K_LUT" BEL = D6LUT;
145 INST "*/H1_LUT" BEL = A6LUT;
146 INST "*/H2_LUT" BEL = B6LUT;
```

```
147
148 INST "matrix_inst/MATRIX_LUT_*" LOC = SLICE_X77Y97;
149 INST "*/MATRIX_LUT_TG" BEL = D6LUT;
150 INST "*/MATRIX_LUT_H1_1" BEL = C6LUT;
151 INST "*/MATRIX_LUT_H1_0" BEL = A6LUT;
152
153 INST "matrix_inst/TRG_DELAY_*" BEL = D6LUT;
154 INST "matrix_inst/TRG_SIGNAL_GEN" LOC = SLICE_X78Y97;
155 INST "matrix_inst/TRG_DELAY_12" LOC = SLICE_X79Y97;
156 INST "matrix_inst/TRG_DELAY_23" LOC = SLICE_X79Y96;
157 INST "matrix_inst/TRG_DELAY_34" LOC = SLICE_X79Y95;
158 INST "matrix_inst/TRG_DELAY_45" LOC = SLICE_X79Y94;
159 INST "matrix_inst/TRG_DELAY_56" LOC = SLICE_X79Y93;
160 INST "matrix_inst/TRG_DELAY_67" LOC = SLICE_X79Y92;
161 INST "matrix_inst/TRG_DELAY_LAST" LOC = SLICE_X79Y91;
162
163 #NET "*/h1*" MAXDELAY = 496 ps;
164 #NET "*/tg*" MAXDELAY = 323 ps;
165 #NET "*/tg_31*" MAXDELAY = 494 ps;
166
167 #NET "*/h2_and*" MAXDELAY = 116 ps;
168 #NET "*/h1_and*" MAXDELAY = 634 ps;
169
170 #NET "matrix_inst/matrix_col*/leap*" MAXDELAY = 188 ps;
171 #NET "matrix_inst/matrix_row*/leap*" MAXDELAY = 117 ps;
172
173 #NET "*/h2<0>" MAXDELAY = 493 ps;
174 #NET "*/h2<1>" MAXDELAY = 520 ps;
175 #NET "*/h2<2>" MAXDELAY = 493 ps;
176 #NET "*/h2<3>" MAXDELAY = 582 ps;
177 #NET "*/h2<4>" MAXDELAY = 496 ps;
178 #NET "*/h2<5>" MAXDELAY = 520 ps;
179 #NET "*/h2<6>" MAXDELAY = 493 ps;
180 #NET "*/h2<7>" MAXDELAY = 520 ps;
181 #NET "*/h2<8>" MAXDELAY = 493 ps;
182 #NET "*/h2<9>" MAXDELAY = 582 ps;
183 #NET "*/h2<10>" MAXDELAY = 496 ps;
184 #NET "*/h2<11>" MAXDELAY = 520 ps;
185 #NET "*/h2<12>" MAXDELAY = 493 ps;
186 #NET "*/h2<13>" MAXDELAY = 520 ps;
187 #NET "*/h2<14>" MAXDELAY = 493 ps;
188 #NET "*/h2<15>" MAXDELAY = 582 ps;
189 #NET "*/h2<16>" MAXDELAY = 496 ps;
190 #NET "*/h2<17>" MAXDELAY = 738 ps; #The only bad one, ranges from 728(20x) to 736(12x) to 738(2x)
191 #NET "*/h2<18>" MAXDELAY = 496 ps;
192 #NET "*/h2<19>" MAXDELAY = 590 ps;
193 #NET "*/h2<20>" MAXDELAY = 493 ps;
194 #NET "*/h2<21>" MAXDELAY = 520 ps;
195 #NET "*/h2<22>" MAXDELAY = 493 ps;
196 #NET "*/h2<23>" MAXDELAY = 515 ps;
197 #NET "*/h2<24>" MAXDELAY = 496 ps;
198 #NET "*/h2<25>" MAXDELAY = 590 ps;
199 #NET "*/h2<26>" MAXDELAY = 493 ps;
200 #NET "*/h2<27>" MAXDELAY = 520 ps;
201 #NET "*/h2<28>" MAXDELAY = 493 ps;
202 #NET "*/h2<29>" MAXDELAY = 515 ps;
203 #NET "*/h2<30>" MAXDELAY = 496 ps;
204
205 // 03/26 @ 14:09:41
206 NET "matrix_inst/tg_29<17>"
207 ROUTE="{3;1;5vsx95tff1136;f03db9f8!-1;-3784;44720;S!0;-843;-824!1;-1738;"
208 "3944!2;1738;-256!3;843;296;L!}";
209 NET "matrix_inst/tg_28<17>"
210 ROUTE="{3;1;5vsx95tff1136;be6321d7!-1;-3784;41520;S!0;-843;-824!1;-1738;"
211 "3944!2;1738;-256!3;843;296;L!}";
212 NET "matrix_inst/tg_29<25>"
213 ROUTE="{3;1;5vsx95tff1136;4b83e23b!-1;49720;44720;S!0;-843;-824!1;-1738;"
214 "3944!2;1738;-256!3;843;296;L!}";
215
216         ...
217         die weiteren ROUTE-Constraints befinden
218         sich auf der beigefügten CD
219         ...
```

Listing B.5: generated.ucf

```
# == MT 0 ==

INST "IN[0]..MT/CLB_00" RLOC_ORIGIN = X5Y3;
INST "IN[0]..MT/*" U_SET = MT0;

NET "CONN_RP[0]" LOC = N33;
NET "CONN_RN[0]" LOC = M33;
NET "CONN_LP[0]" LOC = W24;
NET "CONN_LN[0]" LOC = V24;

INST "IN[0]..MT/ORL1_00/*_LUT" RLOC = X-1Y-1;
INST "IN[0]..MT/ORL1_01/*_LUT" RLOC = X-1Y1;
INST "IN[0]..MT/ORL1_02/*_LUT" RLOC = X-1Y3;
INST "IN[0]..MT/ORL1_03/*_LUT" RLOC = X-1Y5;
INST "IN[0]..MT/ORL1_04/*_LUT" RLOC = X-1Y7;
INST "IN[0]..MT/ORL1_05/*_LUT" RLOC = X-1Y9;
INST "IN[0]..MT/ORL1_06/*_LUT" RLOC = X-1Y11;
INST "IN[0]..MT/ORL1_07/*_LUT" RLOC = X-1Y13;
INST "IN[0]..MT/ORL1_08/*_LUT" RLOC = X-1Y15;
INST "IN[0]..MT/ORL1_09/*_LUT" RLOC = X-1Y17;
INST "IN[0]..MT/ORL1_10/*_LUT" RLOC = X-1Y19;
INST "IN[0]..MT/ORL1_11/*_LUT" RLOC = X-1Y21;
INST "IN[0]..MT/ORL1_12/*_LUT" RLOC = X-1Y23;
INST "IN[0]..MT/ORL1_13/*_LUT" RLOC = X-1Y25;
INST "IN[0]..MT/ORL1_14/*_LUT" RLOC = X-1Y27;
INST "IN[0]..MT/ORL1_15/*_LUT" RLOC = X-1Y29;
INST "IN[0]..MT/ORL1_16/*_LUT" RLOC = X-1Y31;
INST "IN[0]..MT/ORL1_17/*_LUT" RLOC = X-1Y33;
INST "IN[0]..MT/ORL1_18/*_LUT" RLOC = X-1Y35;
INST "IN[0]..MT/ORL1_19/*_LUT" RLOC = X-1Y37;
INST "IN[0]..MT/ORL1_20/*_LUT" RLOC = X-1Y39;
INST "IN[0]..MT/ORL1_21/*_LUT" RLOC = X-1Y41;
INST "IN[0]..MT/ORL1_22/*_LUT" RLOC = X-1Y43;
INST "IN[0]..MT/ORL1_23/*_LUT" RLOC = X-1Y45;
INST "IN[0]..MT/ORL1_24/*_LUT" RLOC = X-1Y47;
INST "IN[0]..MT/ORL1_25/*_LUT" RLOC = X-1Y49;
INST "IN[0]..MT/ORL1_26/*_LUT" RLOC = X-1Y51;
INST "IN[0]..MT/ORL1_27/*_LUT" RLOC = X-1Y53;
INST "IN[0]..MT/ORL1_*/OR_LUT" BEL = D6LUT;


INST "IN[0]..MT/ORL2_00/*_LUT" RLOC = X-1Y1;
INST "IN[0]..MT/ORL2_01/*_LUT" RLOC = X-1Y5;
INST "IN[0]..MT/ORL2_02/*_LUT" RLOC = X-1Y9;
INST "IN[0]..MT/ORL2_03/*_LUT" RLOC = X-1Y13;
INST "IN[0]..MT/ORL2_04/*_LUT" RLOC = X-1Y17;
INST "IN[0]..MT/ORL2_05/*_LUT" RLOC = X-1Y21;
INST "IN[0]..MT/ORL2_06/*_LUT" RLOC = X-1Y25;
INST "IN[0]..MT/ORL2_07/*_LUT" RLOC = X-1Y29;
INST "IN[0]..MT/ORL2_08/*_LUT" RLOC = X-1Y33;
INST "IN[0]..MT/ORL2_09/*_LUT" RLOC = X-1Y37;
INST "IN[0]..MT/ORL2_10/*_LUT" RLOC = X-1Y41;
INST "IN[0]..MT/ORL2_11/*_LUT" RLOC = X-1Y45;
INST "IN[0]..MT/ORL2_12/*_LUT" RLOC = X-1Y49;
INST "IN[0]..MT/ORL2_13/*_LUT" RLOC = X-1Y53;
INST "IN[0]..MT/ORL2_*/OR_LUT" BEL = A6LUT;


INST "IN[0]..MT/ORL3_00/*_LUT" RLOC = X-1Y3;
INST "IN[0]..MT/ORL3_01/*_LUT" RLOC = X-1Y11;
INST "IN[0]..MT/ORL3_02/*_LUT" RLOC = X-1Y19;
INST "IN[0]..MT/ORL3_03/*_LUT" RLOC = X-1Y27;
INST "IN[0]..MT/ORL3_04/*_LUT" RLOC = X-1Y35;
INST "IN[0]..MT/ORL3_05/*_LUT" RLOC = X-1Y43;
INST "IN[0]..MT/ORL3_06/*_LUT" RLOC = X-1Y51;
INST "IN[0]..MT/ORL3_*/OR_LUT" BEL = B6LUT;


INST "IN[0]..MT/ORL4_00/*_LUT" RLOC = X-1Y4;
INST "IN[0]..MT/ORL4_01/*_LUT" RLOC = X-1Y20;
INST "IN[0]..MT/ORL4_02/*_LUT" RLOC = X-1Y36;
INST "IN[0]..MT/ORL4_03/*_LUT" RLOC = X-1Y52;
```

```
73 INST "IN[0]..MT/ORL4_*/OR_LUT" BEL = B6LUT;
74
75
76 INST "IN[0]..MT/ORL5_00/*_LUT" RLOC = X-1Y10;
77 INST "IN[0]..MT/ORL5_01/*_LUT" RLOC = X-1Y42;
78 INST "IN[0]..MT/ORL5_*/OR_LUT" BEL = B6LUT;
79
80
81 INST "IN[0]..MT/ORL6_00/*_LUT" RLOC = X0Y27;
82 INST "IN[0]..MT/ORL6_*/OR_LUT" BEL = C6LUT;
83
84          ...
85          der vollständige Quellcode befindet
86          sich auf der beigefügten CD
87          ...
```

Abkürzungsverzeichnis

6U	Größenangabe für VME-Module, h × b = 233 × 160mm^2
AJAX	Asynchronous JavaScript And XML, ein Konzept der asynchronen Datenübertragung zwischen einem Browser und dem Server.
ANSI	American National Standards Institute
ASIC	Application Specific Integrated Circuit
BEL	Basic Element of Logic, eine Constraints-Definition, um ein logisches Element innerhalb einer Slice zu positionieren.
BMS	Beam Momentum Station
CERN	Conseil Européen pour la Recherche Nucléaire (Europäisches Kernforschungszentrum)
CFD	Constant Fraction Diskriminator
CMOS	Complementary Metal Oxide Semiconductor
CNGS	CERN Neutrinos to Gran Sasso
COMPASS	COmmon Muon Proton Apparatus for Structure and Spectroscopy
CPLD	Complex Programmable Logic Device
DAQ	Data Acquisition (Datenerfassung)
DDR2	Double Data Rate Random Access Memory (Version 2), die Informationen werden sowohl bei steigender als auch bei fallender Flanke des Clock-Signals übertragen.
DIS	Deep Inelastic Scattering (Tiefinelastische Streuung)
DNP	Dynamic Nuclear Polarisation, diese Methode ist ausführlich in [2] beschrieben.
DSP	Digital Signal Processing (digitale Signalverarbeitung)
ECL	Emitter Coupled Logic bezeichnet eine Logikfamilie der Digitaltechnik. In diesem Fall ist ein mit ECL-Technik entwickelter Bustreiber für extrem schnelle differentielle Signalübertragungen gemeint.

EEPROM Electrically Erasable Programmable Read-Only Memory, nicht flüchtiger Speicherbaustein, elektrisch löschbar und überschreibbar.

EMC European Muon Collaboration

EPROM Erasable Programmable Read-Only Memory, nicht flüchtiger Speicherbaustein. Er lässt sich mittels UV-Licht löschen und elektrisch beschreiben.

FFT Fast Fourier Transform

FPGA Field Programmable Gate Array, regelmäßige Anordnung von programmierbaren Logikbausteinen.

GAL Generic Array Logic

GANDALF Generic Advanced Numerical Device for Analytic and Logic Functions

IC Integrated Circuit

IPN Orsay Institut de Physique Nucleaire d'Orsay

LAS Large Angle Spectrometer

LHC Large Hadron Collider

LOC Location Constraints, eine Constraints-Definition, um ein logisches Element in einer bestimmten Slice zu platzieren.

LVDS Low Voltage Differential Signaling ist ein Schnittstellen-Standard für Hochgeschwindigkeits-Datenübertragungen. Er zeichnet sich durch differenzielle und relativ geringe Spannungspegel aus (1.2V $\pm$ 0.3V).

MWPCs multi-wire proportional chambers (Vieldrahtproportionalkammern)

NIM Nuclear Instrumentation Modul-Standard (Signal-Standard) oder Nuclear Instruments and Methods (Journal)

PAL Programmable Array Logic

PHP Personal Home Page Tools, eine Skriptsprache, die hauptsächlich bei der Web-Programmierung zum Einsatz kommt.

PLA Programmable Logic Array

PLD Programmable Logic Device

PROM Programmable Read-Only Memory

QDRII Quad Data Rate Random Access Memory (Version II), Speicherbaustein mit zwei Ports zum gleichzeitigen Lesen und Schreiben. Jeder Bus arbeitet nach dem DDR-Prinzip.

RICH Ring Imaging CHerenkov Detector

RPD Rückstoß-Proton-Detektor

SLAC Stanford Linear Accelerator Center

SMC Spin Muon Collaboration

SPS Super Proton Synchrotron

Tcl Tool command language, ist eine einfache, jedoch sehr leistungsfähige, interpretierte Scriptsprache, die als Open Source zur Verfügung steht.

TDC Time-to-Digital-Converter sind Messkreise, die kurze Zeitintervalle messen und in eine digitale Ausgabe umwandeln.

TDL Tapped Delay Line

TRF Target Rest Frame (Laborsystem eines ruhenden Targets)

VFD Vorderflanken Diskriminator

VITA VMEbus International Trade Association

VME Versa Module Eurocard, standardisiertes Bussystem für 19-Zoll-Einschubgehäuse. Die ursprüngliche Spezifikation wurde inzwischen zu VME64x weiterentwickelt.

VXS VMEbus Switched Serial, Erweiterung des VME-Standards um serielle Hochgeschwindigkeitsleitungen.

Literaturverzeichnis

[1] P. Abbon et al. *The compass experiment at cern.* Nuclear Instruments and Methods in Physics Research Section A, 577:455–518, 2007.

[2] G. Abragam and M. Goldman. *Principles of dynamic nuclear polarisation.* Reports on Progress in Physics, 41:395, 1978.

[3] J. Ashman et al. *A measurement of the spin asymmetry and determination of the structure function g1 in deep inelastic muon-proton scattering.* Physics Letters B, 206:364–370, 1988.

[4] S. Bachmann et al. *Performance of gem detectors in high intensity particle beams.* Nuclear Instruments and Methods in Physics Research Section A, 470:548–561, 2001.

[5] A.R. Baldwin and R. Madley. *An analog mean-timer circuit for use with large-volume scintillation counters.* Nuclear Instruments and Methods, 171:149–152, 1980.

[6] C. Bernet, A. Bravar, J. Hannappel, D.v. Harrach, R. Hermann, E. Kabuß, F. Klein, A. Korzenev, M. Leberig, M. Ostrick, J. Pretz, R. Windmolders, and J. Zhao. *The compass trigger system for muon scattering.* Nuclear Instruments and Methods in Physics Research Section A, 550:217 – 240, 2005.

[7] J.K. Bienlein and R. Wiesendanger. *Einführung in die Struktur der Materie: Kerne, Teilchen, Molekühle, Festkörper.* Vieweg+Teubner, 2002.

[8] Franco Bradamante. *Compass measurements of transverse spin effects.* Compass Talks, 2009.

[9] G. Charpak, L. Dick, and L. Feuvrais. *Location of the position of a particle trajectory in a scintillator.* Nuclear Instruments and Methods, 15:323–326, 1962.

[10] Lattice Semiconductor Corporation. *1000EA, 1000E and 1000 Family Architectural Description.*

[11] Forschungszentrum DESY. *HERA Grafiken.* http://www.desy.de/images/content/e8/e76/imageobject185/strukturfunktion_hr_ger.jpg.

[12] N. Doble et al. *The upgraded muon beam at the SPS.* Nuclear Instruments and Methods in Physics Research Section A, 343:351–362, 1994.

[13] C. Amsler et al. (Particle Data Group). *Review of particle physics.* Physics Letters B, 667:1, 2008.

[14] J. Faust and R.S. Larsen. *A time compensator for large scintillation counters.* Nuclear Instruments and Methods, 116:365–368, 1974.

[15] R.L. Garwin et al. *Observations of the failure of conservation of parity and charge conjugation in meson decays: the magnetic moment of the free muon.* Physical Review, 105:1415–1417, 1957.

[16] L. Gatignon. *M2 user guide.* http://gatignon.home.cern.ch/gatignon/M2manual.html, 2000.

[17] L. Gatignon. *The modifications to the M2 beam for COMPASS.* http://sl.web.cern.ch/SL/eagroup/NewM2/main.html, 2000.

[18] Boris Grube. *A trigger control system for COMPASS and a measurement of the transverse polarization of L and X hyperons from quasi-real photo-production.* Technische Universität München, 2006.

[19] Xilinx Inc. *Achieving Higher System Performance with the Virtex-5 Family of FPGAs.* Xilinx Dokumentation, wp245, 2006.

[20] Xilinx Inc. *Virtex-5 FPGA Constraints Guide.* Xilinx Dokumentation, ug625, 2009.

[21] Xilinx Inc. *Virtex-5 FPGA Data Sheet: DC and Switching Characteristics.* Xilinx Dokumentation, ds202, 2009.

[22] Xilinx Inc. *Virtex-5 FPGA User Guide.* Xilinx Dokumentation, ug190, 2009.

[23] Xilinx Inc. *QDR II SRAM Interface for Virtex-5 Devices.* http://www.xilinx.com/support/documentation/application_notes/xapp853.pdf, 2010.

[24] Aram Kotzinian. *Remarks on acceptance effects in asymmetry extraction.* Compass Notes, 2007.

[25] Carolin Kurig. *Aufbau, Test und Weiterentwicklung des Triggersystems für das elektromagnetische Kalorimeter ECAL1 des COMPASS-Experiments.* Diplomarbeit, Institut für Kernphysik, Johannes-Gutenberg-Universität Mainz, 2007.

[26] T. Mizuno, J. Okamura, and A. Torimi. *Experimental Study of Threshold Voltage Fluctuation Due to Statistical Variation of Channel Dopant Number in MOSFET's.* IEEE Transactions on Electron Devices, 41, 1994.

[27] Ole J. Nähle. *Faserhodoskope im COMPASS-Experiment zum Nachweis von Teilchenspuren innerhalb des Primärstrahls.* Rheinischen Friedrich-Wilhelms-Universität Bonn, 2002.

[28] W.D. Peterson. *The VMEbus Handbook.* VITA, 1997.

[29] Stephane Platchkov. *Spin physics with compass.* Compass Talks, 2010.

[30] Bogdan Povh et al. *Teilchen und Kerne.* Springer Verlag Berlin Heidelberg, 2008.

[31] Swtpc6800 (Pseudonym). *Programmable Logic Device*. Wikimedia Commons: http://upload.wikimedia.org/wikipedia/commons/f/f0/Programmable_Logic_Device.svg.

[32] Andreas Richter. *Zeitkalibration der Faserhodoskope und Qualitätsüberprüfung der Daten für das COMPASS-Experiment*. Diplomarbeit, Physikalisches Institut der Universität Erlangen-Nürnberg, 2006.

[33] Sebastian Schopferer. *Entwicklung eines hochauflösenden Transientenrekorders*. Diplomarbeit, Physikalisches Institut der Albert-Ludwigs-Universität Freiburg, 2009.

[34] R.E. Taylor, J.I. Friedman, and H.W. Kendall. *Deep inelastic scattering: Acknowledgments*. Review of Modern Physics, 63:629, 1991.

[35] D. Thers et al. *Micromegas as a large microstrip detector for the COMPASS experiment*. Nuclear Instruments and Methods in Physics Research Section A, 469:133–146, 2001.

[36] H.P. von Gunten et al. *A fast analog mean-timer*. Nuclear Instruments and Methods in Physics Research Section A, 234:512–516, 1985.

[37] R.M. Wagner. *Commissioning of Silicon Detectors for the COMPASS Experiment at CERN*. Diplomarbeit, Technische Universität München, 2001.

[38] Dr. Winkler. *Programmierbare Logikschaltungen*. http://www2.informatik.hu-berlin.de/sv/lehre/ti1/ti1d/download/doc/ti1d4.pdf.