Schnelle Spurerkennung im H1–Detektor

Diplomarbeit von Thomas Wolff

II. Institut für Experimentalphysik

der Universität Hamburg

Hamburg, Mai 1989

"Ich bin der Geist, der stets verneint!
Und das mit Recht; denn alles, was entsteht,
Ist wert, das es zugrunde geht;
Drum besser wär's, das nichts entstünde.
So ist denn alles, was ihr Sünde,
Zerstörung, kurz, das Böse nennt,
Mein eigentliches Element."

J.W. v.Goethe, Faust, der Tragödie erster Teil

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung und Motivation

Das Deutsche Elektronen Synchrotron (DESY) in Hamburg baut zur Erforschung der Struktur der Materie einen Elektron Proton Speicherring (HERA[1]).

HERA erschließt einen neuen kinematischen Bereich für Untersuchungen der Wechselwirkung zwischen Hadronen und Leptonen. In der neuen Anlage sollen Stöße zwischen 820 GeV Protonen und 30 GeV Elektronen möglich sein [2]. Dies entspricht einer Schwerpunktsenergie von ca. 314 GeV. Die wichtigsten Daten des Speicherringes sind in der Tabelle 1 zusammengefaßt.

Parameter :	Protonenring	Elektronenring	Einheit
Nominelle Strahlenergie	820	30	GeV
Schwerpunktsenergie	315	315	GeV
Einschußenergie	40	14	GeV
Anzahl der Wechselwirkungspunkte	4	4	
Luminosität	$1.5 \cdot 10^{31}$	$1.5 \cdot 10^{31}$	$cm^{-2}s^{-1}$
Umfang	6335	6335	m
Umlauffrequenz	47.31	47.31	kHz
max. Anzahl der Teilchenpakete	220	220	
Teilchenstrom	163	58	mA
Teilchenanzahl pro Paket	$1 \cdot 10^{11}$	$3.5 \cdot 10^{10}$	
horizontale Emittanz (σ_x)	$0.86 \cdot 10^{-8}$	$3.5 \cdot 10^{-8}$	π rad
vertikale Emittanz (σ_z)	$0.43 \cdot 10^{-8}$	$0.69 \cdot 10^{-8}$	π rad
Länge des Teilchenpakets	110	7.8	mm
Energieverlust pro Umlauf	$1.4 \cdot 10^{-10}$	71.9	MeV
Hochfrequenz	208.194	499.6655	MHz
Füllzeit	20	15	min
Stärke der Magneten	4.649	0.1426	T
Vakuum	$< 10^{-8}$	$< 10^{-8}$	mbar

Tabelle 1: Parameter von HERA

Der Speicherring bietet vier Wechselwirkungszonen, von denen zu Beginn zwei für Hochenergieexperimente, H1 und Zeus, zur Verfügung stehen. Um die angestrebte Luminosität von $1.5 * 10^{31} cm^{-2} s^{-1}$ erreichen zu können, wird der

[1] Hadron Elektron Ring Anlage

Ring mit 210 Teilchenpaketen (Bunche) gefüllt. Das bedeutet für die Experimente eine besondere Herausforderung, da sich an den Kreuzungspunkten die Teilchenpakete alle 96ns treffen und deshalb sehr schnelle Entscheidungslogiken (Trigger) benötigt werden. Da die Reaktionszeiten der Detektorkomponenten jedoch in der Größenordnung von einigen μs liegen, werden die von den Detektorkomponenten gemessenen Daten fließbandartig zwischengespeichert und verarbeitet, d.h. die Daten werden in Puffer geschrieben, und dort periodisch mit einer bestimmten Frequenz, z.B. der Bunchfrequenz, weiter geschoben.

1.1 e–p–Physik, die an HERA gemessen werden kann

Im Bereich der Impulsüberträge, die bei HERA auftreten, kann davon ausgegangen werden, daß ein Elektron mit einem Quark eines Protons wechselwirkt, da die Wellenlänge des Elektrons wesentlich kleiner ist als ein Proton. Im Partonenmodell der Hadronen wird die tief inelastische Lepton–Hadronstreuung als elastische Streuung eines Leptons an einer punktförmigen Struktur (Parton), interpretiert.

Die starke und elektroschwache Wechselwirkung werden im 'Standardmodell' als lokale Eichtheorie der Eichgruppe $SU_{3_c} \otimes SU_2 \otimes U_1$ beschrieben. SU_{3_c} beschreibt dabei die starke, $SU_2 \otimes U_1$ die elektroschwache Wechselwirkung, also die Vereinheitlichung von elektromagnetischer und schwacher Wechselwirkung. Obwohl viele Vorhersagen dieses Modells mit experimentellen Beobachtungen übereinstimmen, ist es in mancher Hinsicht noch wenig zufriedenstellend:

1. Es gibt in diesem Modell eine Vielzahl von Teilchen, die man in verschiedene 'Familien' anordnen kann. Das Modell sagt nicht wieviele Familien es gibt, es macht jedoch die Aussage, daß die Anzahl der Quark– und Leptonenfamilien gleich ist.

2. Es gibt in diesem Modell eine Vielzahl von freien Parametern (Massen, Mischungswinkel,Kopplungskonstanten)

3. Das Teilchen, das beim Prozeß der Massenentstehung der Eichbosonen und der Fermionen eine entscheidende Rolle spielt, das Higgs, ist noch nicht gefunden. Das Modell verlangt eine Higgs Masse kleiner 1TeV.

4. Das Modell enthält nicht alle bekannten Kräfte. So fehlt in diesem Modell die Gravitation.

Diese Aufzählung ist nicht vollständig, zeigt aber einige Punkte auf, deren Klärung von erheblichem Interesse ist.

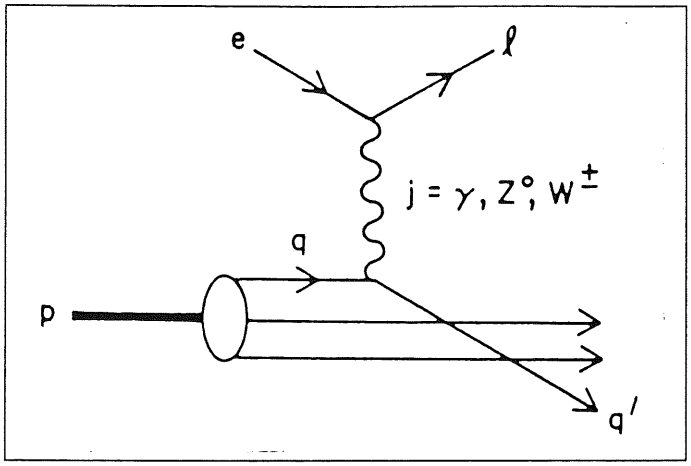

Abbildung 1: Elektron–Quark–Streuung

Der HERA–Speicherring bietet die Möglichkeit, experimentell einige offene Fragen zu klären und Voraussagen von theoretischen Modellen insbesondere im Rahmen der QCD (**Q**uantenchromo**d**ynamik) zu testen. Bei den bei HERA möglichen, hohen Impulsüberträgen von $Q^2_{max} \approx 10^5 GeV^2$ wechselwirkt das Elektron nicht mit dem Proton als ganzem, sondern mit einem der Quarks. Die folgenden Prozesse werden im wesentlichen erwartet:

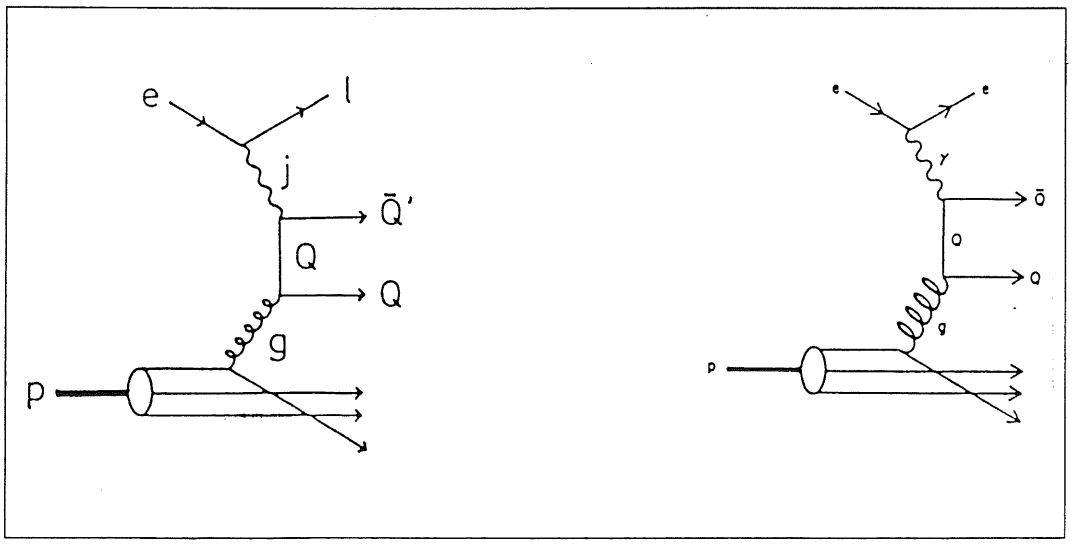

Abbildung 2: Photon–Gluon–Fusion und W–Gluon–Fusion

1. Elektron–Quark–Streuung, siehe Abb. 1

2. Photon–Gluon–Fusion, siehe Abb. 2
3. W–Gluon–Fusion, siehe Abb. 2

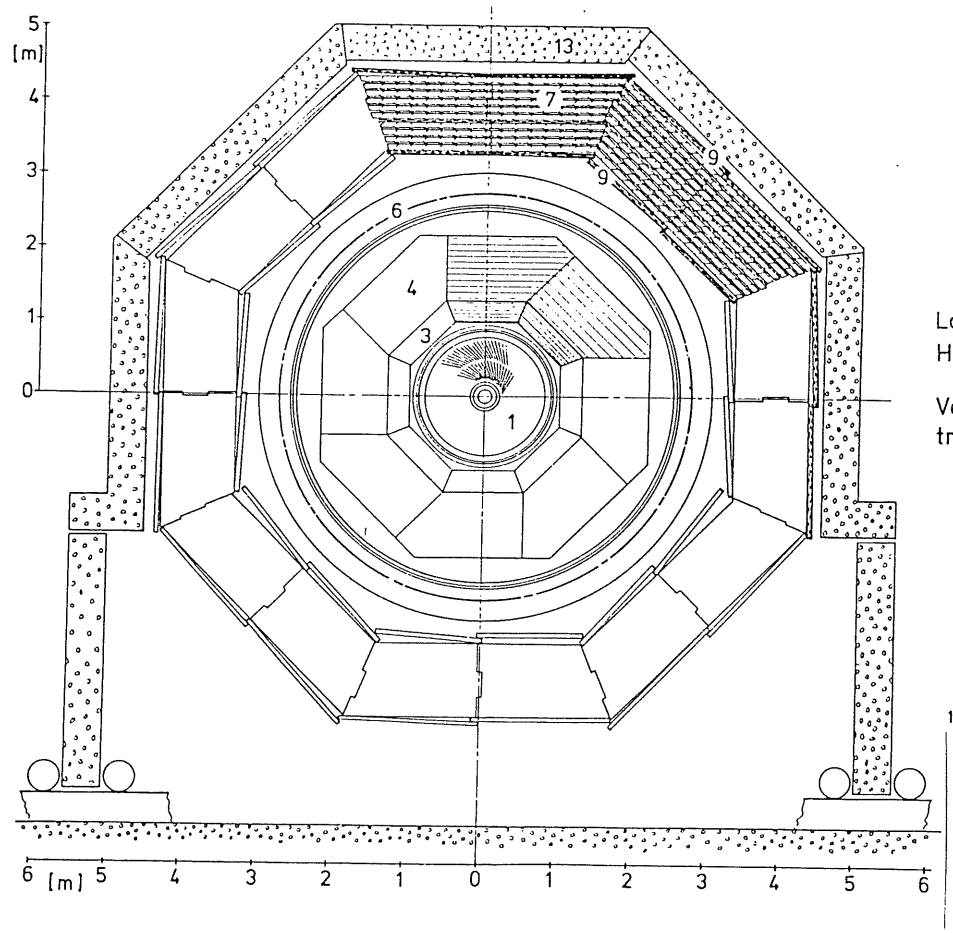

Abbildung 3: Der H1 Detektor, R–Φ–Ebene

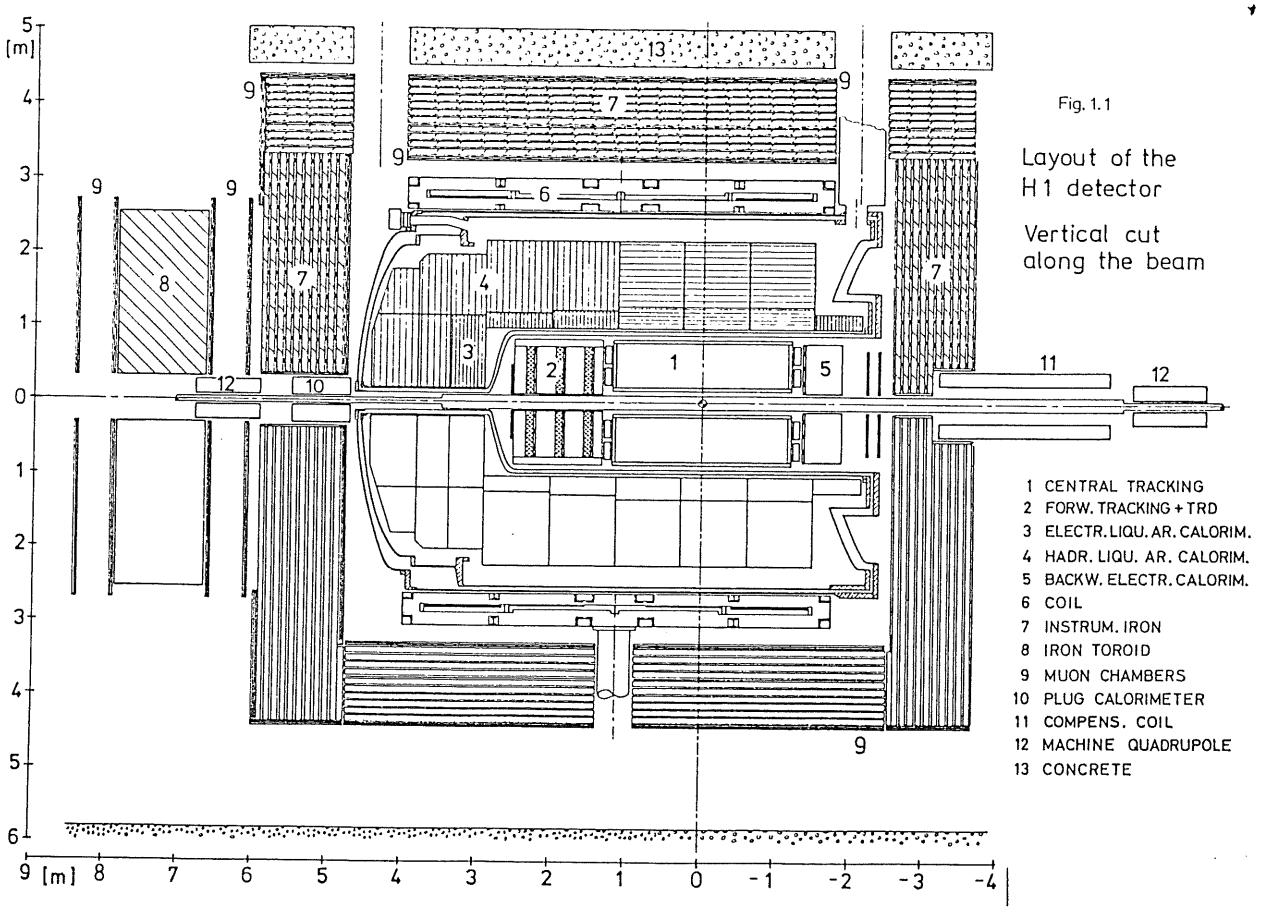

Abbildung 4: Der H1 Detektor, R–Z–Ebene

1.2 HERA–Detektoren

Weil man mit HERA in der Lage ist, Strukturen bis etwa 10^{-18}cm zu untersuchen, werden besondere Anforderungen an die Detektoren bezüglich der x– und der Q^2– Auflösung gestellt. Wegen der großen Impulsdifferenz zwischen Elektronen und Protonen sind die Detektoren asymmetrisch aufgebaut, siehe Abb 4.

Die beiden Detektoren H1 und Zeus sind für leicht differierende Messungen optimiert. Der Detektor ZEUS ist für sogenannte inklusive Messungen besser geeignet, da er die bessere Energieauflösung für Hadronen hat und ein Kalorimeter mit $e/h \approx 1$ besitzt. Der Detektor H1 ist für sogenannte exklusive Messungen – die Energie der einzelnen Teilchen wird bestimmt – besser geeignet. Da das H1 Kalorimeter feiner segmentiert ist, kann man z.B. in Jets einzelne Leptonen an der Schauerform erkennen.

Um die entstehenden Elementarteilchen beobachten zu können, haben die Detektoren zwei verschiedene Komponenten. Zum einen besitzen sie Kalorimeter in denen die Energie und die Richtung von Teilchen gemessen wird. Zum anderen ist um den Wechselwirkungspunkt der Spurdetektor angeordnet um die Ladung und den Impuls einzelner Spuren zu messen.

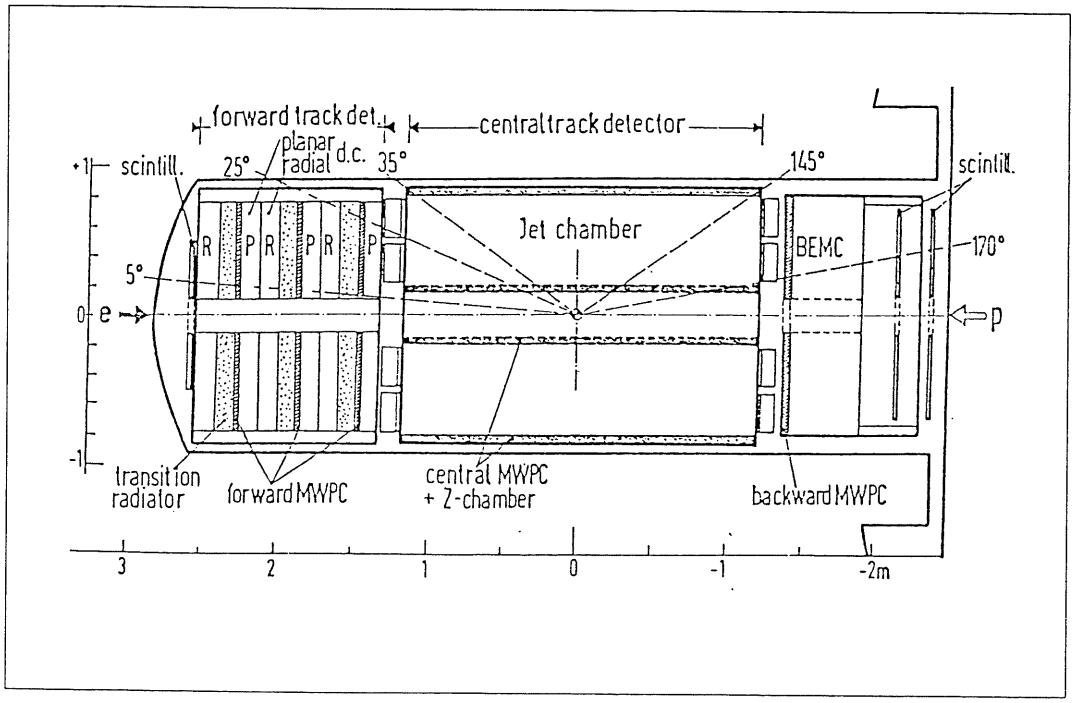

Abbildung 5: Die zentralen Spurkammern

Die Komponenten des H1–Detektors sind:

1. Der Spurdetektor zur Rekonstruktion der Ladung, des Impulses und des Entstehungsortes der Spuren. Er besteht
   1.) aus dem zentralen Spurdetektor, der aus der zentralen Driftkammer und den inneren und äußeren Proportional– und Z–Kammern zusammengesetzt ist, siehe Abb. 5 und
   2.) aus den vorwärts Drift– und Proportionalkammern.

2. Das elektromagnetische Flüssig–Argon–Blei–Kalorimeter.

3. Das hadronische Flüssig–Argon–Edelstahl–Kalorimeter.

4. Das elektromagnetische Kalorimeter in Rückwärtsrichtung. Es ist ein Blei–Szintillator–Kalorimeter, da es sich außerhalb des Kryostaten befindet.

5. Die supraleitende Spule. Sie erzeugt ein Feld von 1.2 T und hat einen Durchmesser von 6m.

6. Das Eisenjoch dient dem Schluß des magnetischen Feldes und ist außerdem mit Streamerrohrkammern versehen, so daß es als ergänzendes Kalorimeter (Tail Catcher) wirkt. Des weiteren dient es als Absorbermaterial für die Myonenkammern.

7. Der Eisentoroid.

8. Die Myonenkammern in Vorwärtsrichtung.

9. Die Abschlußkalorimeter. Sie benötigt man zur Messung von Teilchen, die unter kleinen Winkeln den Wechselwirkungspunkt verlassen.

In den Abbildungen 3 und 4 ist der H1–Detektor in der R–Φ– und in der R–Z–Projektion zu sehen. Die Z–Position liegt bei der R–Φ–Projektion ungefähr am Vertex. Der in dieser Arbeit beschriebene Spurtrigger baut auf der zentralen Driftkammer auf, welche im folgenden Kapitel beschrieben wird.

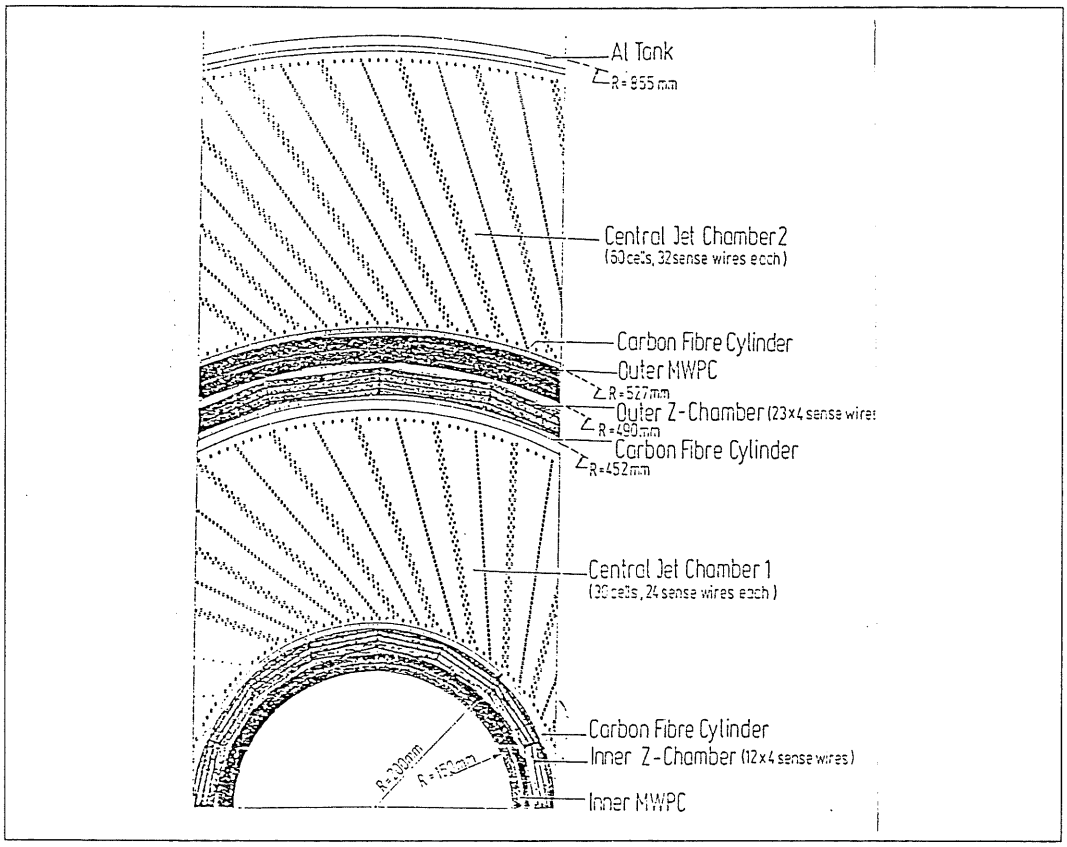

Abbildung 6: Die zentrale Driftkammer

1.3 Die zentrale Jetkammer (CJC)

Die zentrale Jetkammer (CJC[2]) hat die Aufgabe, geladene Teilchen nachzuweisen, deren Impulse möglichst genau zu bestimmen und die Teilchen innerhalb eines Jets aufzulösen. Die Kammer soll zusätzlich Informationen über Ereignistopologien sowie Signaturen neuer Teilchen liefern. Der Aufbau der CJC ist in Abb.6 gezeigt. Sie besitzt 2640 Meßdrähte, die in zwei großen Ringen (CJC1 und CJC2) um den Mittelpunkt der R–Φ–Ebene parallel zur Z–Achse angeordnet sind. Innerhalb dieser Ringe sind die Drähte so angeordnet, daß sie Zellen bilden, wobei der innere Ring aus 30 Zellen mit 24 Lagen und der äußere Ring aus 60 Zellen mit 32 Lagen besteht. Die Zellen sind durch Kathodendrähte getrennt.

[2]Central Jet Chamber

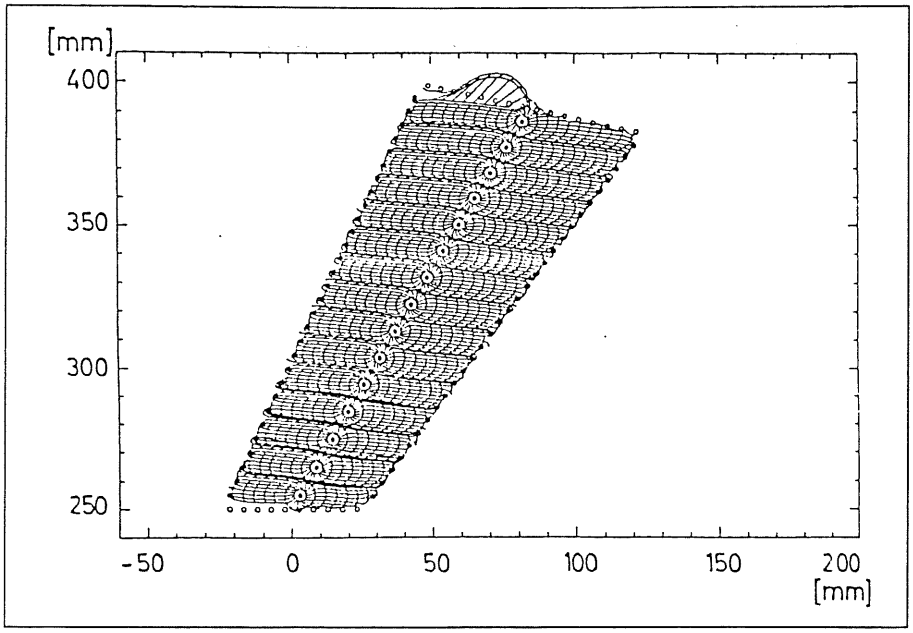

Abbildung 7: Äquipotentiallinien der CJC und Isochronen einer Zelle der inneren Driftkammer

Um jeden Signaldraht sind 4 Potentialdrähte, siehe Abb. 6, angeordnet. Die Potentiallinien und Isochronen von Meßdrähten sind in Abb. 7 dargestellt. Die Meßdrähte einer Zelle stellen eine Fläche in der R–Z–Ebene dar, die Meßdrahtebene (Sensewireplane). Die Meßdrahtebene steht in einem solchen Winkel gegen eine radiale Fläche, daß ungefähr[3] der Lorentzwinkel, den die driftenden Elektronen erfahren, kompensiert wird. (Der Lorentzwinkel ist der Winkel zwischen der Richtung der Driftgeschwindigkeit und dem elektrischen Feld) Die Kompensation hat den Vorteil, daß der Lorentzwinkel nicht numerisch ausgeglichen werden muß. Ein weiterer Vorteil ist, daß Spuren deren Impuls größer als 400 MeV/c ist sicher die Meßdrahtebene schneiden. Diese Eigenschaft ist für den im weiteren beschriebenen Trigger besonders wichtig.

1.4 Ziel der Arbeit

HERA arbeitet mit einer Kreuzungsfrequenz der Teilchenpakete (Bunchcrossingrate) von ca. 10 MHz, d.h. 10 millionenmal pro Sekunde kann ein Ereignis

[3]Der Lorentzwinkel hängt vom Gas und vom Magnetfeld und vom elektrischen Feld ab. Die zentrale Driftkammer kompensiert bei Xenon fast, bei einer Argon–Ethan Mischung nicht vollständig den Winkel [27]

auftreten, das analysiert werden soll. Andererseits hat man nur die Möglichkeit, ca. 5 Ereignisse pro Sekunde abzuspeichern. Des weiteren benötigt die spätere Auswertung von Ereignissen ca. 30 Sekunden CPU–Zeit auf einem Großrechner wie der IBM3090 hier im DESY–Rechenzentrum.

Die Aufgabe des Triggers besteht darin, physikalisch interessante Ereignisse vom Untergrund zu trennen und so die anfallende Datenmenge zu reduzieren. Obgleich die Bunchcrossingrate etwa 10MHz beträgt, sind interessante Physikereignisse weniger als einige pro Sekunde zu erwarten. Die typische Untergrundrate liegt bei einigen kHz. Da schnellen Triggern mit einer Entscheidungszeit von etwa einer μs [4] nur ungenaue Daten zur Verfügung stehen, werden Trigger in mehreren Zeitentscheidungsstufen benutzt, wobei die späteren Stufen immer genauere Daten benutzen und mehr Zeit zur Verfügung haben, wodurch die Entscheidung immer sicherer wird.

Die Datenreduktion des Triggers darf jedoch nicht zu Lasten der gemessenen Physik gehen, d.h es sollen nur solche Daten verworfen werden die uninteressant sind. Dabei handelt es sich zum Beispiel um solche Daten, die von kosmischen Teilchen oder aus einer Wechselwirkung eines Strahlteilchens mit einem Restgasmolekül oder mit dem Strahlrohr herrühren. All diesen Ereignissen ist gemein, daß sie in der Regel nicht aus dem Wechselwirkungspunkt stammen oder zu einem verkehrten Zeitpunkt auftreten.

Das Triggerkonzept von H1 sieht verschiedene Triggerstufen (Level) vor, siehe Abb.8. Die erste Stufe (First Level, L1) arbeitet totzeitfrei, d.h. zu jedem Bunchcrossing gibt es ein Triggersignal. Natürlich kann dieser Trigger nicht so genau Spuren rekonstruieren und hat auch weniger Daten zur Verfügung als spätere Triggerstufen, die aus Microcomputern aufgebaut sind. Die Triggerentscheidung des Triggers der ersten Stufe wird am Ende einer Datenkette (Pipeline) erzeugt, d.h. es gibt zu jedem Bunchcrossing eine Triggerentscheidung, aber diese kommt mit einer bestimmten zeitlichen Verzögerung. Eine der Ursachen für diese Verschiebung liegt darin, daß die Bunchperiode zwar 96ns beträgt, aber die maximale Driftzeit in der Kammer von $1,2\mu s$ abgewartet werden muß.

Ziel dieser Diplomarbeit ist es, für den **F**ull–**S**ize–**P**rototyp (FSP) der H1–CJC einen Trigger der ersten Stufe (First–Level–Trigger) zu entwickeln, der auch im H1 Experiment eingesetzt werden kann. Gerade in der Kombination mit anderen Subdetektoren oder einem R–Z–Trigger an der CJC erwartet man eine hohe Unterdrückung des Untergrundes. Außerdem bietet ein Spurtrigger

[4] 2.2 μs durch periodisch verschiebendes Zwischenspeichern (Pipelining), trotz 100ns Periodendauer

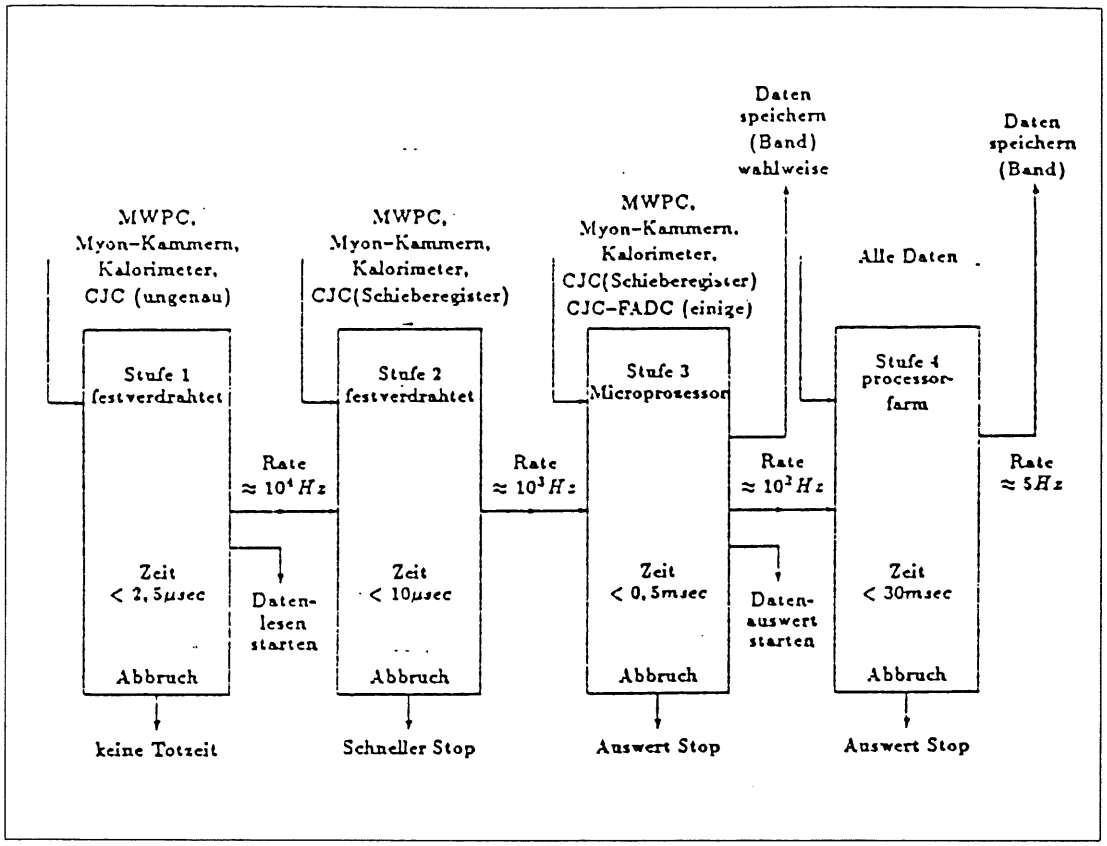

Abbildung 8: Das Triggerkonzept von H1

an der CJC die Möglichkeit Ereignisse zu triggern, die in den Kalorimetern nur eine geringe Energiedeposition haben.

2 Ein Trigger der ersten Stufe für H1

Die wichtigste Aufgabe eines Spurtriggers der ersten Stufe ist es, interessante Ereignisse von uninteressanten zu unterscheiden und so eine Datenreduktion zu erreichen. Eine weitere Aufgabe des Triggers ist es, Daten für die weitere Analyse bereitzustellen. So soll ein späterer Mikroprozessor–Trigger auf die Triggerdaten des Spurtriggers der ersten Stufe zugreifen können, um z.B. den Bereich begrenzen zu können, in dem er Spuren suchen muß.

Ein Spurtrigger der ersten Stufe basiert auf der einfachen logischen Verknüpfung von Signaldrahtinformationen. Diese werden so verknüpft, daß in der R–Φ–Projektion Bereiche entstehen in denen Spuren von geladenen Teilchen aus den e–p–Wechselwirkungen erwartet werden. Diese Bereiche nennt man Triggerstraßen. Die Triggerstraßen zielen auf den Wechselwirkungspunkt und hängen von der Bahnkrümmung, d.h. dem transversalen Impuls der nachzuweisenden Teilchen ab. Die im folgenden Kapitel beschriebenen Trigger unterscheiden sich, außer in der technischen Umsetzung, in der unterschiedlichen Definition dieser Triggerstraßen.

2.1 Der durch den Trigger zu unterdrückende Untergrund

Die Forderung an den Trigger der ersten Stufe, wie auch an alle anderen Trigger, ist die möglichst starke Abweisung des Untergrundes. Den größten Teil des Untergrundes erzeugt der Protonenstrahl. Im Vergleich der Produktionsmechanismen stellt man zwei dominante Quellen für den Untergrund fest:

1. Wechselwirkungen zwischen Strahlprotonen und Restgasmolekülen im Strahlrohr.

2. Protonen, die der Strahl verliert und die in der Wandung des Strahlrohres wechselwirken. Diese produzieren hadronische Schauer.

Einige erwartete Ereignisraten sind im folgenden aufgeführt. Die Annahmen für die Strahlgaswechselwirkung hängen stark von dem Vakuum im Rohr ab. Bei einem bei HERA angestrebten Vakuum von 10^{-9} Torr und $2*10^{13}$ Protonen im Ring erhält man folgende Strahlgasereignisraten pro Meter [28]:

1. Gas Wasserstoff : 240 Hz/m

2. Gas Sauerstoff : 1600 Hz/m

Wenn für den Protonenstrahl eine Lebensdauer von 3,6h angenommen wird, dann ergibt sich ein durchschnittlicher Protonenverlust von 10^5 s^{-1} m^{-1}, der wesentlich höher ist, als die Rate der Strahl–Gas–Wechselwirkung. Dieser Protonenverlust führt zu Strahl–Rohr–Wechselwirkungen die durch einen R–Φ–Spurtrigger vollständig unterdrückt werden können. Ein Großteil der Sekundärteilchen von Strahl–Gas oder Strahl–Rohr wird jedoch durch die verschiedenen Elemente des Detektors und des Ringes zurückgehalten.

Die Untergrundabweisung der im folgenden Kapitel beschriebenen Spurtrigger wird durch die räumliche Eingrenzung des Wechselwirkungspunktes (Vertex) in der R–Φ–Ebene erreicht. Die Eingrenzung des Wechselwirkungspunktes in Z–Richtung geschieht automatisch dadurch, daß sie nur auf einen bestimmten Bereich empfindlich sind. Dieser Bereich ist, wie in Abb. 9 zu sehen, durch die beiden Grenzradien, welche die Trigger zum Messen benutzen, eingeschränkt. Bei einem Trigger, dessen innerer Grenzradius $\approx 20cm$ und dessen äußerer Grenzradius $\approx 65cm$ ist, ist der empfindliche Bereich in Z–Richtung $\approx 4m$.

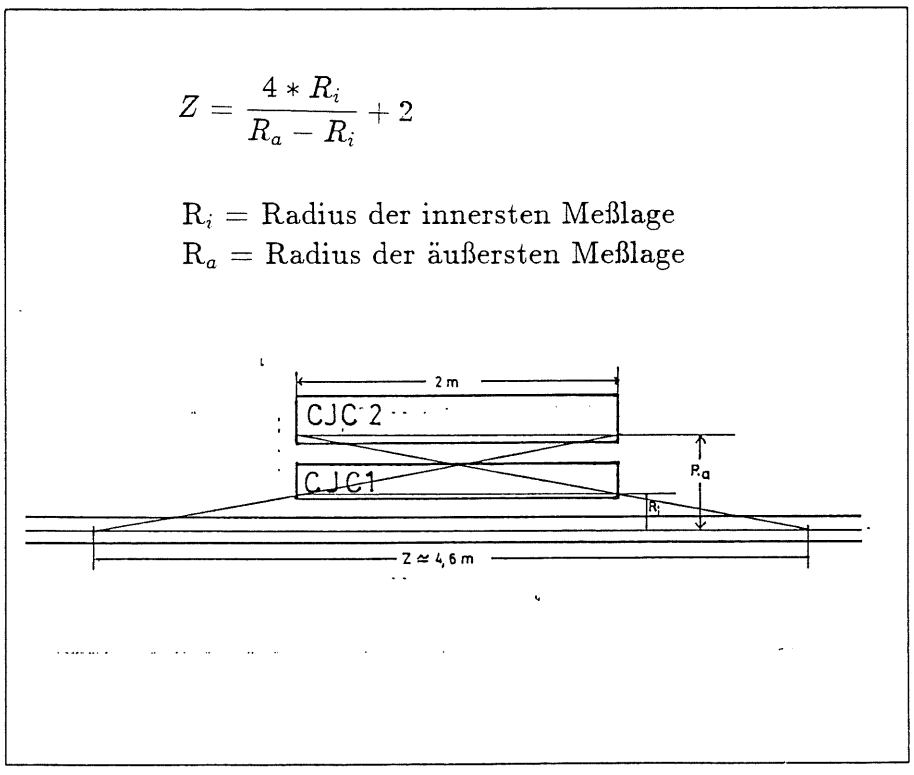

Abbildung 9: Eingrenzung der Wechselwirkungszone in Z

Strahl–Gas–Wechselwirkungen, die innerhalb dieses Bereiches liegen, können mit einem solchen Trigger nicht unterdrückt werden. Daraus resultiert eine erwartete Rate von Untergrundereignissen von 960 Hz für Strahl–Wasserstoff–Reaktionen und 4800 Hz für Strahl–Sauerstoff–Reaktionen, die dieser Trigger zuläßt. Eine Reduktion dieser Rate wäre nur durch die Kombination eines R–Φ–Vertex–Triggers mit einem R–Z–Vertex–Trigger zu erreichen.

2.2 Die Kommunikation zwischen den verschiedenen Triggern des Detektors

Die in Kapitel 1.2 gezeigten Detektorkomponenten besitzen teilweise eigene unabhängige Trigger, wie z.B. den Trigger der ersten Stufe an der zentralen Driftkammer. Die verschiedenen Triggersignale dieser Subdetektortrigger müssen an einer Stelle synchronisiert werden.

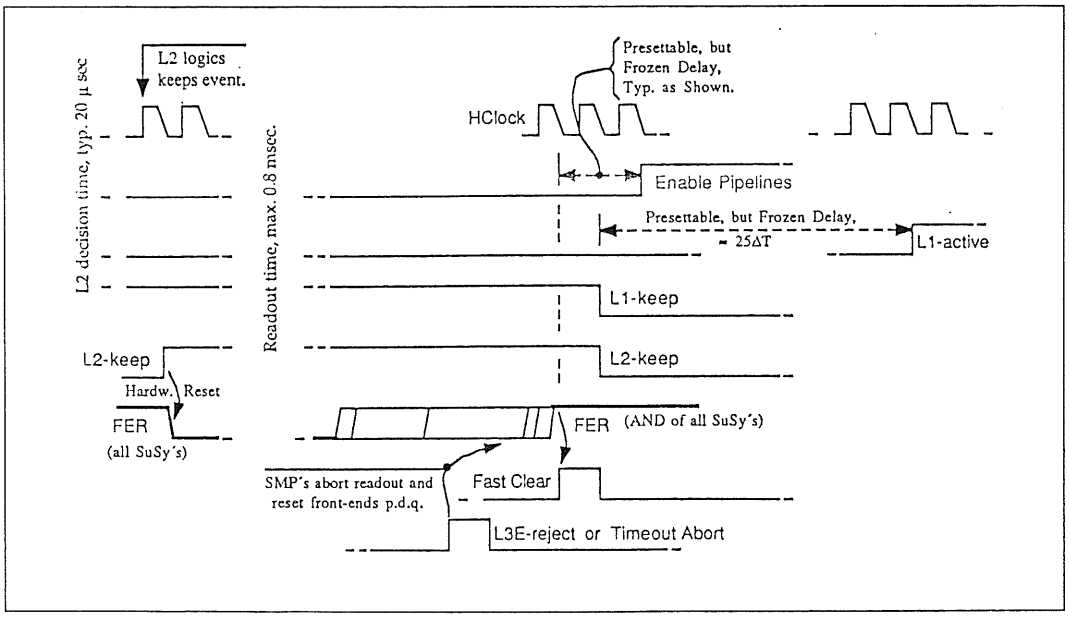

Abbildung 10: Unterbrochene CJC–Auslese

Dazu sendet jeder Subdetektortrigger seine Teilentscheidung innerhalb einer Stufe an den 'Master Trigger', der die Gesamtentscheidung der einzelnen Triggerstufen fällt. Nachdem eine positive Triggerentscheidung der Stufe Eins (L1 Trigger) gefällt worden ist, wird die Datennahme des Detektors angehalten. Die zweite Triggerstufe startet bei einer positiven Triggerentscheidung (L2 keep)

die Datenauslese aus den FADC[5]–Karten[6] in die Scannerkarte,[7] im anderen Fall wird die Datennahme wieder gestartet (Fast Clear), siehe Abb. 10.

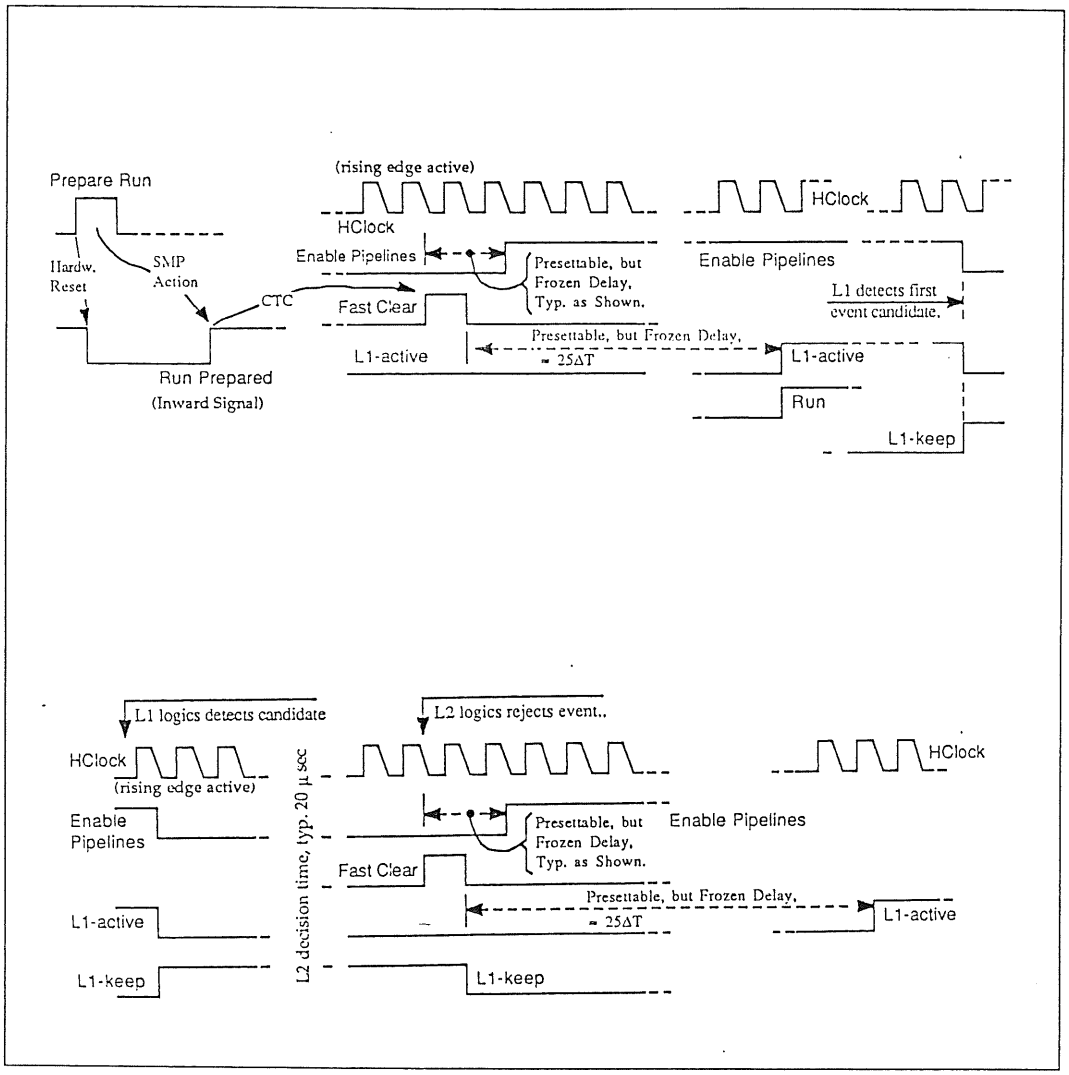

Abbildung 11: Start der CJC–Auslese (oben) und Abbruch der CJC–Auslese durch Triggerstufe 2

Gleichzeitig mit der Auslese der FADC–Karten wird die Auslese der Triggerbits, die Informationen der Trigger der ersten Stufe, in einen Pufferspeicher

[5]**F**lash **A**nalog **D**igital **C**onverter

[6]Auf diesen Karten werden die analog verstärkten Signale der Meßdrähte digitalisiert und zwar mit 100MHz und einer Amplitudenauflösung von 8–Bit

[7]Diese Karte liest die Daten aus den FADC–Karten aus und speichert diese zwischen.

gestartet. Dies geschieht typischerweise nach 2,5 μs (Puffertiefe auf den FADC–Karten). Wenn ein positives Signal (L3 keep) der dritten Triggerstufe kommt, werden die gepufferten Daten aus dem Triggerpuffer und der Scannerkarte in Subdetektormasterrechner übertragen. Die Auslese der Triggerdaten ist parallel zur Auslese der Kammerdaten aus den FADC–Karten. Die beiden Prozesse der Triggerauslese und der Datenauslese laufen asynchron. Deswegen muß es zwischen Triggerlogik, Triggerauslese und Kammerauslese einen 'Handshake' geben, siehe Abb. 10 bis 12, d.h. Signale, die den Start und das Ende eines Prozesses melden. Zum Handshake existieren allerdings nur die oben beschriebenen Trigger bzw. Rücksetzsignale. Deswegen muß die Synchronisation der parallel laufenden Prozesse durch die bekannte Zeitdauer dieser geschehen. Die zeitlichen Vorgaben macht dabei die Auslese der FADC's, siehe Abb. 12.

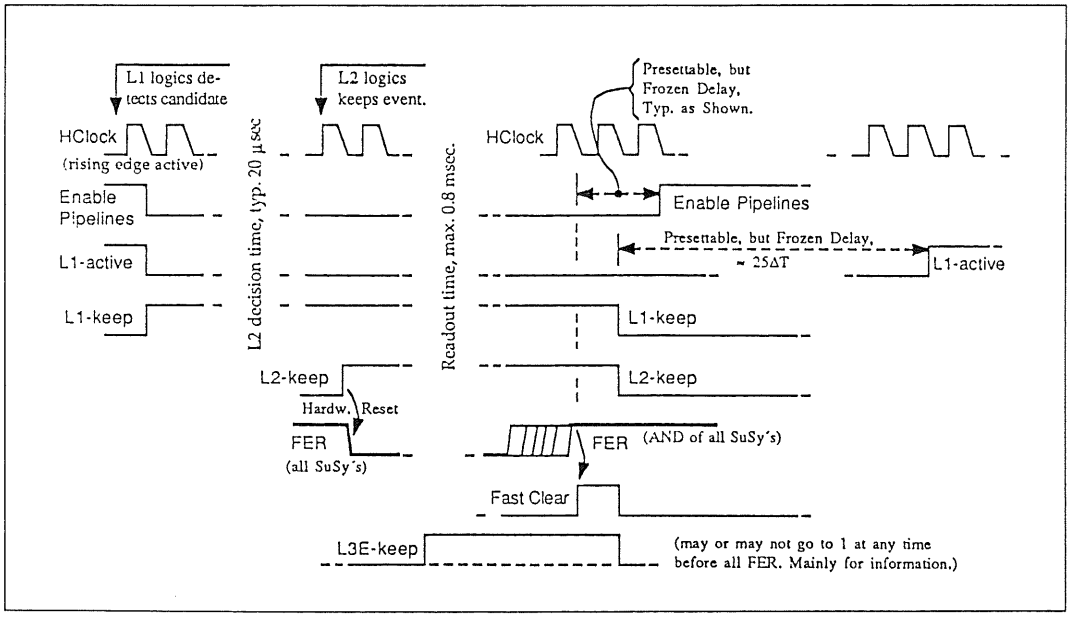

Abbildung 12: Nicht unterbrochene CJC–Auslese

Um einen eindeutigen Handshake sicherzustellen, ist es wichtig, daß die Triggerentscheidung in einer festen zeitlichen Beziehung zum Ereignis steht, da sonst keine Synchronisation der verschiedenen Trigger untereinander und der Trigger mit der Datennahme erzielt wird.

2.3 Größen, die einen Trigger der ersten Stufe kennzeichnen

Die folgenden Größen beschreiben die Leistungsfähigkeit eines Spurtriggers der ersten Stufe.

1. Auflösung in R, *Radialauflösung.* Die Radialauflösung ist definiert als die "Breite der Triggerstraßen am Wechselwirkungspunkt". Um Strahl-Strahlrohr—Wechselwirkungen unterdrücken zu können, muß diese Größe kleiner sein als der Strahlrohrdurchmesser (10cm).

2. Auflösung in t, *Zeitauflösung.* Mit Zeitauflösung ist hier die zeitliche Beziehung zwischen Ereigniszeitpunkt und Triggerzeitpunkt gemeint, gemessen in Zeiteinheiten von 96ns Dauer. Die Zeit innerhalb eines Zyklus von 96ns wird im Trigger der ersten Stufe nicht aufgelöst.

3. Auflösung in p, *Impulsauflösung.* Die Impulsauflösung eines Triggers setzt sich aus der Breite der Triggerstraßen und der Randunschärfe der Triggerstraßen zusammen. Im Idealfall sollten die Triggerstraßen scharfe Kanten haben, d.h. innerhalb der Triggerstraßen sollten alle Spuren gefunden werden, außerhalb jedoch keine, siehe Abb. 13.

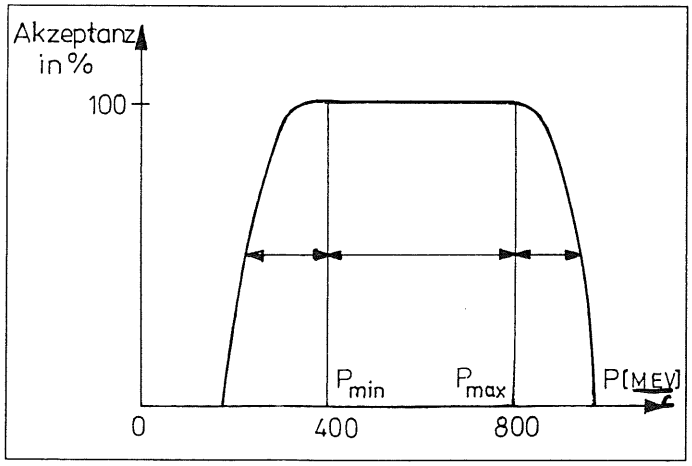

Abbildung 13: Impulsauflösung und Randunschärfe von Triggerstraßen

4. Auflösung in ϕ, *Winkelauflösung.* Die Winkelauflösung sagt aus, wie groß der Winkelbereich ist, in dem eine Spur konstanten Impulses gemessen wurde. Sie ist gleich der Anzahl der Triggerstraßen pro Impuls-

bereich. Normalerweise ist die Winkelauflösung für alle Impulse zwischen den Grenzimpulsen eine Konstante[8].

Diese ersten beiden Größen stellen die theoretische Güte des Triggers dar. Die Impuls und Winkelauflösung sind von besonderem Interesse, wenn die Triggerdaten von höheren Triggerstufen benutzt werden sollen. So kann ein Trigger der dritten Stufe z.B. die Informationen des Triggers der ersten Stufe mitbenutzen und braucht nur in den Raumbereichen die Daten zu analysieren, in denen der Trigger der ersten Stufe Spuren angezeigt hat.

5. Toleranz gegenüber Kammerfehlern (Inefficiencies). Aus den Punkten 1 bis 4 kann man eine Größe ableiten, die im allgemeinen Akzeptanz genannt wird.

6. Fest vorgegeben ist der minimale Transversalimpuls des Triggers für Spuren aus dem Wechselwirkungspunkt, die noch erkannt werden sollen,in unserem Fall 750MeV/c.

7. Der Aufwand, respektive der Preis. Der Trigger sollte möglichst preisgünstig sein. Ein kompakter Trigger hat den weiteren Vorteil, daß er weniger Quellen für Fehler liefert, denn die statistischen Fehler steigen mit der Anzahl der Bauteile. Dieser statistische Bauteilfehler ist in der Auswertung ein systematischer Fehler, der schwer zu analysieren ist.

8. Datenreduktion. Der Datenreduktionsfaktor ist abhängig von der Art, Zusammensetzung und Abfolge der Ereignisse, und somit vom Strahl– und Kammmerverhalten.

Der Trigger sollte aufgrund des zellperiodischen Aufbaus der CJC auch zellperiodisch sein. Ein weiterer Vorteil der Zellperiodizität ist die Einsetzbarkeit des Triggers im **F**ull–**S**ize–**P**rototyp (FSP), einem maßstabsgetreuen Modell der inneren CJC von 3 Zellen. Eine andere Bedingung an den Trigger ist, daß die Triggerbereiche veränderbar sein sollen. Die logische Funktion des Triggers darf nicht fest sein, sondern muß in seiner Verknüpfung veränderbar sein.

[8] $\approx 300/(2 * \pi)$ für die im folgenden beschriebenen Trigger

3 CJC Triggerkonzepte und deren mögliche Realisierung

Bei den im folgenden diskutierten Triggerkonzepten handelt es sich um verschiedene Ansätze, wobei das dritte Konzept die Weiterentwicklung des ersten Ansatzes ist.

Der erste Ansatz benutzt zur Mustererkennung (Patternrecognition) die Kreuzung von Spuren mit der Signaldrahtebene. Diese Kreuzungsvektoren werden weiter zu Triggerstraßen verknüpft. Der zweite Ansatz projiziert die Kammergeometrie der H1–CJC auf eine radialsymmetrische Kammergeometrie, wie sie z.B. beim Detektor CELLO vorzufinden war, teilt diese in elementare Zellen auf und verknüpft diese zu Triggerstraßen. Der letzte Ansatz hingegen ist unabhängig von der Kammergeometrie, da er eine universelle Triggerstraßendefinition benutzt. Dieses Triggerkonzept sieht eine weitere Logik vor, die es ermöglicht, nach weiteren physikalischen Größen zu triggern, wie z.B. der Multiplizität eines Ereignisses. Es benutzt jedoch die Kreuzung von Spuren mit der Signaldrahtebene um den ersten Triggerzeitpunkt einer festen Zeit nach dem Ereigniszeitpunkt zuzuordnen.

3.1 Ansatz 1: Ein Trigger, der auf der Verknüpfung von Durchgängen durch die Signaldrahtebene beruht

3.1.1 Die Triggerlogik

Bei diesem Trigger geht man davon aus, daß Spuren, die eine Signaldrahtebene schneiden, ein bestimmtes Bitmuster in Schieberegistern benachbarter Drähte erzeugen, das man zu erkennen versucht, siehe Abb. 14.

Charakteristisch für diese Spurdurchgänge ist das Entstehen von Pfeilspitzen, d.h. die symmetrische Drift der Elektronen von beiden Seiten auf den Signaldraht zu. Dieses Zeitverhalten der Spuren bildet sich in mit Bunchcrossingfrequenz getakteten Schieberegistern ab und liefert dort Muster wie in Abb. 15 .

In einer ersten Stufe wird die Entscheidung gefällt, ob ein Spurdurchgang an einem Draht gemessen wurde, indem man versucht, eines der Muster zu erkennen. Die Muster entstehen durch die Bedingung, daß sowohl das erste Bit des Schieberegisters der betrachteten Lage (höchst signifikant) und eines der ersten drei Bits der Schieberegister der nächsten Nachbarn und ein Bit der Bits 2,3 oder 4 der übernächsten Nachbardrähte gesetzt sein muß. Diese erste Stufe der Spurerkennung liefert Kreuzungsvektoren, d.h. die Spur kreuzt die

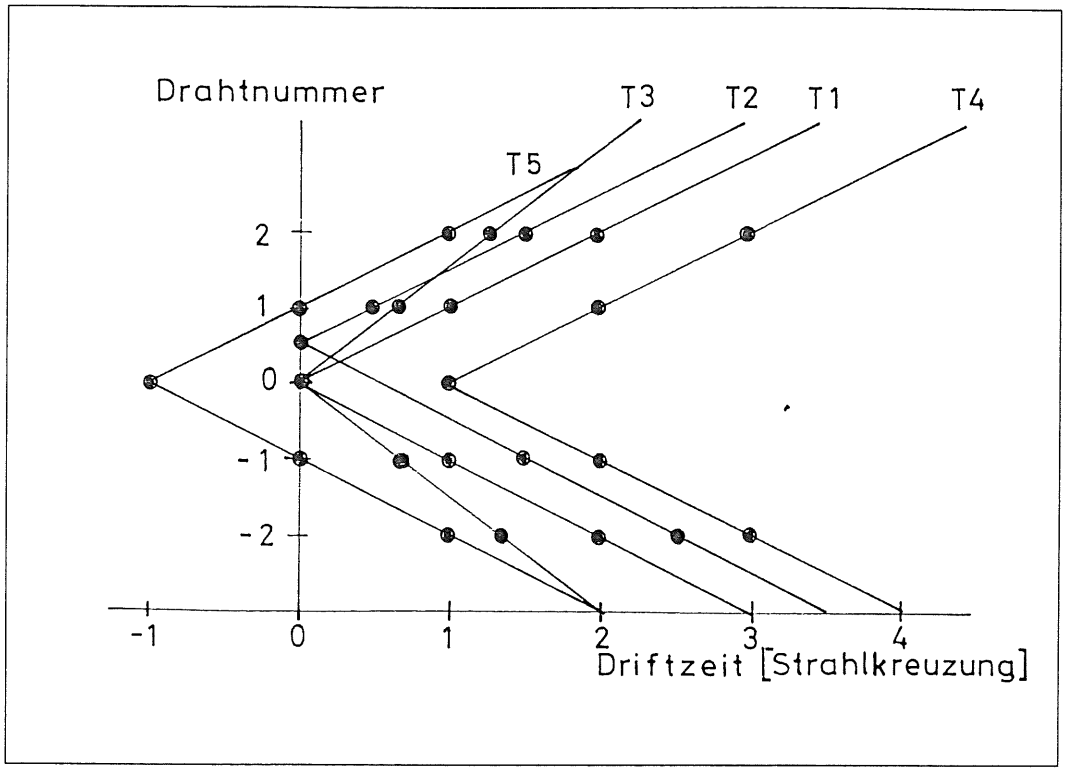

Abbildung 14: Drahtlage gegen Zeit einer Spur, die die Signaldrahtebene schneidet

Meßdrahtebene in einem Punkt und die Spur kommt aus einem eingeschränkten Winkelbereich, der durch die verschiedenen Muster vorgegeben ist, siehe Abb. 16.

In einer zweiten Stufe werden diese Informationen wiederum so miteinander verknüpft, daß man entscheiden kann, ob eine Spur aus dem Wechselwirkungspunkt stammt oder nicht. Dies geschieht, indem man zu einem Durchgang der Spur durch eine Signaldrahtebene der inneren Kammer alle möglichen Durchgänge durch die äußere Kammer, innerhalb eines Impulsbereiches nimmt und vergleicht, ob die Spur an einer dieser Stellen gefunden wird. Wenn dies der Fall ist, ist die Triggerentscheidung wahr, im anderen Fall falsch. Der dabei entstehende Impulsbereich entspricht einem Impuls– Orts–Bereich[9] des dritten Ansatzes. Dies war auch der Ausgangspunkt für den dritten Triggeransatz. Das Vertexauflösungsvermögen dieses Triggers ist ca 5cm, siehe [5], der minimale Transversalimpuls liegt bei 750 MeV/c.

[9]"Football"

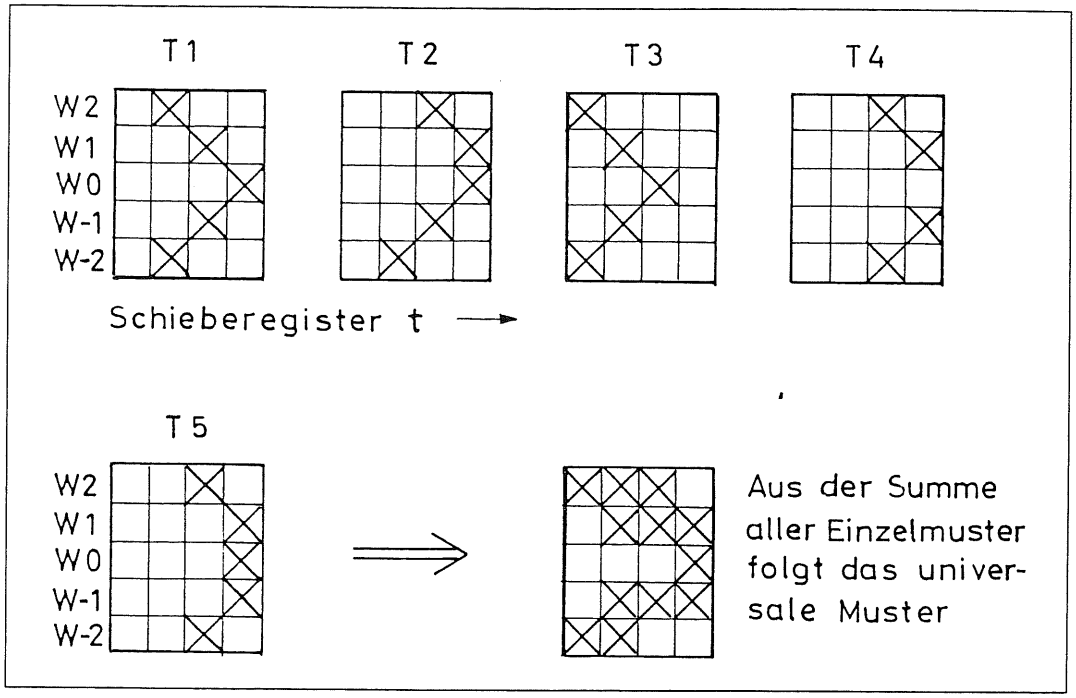

Abbildung 15: Inhalt benachbarter Schieberegister bei den Spurkreuzungen mit der Signaldrahtebene

3.1.2 Der Aufbau

Die Signale der beiden Enden eines Signaldrahtes werden auf einen linearen Verstärker gegeben und addiert. Aus diesem Signal wird nun über einen Schwellendiskriminator und einen Synchronisator ein synchroner Einheitsimpuls geformt.

Das nächste Bauelement ist ein Schieberegister mit seriellem Eingang und parallelem Ausgang, das mit dem Systemtakt (Bunchclock) – ebenso wie der Synchronisator – betrieben wird. Am Parallelausgang dieses Schieberegisters liegt die Zeitinformation des Kammerdrahtes. Diese Schaltung wird an jeden Signaldraht angeschlossen. Um eine Entscheidung über einen Spurdurchgang zu erhalten, benötigt man eine Schieberegistertiefe von 5 Takteinheiten, wobei die größte Signifikanz im ersten Bit liegt.

Um eine erste Datenreduktion zu erhalten, werden nun die Bits 3 bis 5 verodert, da sie nur noch eine Information über den Winkel zwischen Spur und Signaldrahtebene enthalten. Hiernach erhält man nun 3 Bit Information, die den Eingang für eine Entscheidungslogik darstellen. Solche booleschen Funktionen realisiert man in RAM–Bausteinen, indem die vollständige Wertetabelle

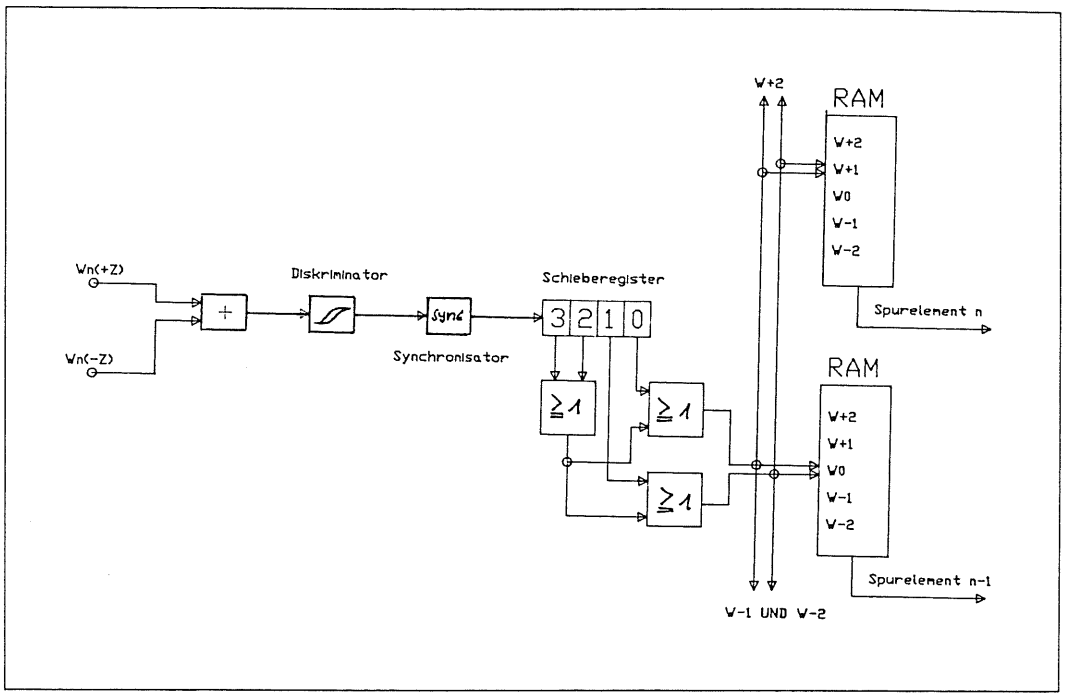

Abbildung 16: Beschaltung benachbarter Register

der Funktion in diesem RAM abspeichert wird. Auf die Adressleitungen wird der Eingangsvektor (das Argument) gelegt. Auf den Datenleitungen liegt nach der bauteilbedingten Verzögerung der zugehörige Ausgangsvektor (Funktionswert des Arguments) an. An einem dieser Bausteine muß man, um eine Entscheidung treffen zu können, die Information von fünf benachbarten Drähten anlegen, siehe Abb. 15.

Das Ergebnis dieser Entscheidung ist, ob eine Spur die Signaldrahtebene an diesem Punkt gekreuzt hat oder nicht. Um flächendeckend arbeiten zu können, müssen von diesen Einheiten 30 * 20 + 60 * 28 = 2280 (innere + äußere Kammer) realisiert werden. [10]

Diese Ergebnisse werden einer zweiten RAM–Logik als Eingangsvektor zur Verfügung gestellt. Verknüpft werden hierbei ein Signal der inneren Kammer mit den entsprechenden der äußeren, die zwar zellperiodisch, aber abhängig von der Lage (Layer) des Signaldrahtes sind. Von dieser Einheit benötigt der Trigger 30 * (24 - 4) = 600 Einheiten, siehe Abb. 17. Das Ergebnis dieser Einheiten ist die, in den so definierten Triggerstraßen, gemessene Spur. Eine

[10]Das sind alle Meßdrähte der beiden Kammern, außer den beiden Randdrähten jeder Zelle. Zeichnet man die Triggerstraßen des 3. Meßdrahtes und des drittletzten einer Zelle, sieht man, daß sich diese teilweise überschneiden

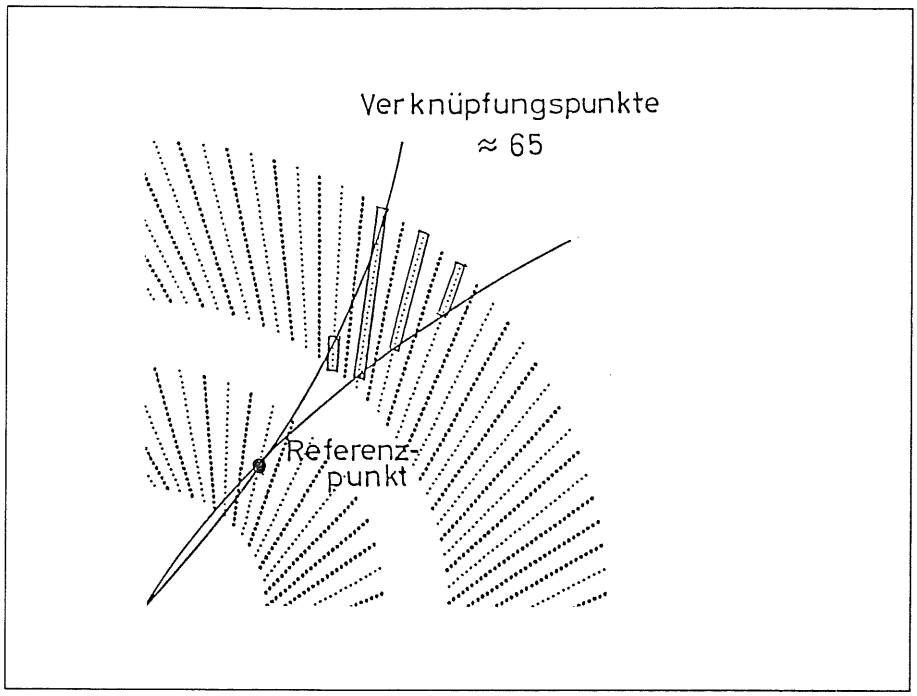

Abbildung 17: Verknüpfung von inneren und äußeren Spurdurchgängen mit Referenzpunkten in der inneren Driftkammer

andere Möglichkeit ist, die Bezugspunkte in die äußere Jetkammer zu legen, siehe Abb. 18.

In diesem Falle bräuchte man 60 * (32 - 4) = 1680 Einheiten, was etwa den 3–fachen Aufwand bedeuten würde, siehe Abb. 18. Der Vorteil ist jedoch die geringere Anzahl von Signalen die miteinander verknüpft werden müssen. Im ersten Fall sind dies etwa 65, im zweiten Fall sind es etwa 25. Beide Anzahlen sind jedoch zu groß um in einem RAM–Baustein realisiert zu werden, da diese nur mit bis zu 20 Adresseingängen erhältlich sind [11]

3.1.3 Beurteilung dieses Triggeransatzes

Dieser Trigger stellt den Versuch dar, auf einer möglichst frühen Ebene eine größtmögliche Information über eine Spur zu gewinnen. Dies soll dadurch erzielt werden, daß die ersten Informationen über die Spur nicht nur Punkte, sondern Vektoren sind. Um diese Information zu erhalten, ist es jedoch notwendig, daß

[11] Währed der Suche nach logischen Bauelementen die über eine größere Anzahl von Eingangssignalen verfügen, kamen die XILINX–Logic–Cell–Arrays auf den Markt, siehe Anhang B. Diese lösten diese Probleme

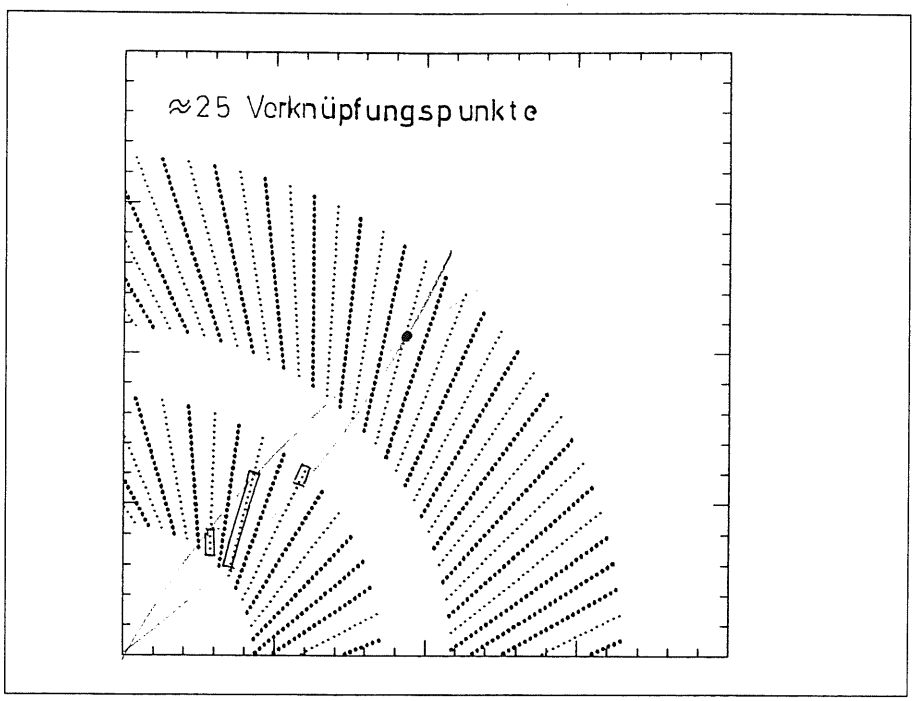

Abbildung 18: Anordnung der Referenzpunkte in der äußeren Driftkammer

die Spur die Meßdrahtebene schneidet.

Für eine Rekonstruktion einer Spur benötigt man drei Punkte, dadurch ist ein Kreis definiert. Da die ersten Informationen jedoch Vektoren sind, reichen zwei dieser Vektoren aus, um eine Spur zu finden. Bei einer solch geringen Anzahl von Meßvektoren ist die Vertexauflösung deutlich schlechter als bei den im folgenden beschriebenen Triggern, die die Spur an 7 bzw. 6 Punkten definieren[12].

Typische Spuren haben $\approx$ 2 auffindbare Schnittpunkte mit der Meßdrahtebene. Das Vertexauflösungsvermögen dieses Triggers liegt bei etwa 5cm [5]. Spuren, die die gleiche Krümmungsrichtung wie die Meßdrahtebene haben, liefern nur wenige Kreuzungspunkte (≈ 1), so daß die Effektivität dieses Triggers für diese Spuren gering ist.

Das Konzept die Spurkreuzungen zu benutzen, um ein exaktes zeitliches Verhalten zu erzielen, wird im dritten Triggeransatz wieder aufgenommen. Es wird dabei jedoch berücksichtigt werden, daß das erste Bit das signifikanteste Bit ist, so daß die Informationen der Nachbardrähte nicht benötigt werden.

[12]Definieren bedeutet nicht messen, sondern die Algorithmen benutzen 7 Punkte unterschiedlicher Radien um die Spur zu rekonstruieren

Der folgende Triggeransatz wurde parallel zu den anderen beiden entwickelt. Er verwendet auch, in der jetzt vorliegenden Version, Triggerstraßen, die mit den Triggerstraßen der anderen beiden Ansätze vergleichbar sind, jedoch projiziert er erst die Kammergeometrie und erzeugt größere Bereiche in denen die Spur gemessen wird.

3.2 Ansatz 2: Ein Trigger, der die Kammergeometrie auf eine radialsymmetrische Geometrie projiziert und darauf einen Baumsuchalgorithmus anwendet

3.2.1 Das Triggerkonzept

Im ersten Schritt wird die Geometrie der Kammer auf eine zentralsymmetrische Geometrie projiziert [7]. Diese Projektion teilt die Kammer in 2,4° große Triggersektoren auf, die an einer Zellgrenze beginnen, siehe Abb. 19. Bei einer Driftgeschwindigkeit von 4,5mm/ 100ns liegen 3 bis 6 Driftstrecken in den Lagen des Triggersektors.

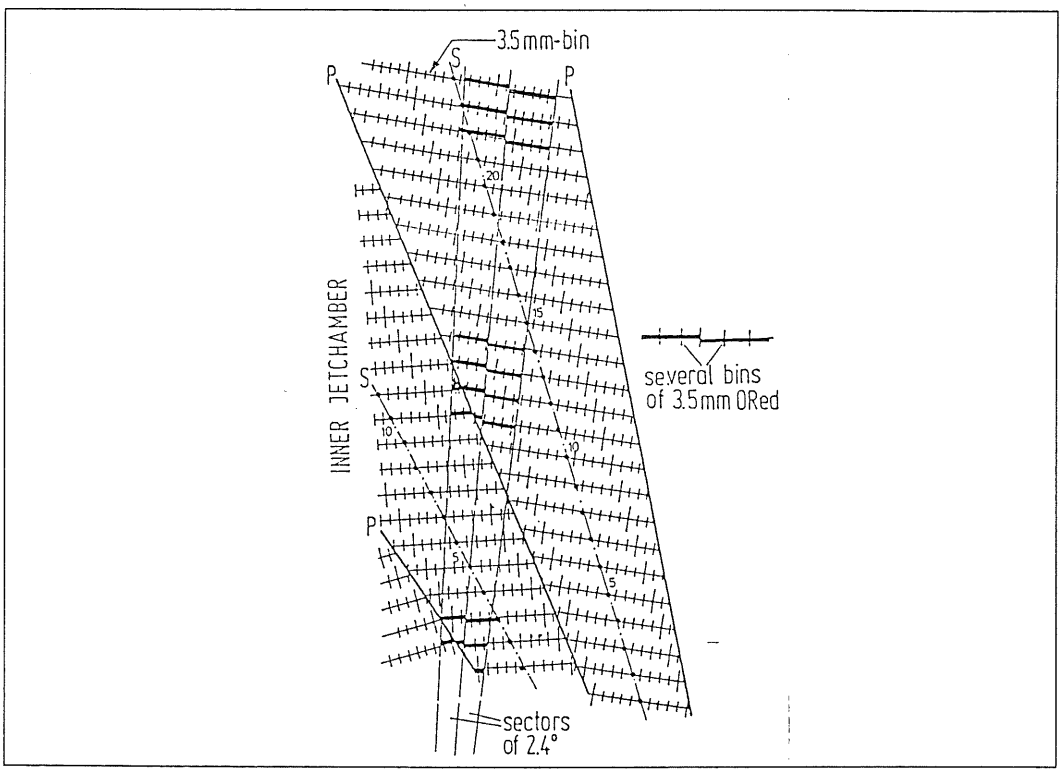

Abbildung 19: Aufteilung der CJC in 2,4° Segmente

Diese 3 bis 6 Bits entscheiden, ob eine Spur in dieser Triggerzellage liegt. Um Spuren, die über eine Sektorgrenze hinausgehen, rekonstruieren zu können, werden diese Triggersektoren zur Hälfte mit den Nachbarsektoren überlappt. Somit erhält man $(360/1,2) * 56 = 16800$ Triggerzellagen.Davon werden jeweils 8 benachbarte Triggerzellagen in einer 6 aus 8 Logik zusammengefaßt, sie stellen die elementaren Zellen dar, aus denen die Triggerstraßen gebildet werden.

Dadurch erhält man in der inneren Kammer 3 Triggerbereiche und in der äußeren Kammer 4. Die 6 aus 8 Logik hat gegenüber einer reinen 'UND–Logik' den Vorteil, daß sie unempfindlicher gegenüber Kammer– oder Auslesefehlern ist. Von diesen Triggerzellen existieren 16800/8=2100. Die technische Realisation dieser Logik ist in Abb. 20 dargestellt. In der nächsten Stufe werden diese 2100 Triggerzellen zu Triggerstraßen verknüpft. Dazu wurde von H.J. Behrend eine umfangreiche Studie angefertigt. Das Auflösungsvermögen dieses Triggers wurde in einer Simulation ermittelt.

Der Trigger wird in 300 Triggerstraßen aufgeteilt, aus deren Signalen die Triggerentscheidung gefällt wird. Die Triggerstraßen werden nach einem Baumsuchalgorithmus zusammengesetzt, der wahlweise nur den inneren Ring bzw. beide Ringe berücksichtigt. Nach neueren Vorschlägen wird diese Logik auch in XILINX–Gatearrays realisiert und der Algorithmus entspricht dem im folgenden beschriebenen Football–Algorithmus, wobei die Referenzpunkte auf einem Kreis um den Vertex angeordnet sind.

Das Vertexauflösungsvermögen wurde mit 1.5 cm angegeben, der minimale Grenzimpuls, der durch die Eingangszellen und den Öffnungswinkel der Suchbäume gegeben ist, wurde mit 350 MeV/c angegeben.

3.2.2 Der Aufbau

Die Signaldrahtinformation wird wieder, wie auch bei dem Trigger von Ansatz 1, verstärkt, linear addiert und synchronisiert. Die am Ausgang des Synchronisators anliegende Information wird in den seriellen Eingang eines 11–Bit tiefen Schieberegisters mit parallelem Ausgang gegeben. Der Ausgang des Schieberegisters wird mit einem Statusregister durch den logischen UND–Operator verknüpft, siehe Abb. 20.

Der Inhalt des Statusregisters entscheidet, ob eines der Zeitbits des Schieberegisters mit zu der betrachteten $2{,}4^o$ Triggerzelle gehört. Dies ist der Fall bei einer logischen 1 als Statusbit. Die so entstandenen 11 Signale werden logisch verodert. Das Ergebnis nach dieser Stufe gibt an, ob ein Spurdurchgang in der Triggerzellage gemessen wurde oder nicht. Der Synchronisator, das Schieberegister, das Statusregister und die Logik sollen in einem selbst entwickelten integrierten Schaltkreis (Customergatearray) zusammengefaßt sein. Von diesen Triggerzellagen sollen 8 in einem Gehäuse realisiert sein. Der Trigger benötigt 16800/8=2100 dieser Gatearrays .

In der nächsten Stufe werden 8 benachbarte Lagen einer Zelle in einer '6 aus 8 Logik' verknüpft. Diese Logik enthält einen RAM–Baustein, der die Wahrheitstabelle der '6 aus 8 Logik' enthält. Dazu werden die 8 Ausgänge eines

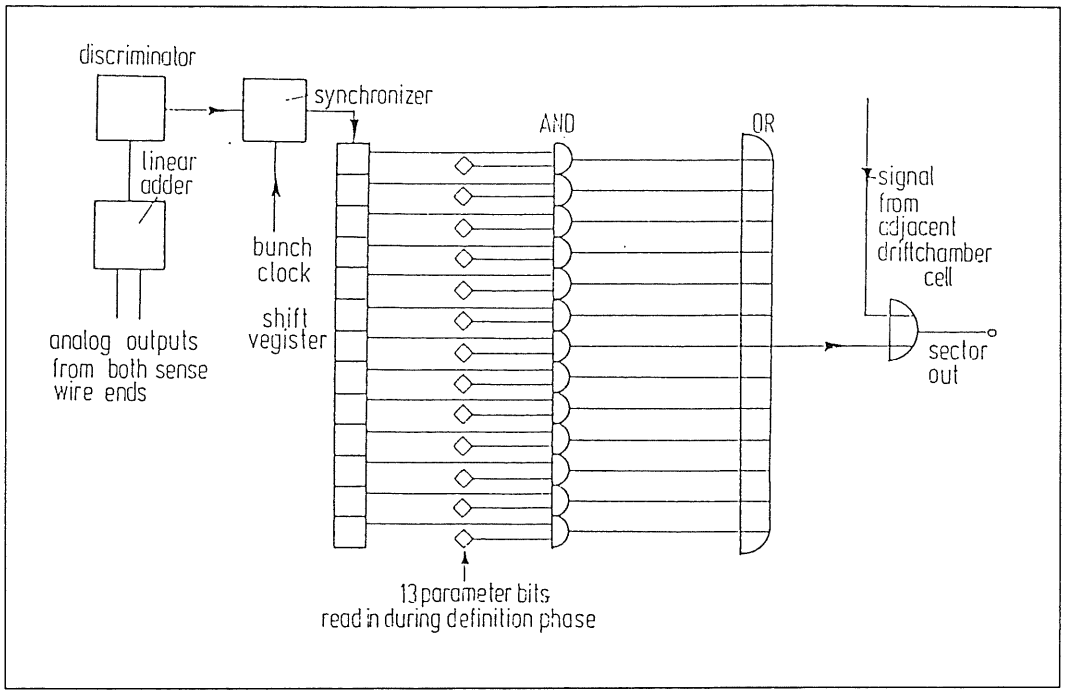

Abbildung 20: Realisierung der 2,4° Segmente in Hardware

Gatearrays an die Adressleitungen des RAM–Bausteins gelegt. Die Datenleitung des RAM–Bausteins ist der Ausgang. Davon sind, ebenso wie von den Gatearrays, 2100 vonnöten.

Die folgende Logik besteht aus 3600 RAM–Bausteinen, die in Gruppen zu je 12 Bausteinen organisiert ist. Diese 12 Bausteine beinhalten eine Triggerstraße. Diese Stufe wird in der letzten Version in XILINX–Logic–Cell–Arrays realisiert.

3.2.3 Beurteilung dieses Triggeransatzes

Dieser Trigger versucht eine möglichst hohe Vertexauflösung und eine möglichst hohe Redundanz zu erreichen. Die dadurch entstehenden Probleme sind folgende:

1. Der minimale Transversalimpuls von 400 MeV/c, der durch die 2,4° Segmente zustande kommt. Eine Spur ist innerhalb eines Sektors sowohl in radialer Richtung als auch in Φ–Richtung nur ungenau lokalisiert. Während ein Schnittvektor des ersten Triggeransatzes genau einem Punkt zuzuordnen ist, kann die Spur bei diesem Trigger irgendwo innerhalb eines Sektors sein. Daraus resultiert eine Radialauflösung von 1.5 cm.

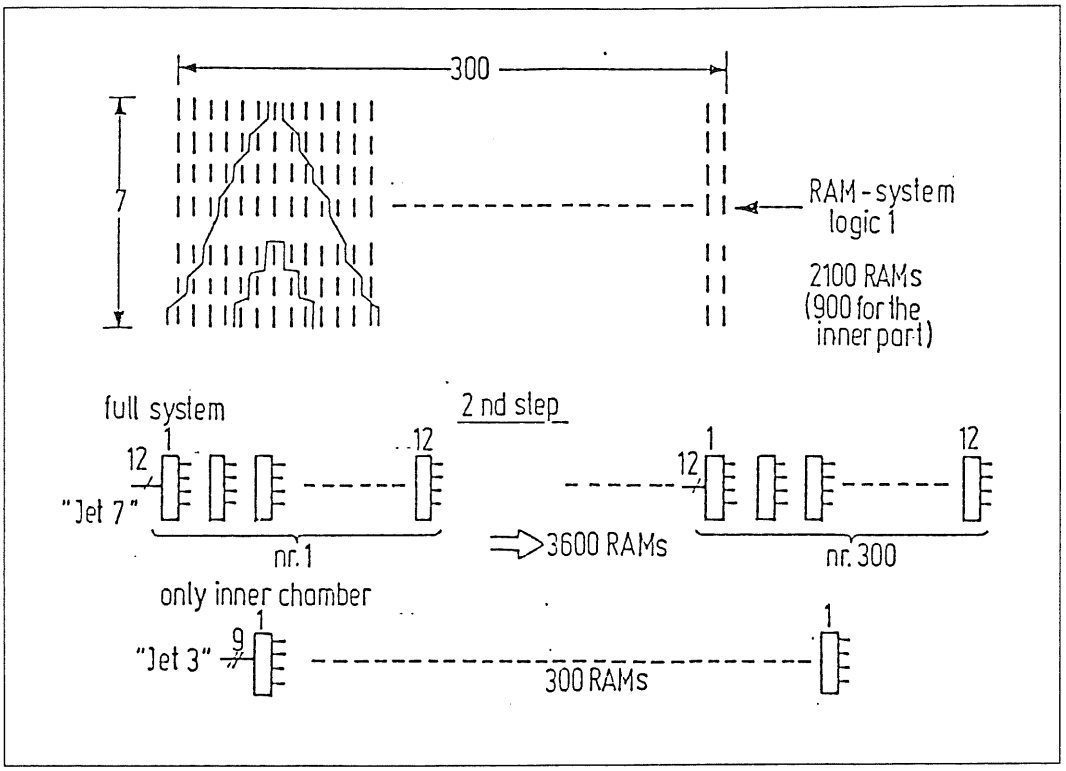

Abbildung 21: Kombination der Segmente zu Triggerstraßen

2. So, wie der Aufbau jetzt realisiert ist, steht der Algorithmus fest, der Trigger ist nicht programmierbar.

3. Ein weiteres Problem dieses Triggers ist die Zeitauflösung. Dadurch, daß in der gegenwärtigen Version die Referenzpunkte, deren Definition im nächsten Abschnitt folgt, nicht das erste Bit eines Schieberegisters benutzen, sondern die in einem bestimmten Radius aufeinanderfolgenden Bits, verliert dieser Trigger die zeitliche Relation zum Bunchchrossing. Dies gilt nicht für eine einzelne Spur, sondern für Jets. Das Triggerbit kann hier ± 3 HERA Taktzyklen vor oder nach dem Triggersollzeitpunkt kommen. Dies ist bei dem im folgenden beschriebenen Trigger nicht so.

Die Folge dieser ersten Stufe, die aus 8 Gatearrays und einem RAM pro Sektor besteht, ist, daß die Genauigkeit der Informationen verloren geht. Jedes Bit stellt einen Bereich (Sektor) dar und nicht mehr einen Punkt (Bin).

Der von mir entwickelte Trigger, welcher im folgenden Abschnitt beschrieben wird, rechnet direkt auf den Bits der Schieberegister und macht keine Koordinatentransformation, d.h. die Triggerstraßen werden direkt aus den Driftzeiten

(Bits der Schieberegister) erzeugt. Er erzeugt das erste Triggerbit genau 17 HERA Taktzyklen nach dem Bunchcrossing, nie früher, und kann in den darauffolgenden beiden Taktzyklen noch zusätzliche Triggerbits produzieren.

3.3 Ansatz 3: Ein Trigger, der den Footballalgorithmus benutzt

3.3.1 Die Triggerstraßendefinition

Bei diesem Trigger handelt es sich ebenfalls, wie auch im vorhergehenden Ansatz, um einen Spurtrigger. Jedoch benutzt dieser Trigger eine spezielle Definition von Triggerstraßen, die dadurch entsteht, daß Spuren, die durch den Wechselwirkungspunkt und einen Signaldraht, den Bezugsdraht, gehen, also die Signaldrahtebene kreuzen, die Triggerstraßen aufspannen, siehe Abb. 22.

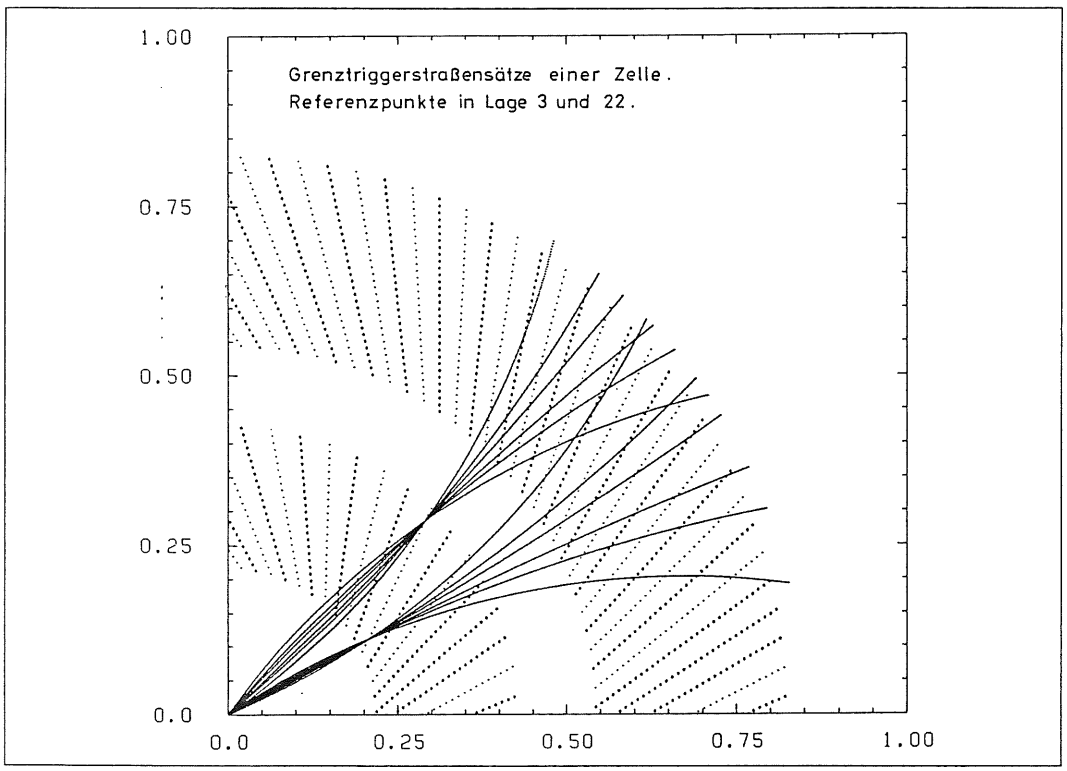

Abbildung 22: Triggerstraßen des Triggers

Diese Triggerstraßen stellen Zweierimpulsbereiche dar. Um festzustellen, ob eine Spur in einer dieser Triggerstraßen liegt, hat man die Triggerbedingung, daß die Spur die Signaldrahtebene in einem Bereich um den Bezugssignaldraht schneidet und die Zeitinformation in den anderen Signaldrähten der Meßlagen[13] dieser Triggerstraße richtig ist.

[13]siehe Abb. 23

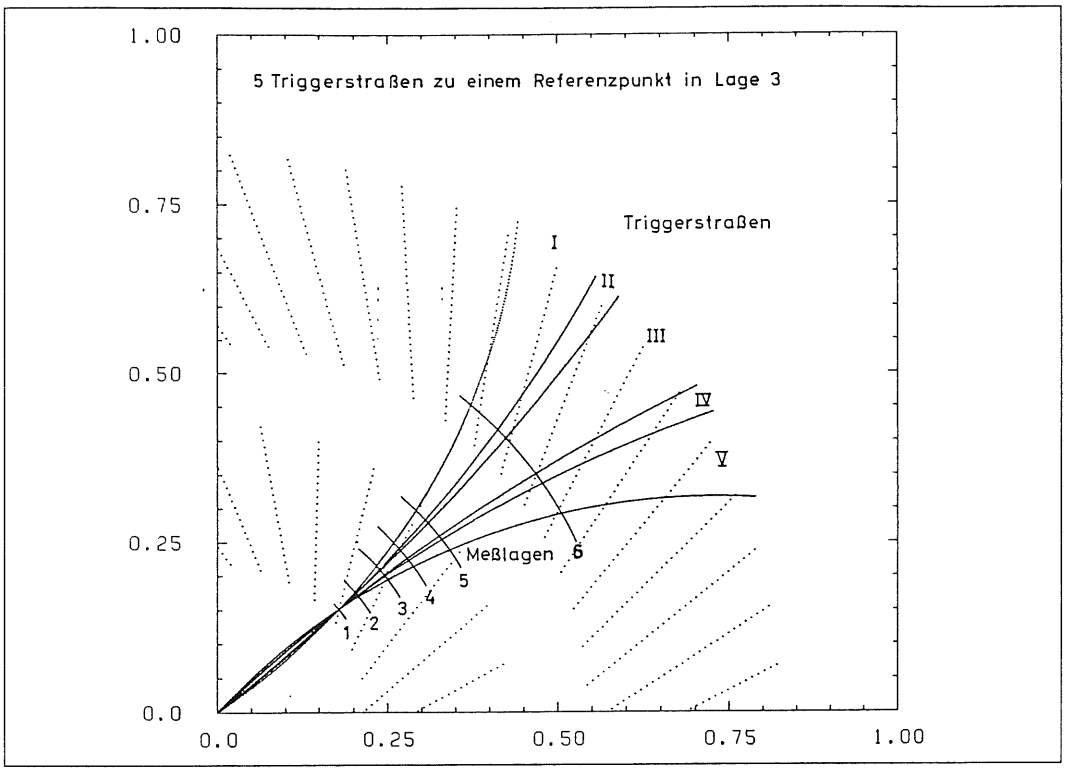

Abbildung 23: Meßlagen des Triggers

Dieser Algorithmus erhielt seinen Namen aus der Form der fünf Impulsbereiche, die zu einem Referenz– oder Bezugsdraht gehören.

Die Referenzpunkte müssen so gewählt werden, daß sie sowohl den Ortsraum als auch den Impulsraum zwischen p_{min} und p_{max} komplett überdecken. Um den Ortsraum vollständig zu überstreichen, gibt es bei diesem Trigger in Kombination mit dieser Kammer zwei Möglichkeiten. Die eine wählt als Referenzpunkte die Meßdrähte einer Meßdrahtebene, die andere Möglichkeit wählt als Referenzpunkte die Zeitbins einer Lage, siehe Abb. 24.

Beide Varianten sind in diesem Trigger programmierbar. Der Vorteil der Variante, die die Kreuzung der Spur mit der Meßdrahtebene als Referenzpunkte benutzt, ist, daß die Zeitrelation zwischen Bunchcrossing und Triggersignal erhalten bleibt. Der Vorteil der zweiten Variante ist, daß sie einfacher zu programmieren ist, da alle Logikbausteine denselben Algorithmus beinhalten.

Da es zur Zeit der Entwicklung dieses Triggers nicht möglich war, die Zeitinformation von allen Lagen zu verarbeiten, weil der Aufwand für den Trigger nur begrenzt sein sollte, wurden 6 Lagen als Messlagen benutzt, siehe Abb.23, von

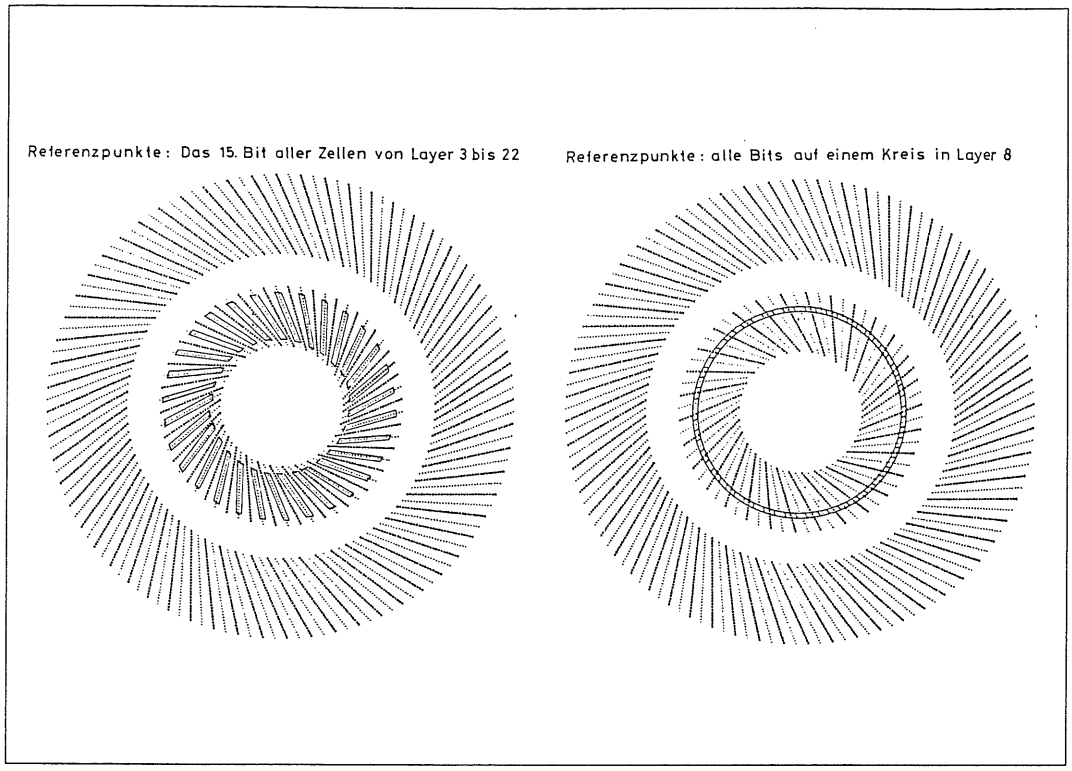

Abbildung 24: Die beiden Möglichkeiten, die Referenzpunkte anzuordnen, links Variante 1, rechts 2

denen mindestens 5 eine wahre Triggerteilentscheidung liefern müssen , um als Ereignis gewertet zu werden. Diese 5 aus 6 Logik soll den Trigger gegen Kammerfehler und zufällige Fehler unempfindlich machen.

Seit Anfang 1989 stehen auch größere Chips, d.h. Chips mit mehr Eingängen und erweiterten logischen Möglichkeiten, zur Verfügung, so daß ein Trigger möglich wäre, der alle Lagen zur Definition der Triggerstraßen benutzen könnte.

Statt der 5 aus 6 Auswahl ist es auch möglich, zwei benachbarte Drähte vor den Synchronisatoren digital zu verodern. Man erhält dabei Doppellagen. Der Vorteil hierbei ist, daß auch im Falle eines defekten Drahtes noch ein richtiges Signal zur Verfügung steht, so daß dieses Verfahren gerade auch für den FSP in Frage kommt. Dem zweiten Verfahren gebe ich aus zwei Gründen den Vorzug:

1. Die absolute Anzahl der benutzten Drähte ist doppelt so groß und somit die Information sicherer.

2. Man kann auf die 5 aus 6 Logik verzichten oder sie optional benutzten.

Der Nachteil der Veroderung von zwei benachbarten Drähten ist, daß die Vertexauflösung schlechter wird, da der effektive Driftraum in R–Richtung größer wird und dadurch die Meßpunkte der Spur ungenauer werden.

Nachdem die Signale zweier benachbarter Meßdrähte verodert und synchronisiert sind, werden die Informationen dieser Drähte in Schieberegister geschrieben und dort periodisch weitergeschoben. Wegen der großen Öffnung der Impulsortsbereiche (Triggerstraßen) in den äußeren Meßlagen verteilt sich die Lageninformation einer Triggerstraße über mehrere Schieberegister benachbarter Zellen einer Lage. Diese Informationen der 6 Meßlagen, die zu einer Triggerstraße gehören, werden logisch verundet. Um einen Kreis und damit eine Spur zu definieren, benötigt man 3 Punkte.

Wenn sowohl die Zeitinformation in den Meßlagen, als auch das Referenzsignal des Impulsortbereiches[14] ein positives Signal liefern, dann ist eine Spur in dieser Triggerstraße gefunden worden, und die Logik gibt einen positiven Triggerimpuls für diese Triggerstraße.

Zu jedem Referenzpunkt gehören also 5 Triggerbits – Triggerbits sind die Ergebnisse der logischen Funktion – wobei das Triggerbit einer Triggerstraße zugeordnet ist. Bei 30 Zellen zu 10 Doppellagen[15], erhält man also 30 * 10 * 5 = 1500 Triggerbits.

Um die endgültige Triggerentscheidung des Triggers der ersten Stufe zu bekommen, werden diese 1500 Triggerbits in einer anschließenden Logik ausgewertet [16]. Diese Logik ermöglicht, sowohl die gesamten Triggerbits zu verodern, so daß man die triviale Triggerentscheidung, es ist eine Spur gefunden worden, erhält, als auch weitere Informationen von anderen Subdetektortriggern zu benutzen und komplexere Entscheidungen zu treffen. Z.B. kann man auf die Multiplizität eines Ereignisses triggern oder auf geometrische Korrelationen.

Das Vertexauflösungsvermögen dieses Triggers ist $\approx$ 1.2 cm. Der minimale Grenzimpuls, der von den festen äußeren Grenzen der Triggerstraßen herrührt, ist 400 MeV/c.

3.3.2 Der Aufbau

Die an den Signaldrähten entstehende Ladungsinformation wird über Verstärker und einen linearen Addierer – die beiden Enden des Drahtes sind gleichberechtigt – in einen Synchronisator gegeben. Dieser erzeugt, wenn ein Spurdurchgang gemessen wurde, einen synchronen Einheitsimpuls. Diese Ausleseelektro-

[14] Football

[15] Eine Doppellage erhält man durch verodern von zwei benachbarten Meßdrähten

[16] Am Ende will man eine Entscheidung, also ein Bit haben

nik wird an allen Signaldrähten der inneren Kammer und an die Drähte der Lagen der äußeren Kammer, die zur Triggerentscheidung mitbenutzt werden, angeschlossen. Das Einheitssignal geht im Falle der Entscheidungsdrähte an den seriellen Eingang von 16-Bit tiefen Schieberegistern mit parallelem Ausgang. An diesen liegt die Zeitinformation des Signaldrahtes. Von den Signaldrähten der inneren Kammer, von denen keine Zeitinformation benötigt wird, geht das Einheitssignal in ein 16-Bit langes Schieberegister mit seriellem Eingang und seriellem Ausgang (FIFO[17]), siehe Abb. 25.

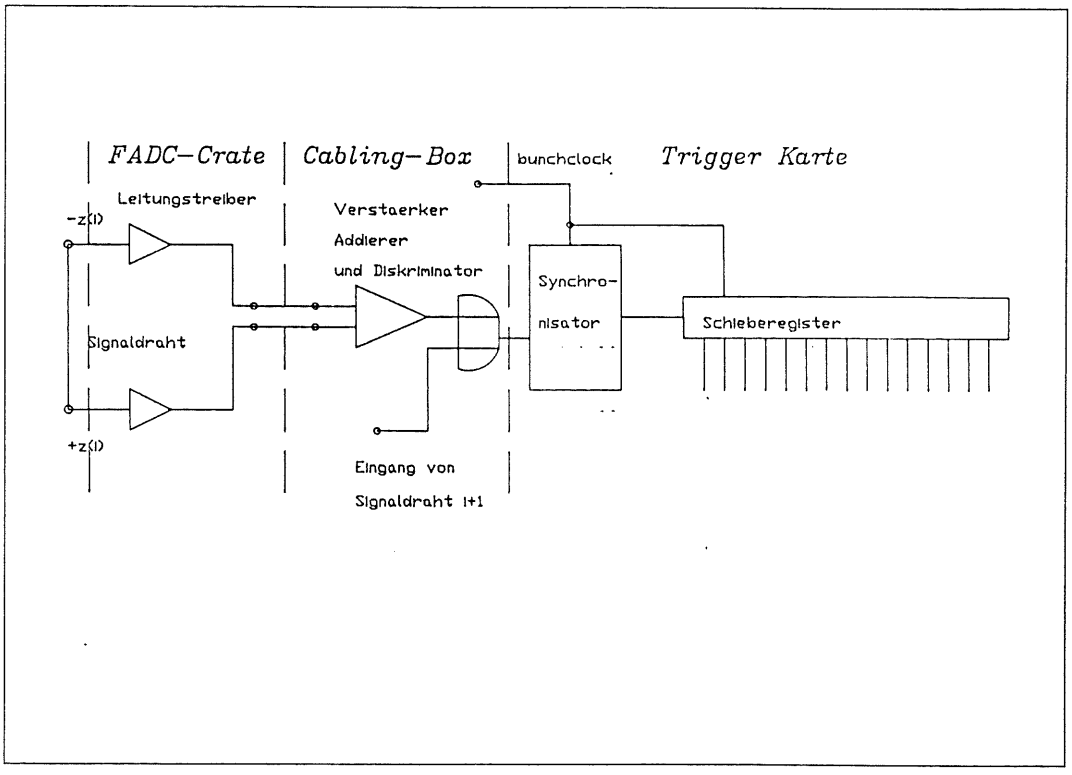

Abbildung 25: Eingangslogik dieses Triggers

Die Triggerentscheidung wird deswegen 16 Taktzyklen nach der Erzeugung der Spur getroffen. Wenn der Ausgang des FIFO'S logisch wahr ist, wird durch eine Logik die Zeitinformation der Schieberegister so miteinander verknüpft, daß am Ende die Triggerentscheidung steht. Dabei wird zuerst die Zeitinformation der einzelnen Lagen durch ein logisches ODER miteinander verknüpft. 6 dieser Signale werden in einer '5 aus 6 Logik' zu einer Ausgangsentschei-

[17]Im Falle eines Nichtmeßlagendrahtes muß das Signal nur um 16 Taktzyklen verzögert werden, da jeder Draht, von 3 bis 22, der inneren Kammer als Referenzdraht dient

dung verknüpft. Dieses Signal sagt aus, ob eine Spur in der zu dieser Logik gehörenden Triggerstraße gefunden wurde. Man benötigt also für jede Triggerstraße eine eigene Logik. Andererseits soll es auch möglich sein, daß man die Triggerstraßen ändern kann. Deswegen werden die Entscheidungslogiken der 5 Triggerstraßen eines Signaldrahtes einer Lage in einer integrierten Schaltung (Chip) zusammengefaßt. An dieser Schaltung liegen alle Zeitinformationen an, die zur Entscheidung notwendig sind. Es wurde ein Programm erstellt, das eine Verbindungsliste erzeugt, d.h. das entscheidet, welche Schieberegister logisch miteinander verbunden werden müssen. Diese können aber variabel miteinander verknüpft werden, so daß man verschiedene Triggerstraßen realisieren kann. Die Triggerstraßenlogiken werden in programmierbaren Chips der Fa. XILINX realisiert, deren logischer Inhalt auf einem IBM-AT Rechner erzeugt und in den Trigger heruntergeladen wird. Um den gesamten Winkelbereich abdecken zu können, benötigt man solche Einheiten in der inneren Kammer von Lage 3 bis 22. Wenn man nun noch im Vorwege 2 benachbarte Signaldrähte logisch zusammenfaßt, kommt man zu 10 * 30 * 5 = 1500 Triggerstraßen, die in 300 Logikbausteinen untergebracht werden.

In einer weiteren Stufe können diese 1500 Triggerbits nach physikalischen Gesichtspunkten analysiert werden. Diese Triggerstufe besteht aus 30 + 3 Logikarrays auf 3 Karten, sie sind über einen 2*30 Bit breiten Triggerbus miteinander verbunden.

3.3.3 Beurteilung dieses Triggeransatzes

Dieser Trigger benutzt eine universelle Triggerstraßendefinition. Diese Triggerstraßen werden direkt aus den Raumbins[18] der CJC errechnet. Die Vertexauflösung dieses Triggers liegt bei $\approx$ 1,6 cm und im gleichen Bereich wie die Auflösung des vorhergehenden Triggers. Ein Nachteil dieses Systems ist der ungewöhnliche Umgang mit den Logikarrays. Die Vorteile dieses Triggerkonzeptes sind:

1. Die Triggerstraßen sind durch Software definierbar, so daß einfach verschiedene Triggerlogiken programmierbar sind. Es ist z.B. möglich, daß als Referenzpunkt die verschiedenen Meßdrähte einer Zelle benutzt werden oder die verschiedenen Bits einer Lage.

2. Das hohe Auflösungsvermögen in Relation zur Einfachheit des Systems. Damit verbindet dieser Trigger die Vorteile der beiden vorhergehenden Ansätze, ohne dabei deren Nachteile zu haben.

[18] Driftlänge pro HERA–Bunchclock

3. Feste Zeitrelation zwischen dem Zeitpunkt des Ereignisses und dem Setzen des Triggerbits.

4. Die einfache Kombination mit anderen Triggern, z.B. einem R–Z–Trigger an der CJC oder dem Z–Vertex–Trigger der Z–Kammern. So kann dieses System, die in Φ selektierte Informationen von Z–Triggern, ohne Erweiterung verarbeiten, da der Aufbau die Signale dieser Trigger als Eingangssignale vorsieht.

3.4 Die Entscheidung für den dritten Triggeransatz

Aufgrund der in diesem Kapitel beschriebenen Vor- und Nachteile habe ich mich für die Realisierung des dritten Triggerkonzepts entschieden. Zum einen ist er sehr kompakt, damit preisgünstiger als die beiden anderen Konkurrenten, zum anderen bietet er die gleichen theoretischen Leistungen wie der Trigger, der die Kammersymmetrie projiziert. Des weiteren liefert er ein eindeutiges zeitliches Verhalten. Der Name dieses Systems soll AllaH sein (**A** large **l**ogic **a**rray for **H**1).

4 Die Simulation des dritten Triggeransatzes

Nachdem in den vorangegangenen Kapiteln verschiedene Triggerkonzepte diskutiert wurden, werden in diesem Kapitel die theoretischen Möglichkeiten und Eigenschaften des dritten Triggeransatzes analysiert. Dies wird durch eine Simulation des Triggers erreicht, die

1. Einzelspuren und
2. Spuren von verschiedenen Ereignisklassen (c–$\bar{c}$, b–$\bar{b}$ Ereignisse und Untergrundereignisse von Strahl– Gas–Wechselwirkungen),

mit dem Großrechner bei DESY erzeugt und deren Verhalten in der Triggerlogik simuliert.

Ziel der Simulation ist die Bestimmung der Effizienz des Triggers für Spuren geladener Teilchen von e–p–Wechselwirkungen und die Akzeptanz des Triggers für Spuren von Untergrundereignissen. Darüberhinaus lassen sich problematische Fälle analysieren und der Triggeralgorithmus optimieren.

4.1 Das maximale theoretische Auflösungvermögen der Trigger

Um das theoretische Vertexauflösungsvermögen des Triggers, d.h. die Breite der Triggerstraßen am Vertex, zu berechnen, ist als erstes die größtmögliche Winkelabweichung zwischen wahrem Ort der Spur und gemessenem Ort zu bestimmen. Die maximale Abweichung erhält man durch Bitversatz. Die Quelle hierfür ist im wesentlichen die Synchronisiereinheit.

Die Vertexauflösung wurde als Funktion der Spurimpulse und der Anzahl der aktiven Meßlagen untersucht. Die Ergebnisse dieser Untersuchungen sind in Abb. 26 und 27 dargestellt. Die Vertexauflösung für einen Trigger mit 6 Meßlagen und einem Referenzpunkt ist $\approx 1.1 \pm 0.1$cm. Diese Größe muß in Relation zur Ausdehnung des Strahls[19]und der lichten Weite des Strahlrohres[20] gesehen werden. Da Beam–Gas–Wechselwirkungen innerhalb des sensitiven Z–Bereiches nicht unterdrückt werden können, ist die Konsequenz, daß es ausreicht, einen Trigger mit 6 Meßlagen zu realisieren.

[19]Der Protonenstrahl hat eine Ausdehnung von ≈cm [2]
[20]Radius ≈ 10 cm [3]

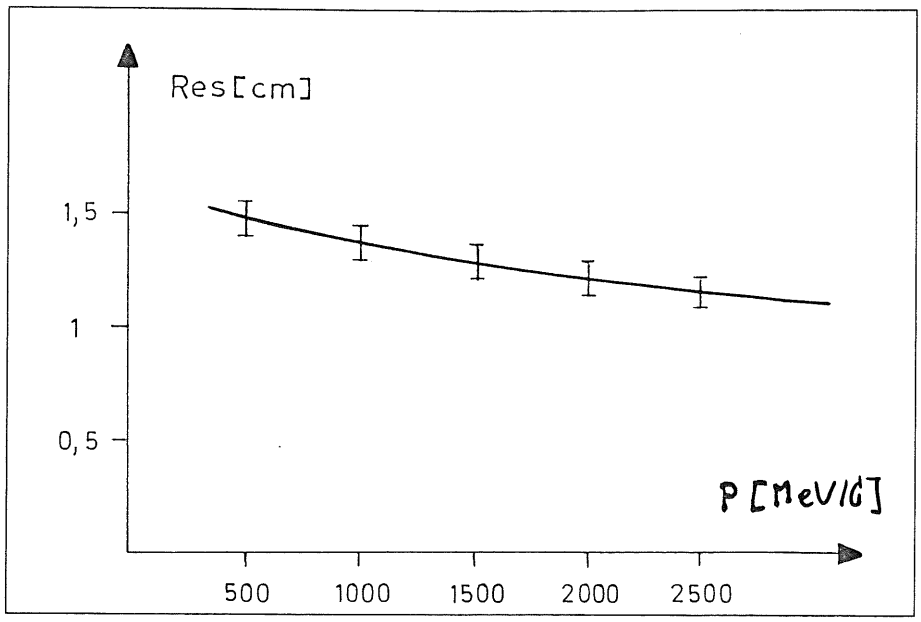

Abbildung 26: Die Vertexauflösung der CJC in Abhängigkeit vom Impuls bei 6 Meßlagen

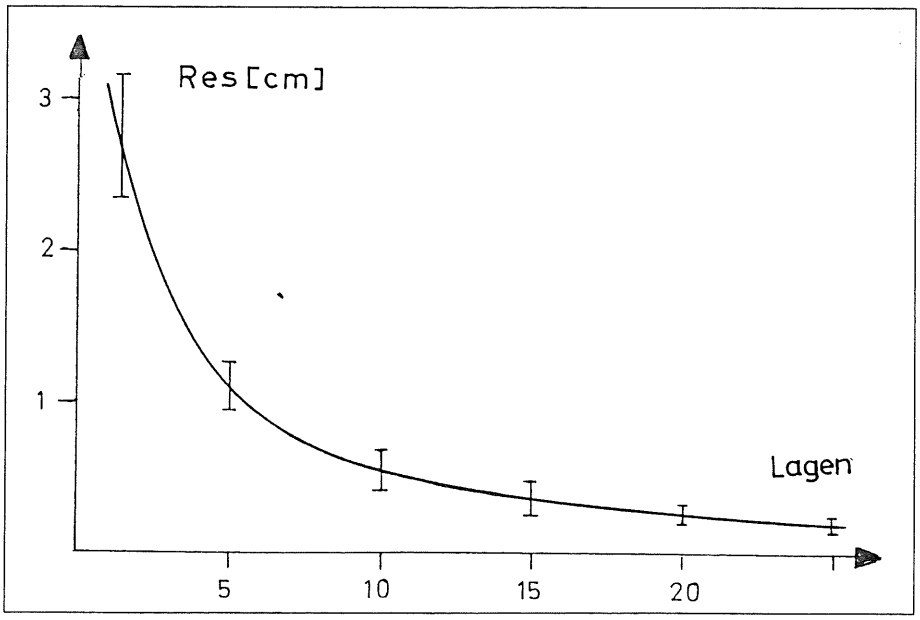

Abbildung 27: Abhängigkeit der Vertexauflösung in Abhängigkeit der Anzahl der Meßlagen bei konstantem Impuls (750 MeV/c)

4.2 Die Triggersimulation

Die Simulation sollte die zum oben, Kapitel 3.3, beschriebenen Algorithmus gehörende Hardware möglichst genau wiedergeben. Die Simulation der Hardware startet deswegen bei den Schieberegistern. Der analoge Teil der Hardware, die Digitalisierung und die Sychronisation dieser Impulse mit der HERA–Bunchclock wurden nicht simuliert.

Um die Lage der Spur in den Schieberegistern zu berechnen, wurde die folgende Näherung benutzt:

$$d = \frac{2 * \pi * R}{v_d * t}$$

Diese Formel beschreibt eine Drift der Sekundärelektronen auf einer Kreisbahn um den Vertex zum Signaldraht hin. Somit ist der Zusammenhang zwischen Abstand der Spur vom Signaldraht in einer Meßlage und der Driftzeit gegeben. Diese Driftzeitintervalle werden in der Hardware durch Bits in Schieberegistern repräsentiert. Die Simulation startet mit der Einlese von Spuren in diese Schieberegister. Auf den Inhalt der Schieberegister wird die Logik angewendet, die dem Inhalt der Gatearrays entspricht.

Die Logik wird sowohl für die Gatearrays des realen Triggers, als auch für die Simulation, d.h. die Berechnung der Masken, von dem gleichen Programm erzeugt. Deswegen lief das Simulationsprogramm nicht auf dem Großrechner bei DESY, sondern auf einem lokalen System, welches auch die Entwicklungsumgebung der XILINX–Chips enthielt.

Des weiteren gibt es die Möglichkeit, das Fehlen oder den Ausfall von Kammerdrähten zu simulieren, in dem die Schieberegister mit einer Maske verundet werden. Diese Maske kann mit einem Zufallszahlengenerator gesetzt werden. Es gibt hier die Möglichkeit, sowohl ein defektes Schieberegister oder einen defekten Draht zu simulieren, als auch zufällige zusätzliche Einträge in die Schieberegister (Hits).

4.2.1 Eigenschaften der vorhandenen Monte–Carlo–Daten

Die Monte–Carlo–Daten, die zur Simulation des Triggers benutzt wurden, lagen auf der IBM als BOS[21]– Datein vor. Dabei waren die echten Physik–Ereignisse im H1–GEANT[22] Format abgespeichert, die Untergrunddaten in einem eigenen Format.

Für die Erzeugung der Daten wurden verschiedene Eventgeneratoren verwendet:

1. Ein Programm von R. Eichler, welches auf Messungen der Impulsspektren von Teilchenschauern bei Experimenten am Speicherring ISR (p–p Ereignisse) basiert, zur Simulation von Untergrundereignissen.

2. FRITIOF, ein Generator für Ereignisse, der nach dem LUND–Modell arbeitet und e–p–Wechselwirkungen simuliert.

Drei physikalische Größen können in Driftkammern gemessen werden, die zur R–Φ–Triggerentscheidung führen.

1. Der transversale Impuls geladener Teilchen.

2. Die Anzahl der geladenen Teilchen eines Ereignisses, die Multiplizität.

3. Der Abstand der Spur vom Vertexpunkt [23].

Die Anzahl der Untergrundereignsisse war 3000, die Anzahl der c–$\bar{c}$ und der b–$\bar{b}$ Ereignisse war jeweils 1000. Die Verteilung der Monte–Carlo–Daten ist in den Bildern 29 und 28 gezeigt. Man sieht, daß etwa 80% der physikalisch interessanten Ereignisse ihren Vertex innerhalb eines Radius von 1.5cm haben. Dagegen haben nur etwa 40% der Untergrundereignisse ihren Vertex in diesem Bereich. Die Multiplizität der Untergrundereignisse ist deutlich geringer als die Multiplizität der interessanten Ereignisse. Die durchschnittliche Multiplizität der Ereignisse war für die Untergrundereignisse etwa 2.0 und der interessanten Ereignisse war etwa 11.

[21] Bank [29]
[22] "GEANT3 ist ein System zur Detektorbeschreibung und ein Simulationswerkzeug [31]
[23] Diese Größe wird manchmal auch "distance of closest approach" genannt

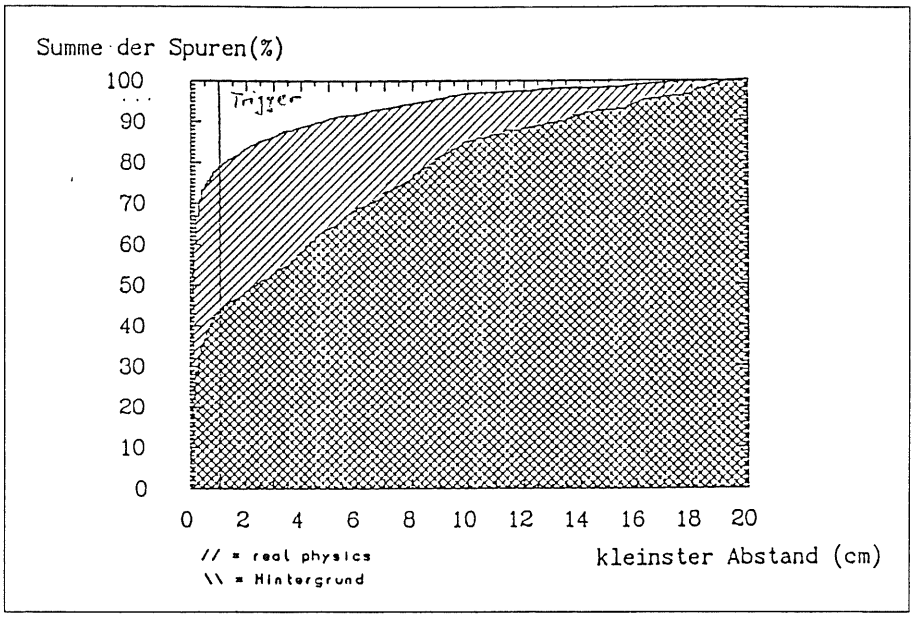

Abbildung 28: Physikalisch interessante Ereignisse und Untergrundereignisse in Bezug auf den Abstand zum Vertex

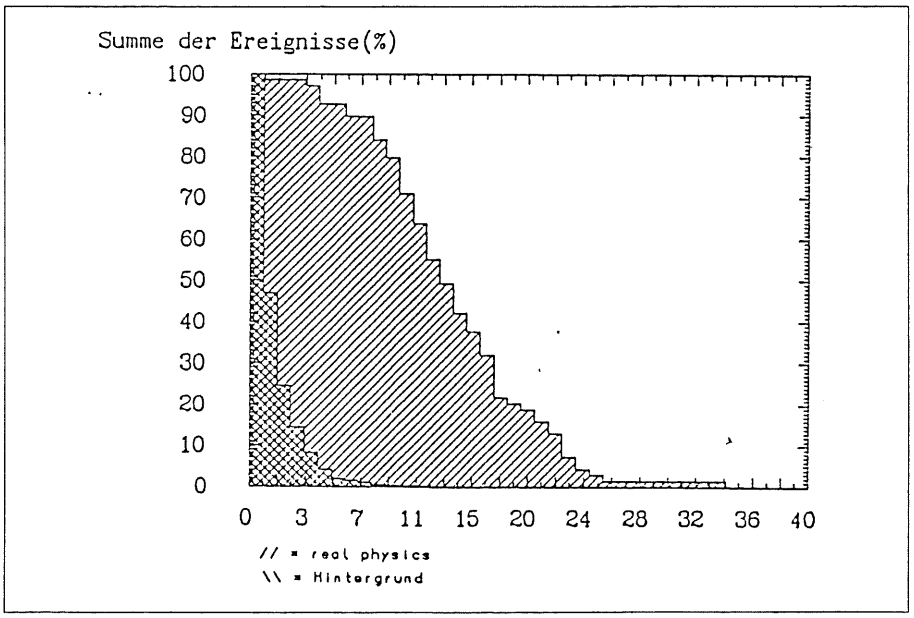

Abbildung 29: Verteilung der Multipliziät für physikalisch interessante Ereignisse und Untergrundereignisse

4.2.2 Simulation des Triggers mit Monte–Carlo–Ereignissen

Die Simulation des Triggers benutzte die im vorangegangenen Kapitel beschriebenen Daten. Dabei soll die Auswertung der Simulation nach folgenden Gesichtspunkten erfolgen:

1. Wieviel Triggerbits erzeugt ein Ereignis? Diese Frage ist insofern von Interesse, weil im allgemeinen die Multiplizität eines Ereignisses ungleich der Anzahl der Triggerbits sein wird.

2. Wie groß ist die Akzeptanz des Untergrundes und wieviel interessante Ereignisse werden unterdrückt?

3. Wann entsteht das erste Triggerbit? Dies ist, wie vorher beschrieben für die Sychronisation von Wichtigkeit.

Die Ereignisse wurden unter der Bedingung, daß eine Spur aus dem Vertex gefunden wurde, getriggert. Dabei war die Multiplizität der getriggerten Spuren im Mittel 6,3. Der Vergleich von Untergrundereignissen mit den c–$\bar{c}$ oder b–$\bar{b}$ ergab folgende Ergebnisse:

1. Die Untergrundereignisse wurden bis auf eine Annahme unterdrückt (von 3000). Die Unterdrückung der interessanten Ereignisse lag bei etwa 30%. Diese hohe Unterdrückung von interessanten Ereignissen resultiert aus dem Fehlen von Spuren in der äußeren Kammer, d.h. das Ereignis war stark vorwärts gerichtet.

2. Der erste Triggerzeitpunkt ist genau 17 Takte nach dem Ereigniszeitpunkt für alle Ereignisse, die getriggert wurden. Die weitere zeitliche Verteilung der folgenden Triggerbits ist in Abb. 30 dargestellt.

3. Die durchschnittliche Anzahl der Triggerbits pro getriggertem Ereignis ist 10,3 $\pm$5,. Sie ist etwa 1.7 mal größer als die durchschnittliche Multiplizität der Ereignisse. Dies war aber auch zu erwarten, denn eine Spur kreuzt durchschnittlich zweimal eine Meßdrahtebene der inneren Jetkammer, so daß zwei Triggerbits erzeugt werden.

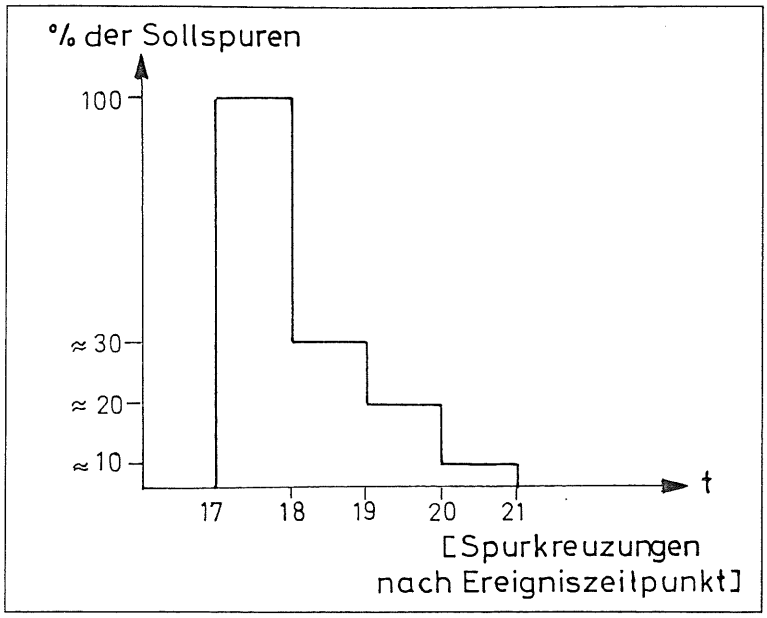

Abbildung 30: Verteilung der Triggerzeitpunkte

Über die tatsächlich bei HERA auftretende Unterdrückung der Ereignisrate ist nur schlecht eine signifikante Aussage zu machen. Dazu bräuchte man genaue Kenntnis über den Strahl, seine zeitliche Änderung und seine Zusammensetzung. Hierzu sind aufwendige Simulationen, nicht nur des Strahlorbits mit Synchrotron– und Betatron–Schwingungen nötig, sondern es müßte auch ein Feedback der einzelnen Bunche simuliert werden, siehe [22].

4.2.3 Die Simulation des Triggers mit speziell generierten Spuren

Um "Problemstellen" des Trigger aufzuspüren, wurde der Trigger mit speziellen einzelnen Spuren getestet. Diese Spuren sind mit wenig Rechenaufwand herzustellen. Die weitere Aufgabe dieses Testsatzes von Spuren ist die Programmierung des Triggers zu testen[24]. Die Simulation liefert ein Maß für die Schärfe der Grenzen der Triggerstraßen. Ein Satz von Beispielergebnissen ist in Abb. 31 dargestellt.

4.3 Zusammenfassung der Simulation

Die Simulation zeigt, daß dieses Triggerkonzept durchaus in der Lage ist, wirksam Untergrund zu unterdrücken. Seine speziellen Vorteile liegen in dem exak-

[24]Es gibt günstige und ungünstige Grenzimpulse der Triggerstraßen

ten Einhalten zeitlicher Relationen und in der genauen Vertexdefinition, sowie den relativ scharfen Kanten der Triggerstraßen.

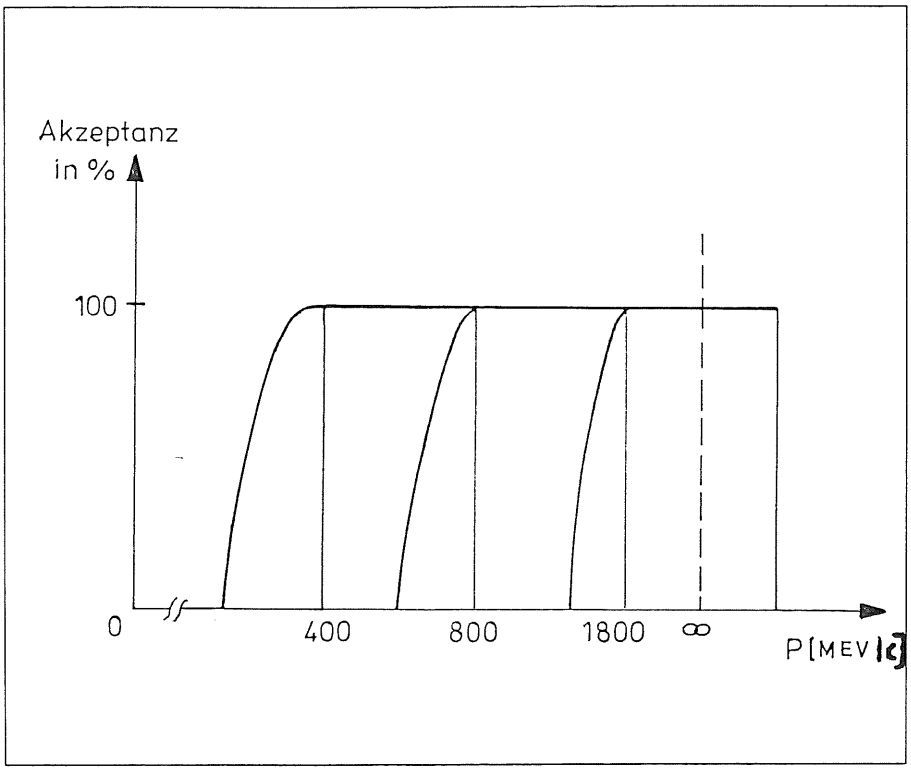

Abbildung 31: Ein Satz von Triggergrenzimpulsen. Aufgetragen wurde der Impuls gegen die Akzeptanz.

5 Kosten für den Trigger

Die Kosten für den oben beschriebenen Trigger sind in Tabelle 2 dargestellt. Sie verteilen sich dabei auf die folgenden Komponenten:

1. die Triggerkarten
2. die Scanner- und Pufferkarten
3. das Crate mit lauffähigem System
4. der IBM-AT
5. die Verkabelung
6. die Software

Den größten Anteil an den Kosten stellen die Triggerkarten dar. Der Stückpreis für eine solche Karte hängt von der Art und Weise der Fertigung ab. Unter der Annahme, daß das Layout der Karten hier bei DESY erstellt wird, liegt der Stückpreis bei ca. 2 KDM. Der gleiche Preis gilt in etwa für die Scanner- und Pufferkarten. Die Gesamtkosten für diesen Trigger liegen somit bei ca. 100 kDM.

Baugruppe	Preis in KDM
30 Triggerkarten	80
Prozessorkarte	10
Festplatte	3
Trigger–Test–Karte	3
2 Crates	10
Summe	$\approx$ 100

Tabelle 2: Kosten des Triggers

6 Über die mögliche Erweiterung des Triggers

Ziel dieser Arbeit war für den FSP[25] einen Trigger bereitzustellen, der auch im H1–Experiment einsetzbar ist. Dieser hier beschriebene Trigger stellt eine abgeschlossene Einheit dar, die nicht erweiterbar ist. Dies gilt nicht für die Scanner und Pufferlogik, da diese universell ist. Soll die Anzahl der Meßpunkte des Triggers vergrößert werden, im wesentlichen in der äußeren CJC, so müssen dafür spezielle Triggerkarten entwickelt werden, die dann in der Scanner- und Pufferkarte mit den Informationen dieses Triggers kombiniert werden können.

Unter heutigen Gesichtspunkten (es sind neue Chips auf dem Markt, die eine wesentlich größere Anzahl von I/O-Pins zur Verfügung haben) könnte dieser Trigger mehr Meßlagen benutzen. Dies würde die Triggerauflösung zwar nicht verbessern, der Trigger wäre jedoch gegen Fehler unempfindlicher. Mehr Meßlagen heißt in diesem Fall, daß ein Trigger möglich ist, der alle Signaldrähte als Eingang benutzt, wobei diese paarweise verodert sind.

Um das Auflösungsvermögen des Triggers zu verbessern, ist es nicht nötig, die Anzahl der Meßlagen zu vergrößern. Das Auflösungsvermögen ist im wesentlichen durch die Bingröße der Driftstrecken pro Bunchclock gegeben.

Um eine weitere Verbesserung des Triggers zu erzielen, d.h eine höhere Unterdrückung des Untergrundes zu erhalten, müssen die vorhandenen Daten genauer oder noch andere Daten dazu genommen werden. Genauere Daten heißt, daß die Schieberegister mit einer höheren Frequenz getaktet werden müssen. Andere Daten dazunehmen heißt, die Triggerbits mit den Triggerbits anderer Trigger zu kombinieren. Dies sieht dieses Triggerkonzept als wesentlichen Teil vor. Eine Möglichkeit ist, die jetzt ebenen Triggerstraßen in den Raum, d.h. längst der Z-Achse auszudehnen, um so nicht nur den Vertex des Ereignisses in der R–Φ–Ebene zu bestimmen, sondern im Raum. Da eine große Anzahl von Untergrundereignissen erwartet wird, die in Z–Richtung eine große Abweichung vom Vertex haben, sollte gerade diese Möglichkeit ausgenutzt werden.

[25] **F**ull- **S**ize-**P**rototype

7 Zusammenfassung

Im Rahmen dieser Arbeit wurden verschiedene Trigger der ersten Stufe (First-Level-Trigger) für die zentrale Driftkammer von H1 untersucht. Ein solcher Trigger soll totzeitfrei alle 96 ns eine Entscheidung über ein Ereignis fällen. Dazu wurden drei verschiedene Konzepte studiert, die jeweils eine Kombination aus Triggeralgorithmus und Hardware darstellen.

Der erste Ansatz basiert auf der Erkennung von Spurdurchgängen durch die Zähldrahtebene. Diese Spurdurchgänge erzeugen spezielle Patterns (Muster) in mit Bunchfrequenz getakteten Schieberegistern. Diese Muster werden in einer RAM-Logik getestet. In einer zweiten Stufe werden die Ergebnisse dieser Spurchdurchgänge zu Triggerstraßen kombiniert. Das Konzept dieses Triggers wurde aufgrund der Probleme bei der Erkennung von Spuren, die in gleicher Richtung wie die Zähldrahtebene gekrümmt sind, verworfen.

Das zweite Konzept projiziert die H1-Kammergeometrie auf eine zentralsymmetrische Kammergeometrie. Dazu werden spezielle zu entwerfende Gatearrays benötigt. Des weiteren wird in einer zweiten Stufe ein Baumsuchalgorithmus auf die transformierten Daten angewendet. Dieses System zeigt zwar brauchbare Ergebnisse, ist aber extrem aufwendig.

Der dritte Ansatz benutzt den sogenannten "Footballalgorithmus" zur Spurensuche. Dieser ist komplett unabhängig von der Kammergeometrie und läßt sich sehr einfach in programmierbaren Gatearrays realisieren. Dieser Trigger liefert die gleichen theoretischen Werte wie der zweite Ansatz, ist dabei aber ungleich einfacher. Deswegen fiel die Entscheidung der Realisierung für den FSP (Full–Size-Prototype) auf den Trigger des dritten Ansatzes.

Dieser Trigger besitzt im schlechtesten Fall ein Vertexauflösungsvermögen von ungefähr 1,5cm, normalerweise von besser als 0,8cm. Die Impulsauflösung ist in fünf Bereichen programmierbar, wobei die Breite der Bereiche gleich der Impulsauflösung ist. Diese Bereiche sind die Triggerstraßen. Die minimalen Grenzimpulse dieses Triggers liegen bei $\pm$ 400MeV/c.

Im Rahmen dieser Arbeit wurden von diesem Trigger für den FSP der H1-CJC drei Triggerkarten und eine Karte zum Test der Triggerkarten hergestellt. Außerdem wurde die Betriebssoftware für den Trigger geschrieben.

Anhänge

A Die Realisierung des Triggersystems

A.1 Die Triggerhardware

Der Trigger wird aus 30 VME[26]–Platinen aufgebaut, wobei jede Platine eine Triggerzelle aufnimmt. Wie oben beschrieben, erzeugen diese 30 Platinen aus 750 (1500) Eingangssignalen 1500 Ausgangssignale, die Triggerbits. Um die 1500 Triggerbits dem VME–Bus und darüber weiteren Triggerstufen zugänglich zu machen, werden drei Karten benötigt, welche die Triggerbits aus den 30 Triggerkarten auslesen und daraus die Gesamttriggerentscheidung fällen.

Weiter werden die Triggerbits analysiert (scannen) um nicht alle Triggerbits übertragen zu müssen. Dazu werden die Bits durchnumeriert, und es wird nur, wenn ein Triggerbit gesetzt ist, die zugehörige Adresse gespeichert und dem Bus zur Verfügung gestellt.

Zur Kommunikation mit dem Subdetektormaster ist eine Kommunikationskarte mit einer seriellen Verbindung vorgesehen. Für den FSP Testaufbau ist hier eine Ethernetkarte zur IBM vorgeschlagen. Dazu kommt eine Prozessorkarte mit einem MC68000 Prozessor von Motorola, auf der das Betriebssystem OS–9 [27] installiert ist, und eine Karte zur Ansteuerung einer Festplatte, so daß das gesamte System 36 VME–Steckplätze benötigt. Ein Crate besitzt 21 Steckplätze, so daß für diesen Trigger 2 Crates benötigt werden.

Die für den Triggerbetrieb notwendigen Programme werden in der Programmiersprache OMEGASOFT–PASCAL unter dem Betriebssystem OS–9 entwickelt. Die Logikinhalte der XILINX–Gatearrays werden auf einem IBM–AT unter MS–DOS erzeugt und über eine serielle Schnittstelle (RS–232, V–24) oder ein PC–VME– Interface[28] in das Triggercrate heruntergeladen. Die XILINX–Software ist nur auf IBM–AT Rechnern lauffähig, so daß es unmöglich ist, auf einen solchen Rechner zu verzichten. Zum Test der Triggerkarten ist eine Triggertestkarte entwickelt worden, die nicht während des Betriebs im Crate lokalisiert sein muß.

A.1.1 Die Triggerkarte

Die beiden Signaldrahtinformationen werden in einem Vorverstärker auf der FADC–Karte auf einen digitalen Pegel von 5V verstärkt. Dieser Ausgang kann direkt als Eingang für die Triggerkarten benutzt werden. Falls dieser digitale

[26] Versabus Module Europe [30]

[27] OS–9/68000 ist ein 'Multiuser', 'Multitasking' Betriebssystem der Firma Microware Systems

[28] PC/VME MASTER der Firma Ajida

Ausgang noch nicht zur Verfügung steht, soll das Signal des analogen Ausgangs der FADC–Karte auf einer Zwischenplatine, die auf die FADC–Karte aufgesteckt wird, zu einem digitalen Signal umgewandelt werden. Der Eingang auf einer solchen Zwischenkarte soll einen Eingangswiderstand von 50Ω, eine Bandbreite von einigen 10MHz und eine maximale Impulsanstiegszeit von einigen Nanosekunden ermöglichen. Als Eingangsverstärker wurde der MAR–1 gewählt, da er die Anforderungen erfüllt und preisgünstig ist. Diesem Vorverstärker ist ein Schmitt–Trigger und ein Leitungstreiber nachgeschaltet. Bei beiden handelt es sich um Standard–TTL[29]–Bausteine.

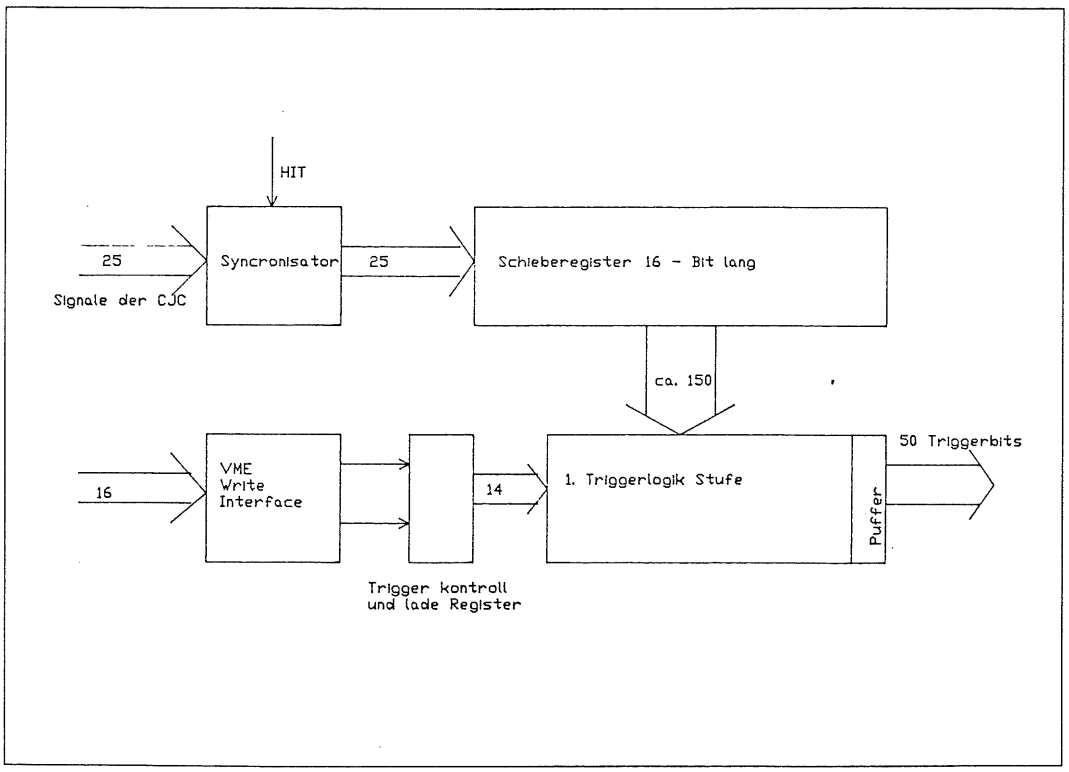

Abbildung 32: Blockschaltbild Triggerkarte

Im Falle, daß die FADC–Karte einen digitalen Ausgang erhält, kann auf die Eingangsverstärker verzichtet werden, und die Signale können direkt in die Synchronisierkreise auf den Triggerkarten geleitet werden. Die Synchronisation der Impulse mit der Bunchclock wird in Schaltkreisen, die von H. Krehbiel entwickelt wurden, realisiert. Diese Gatearrays haben 8 Synchronisatoren auf einem Chip und des weiteren 8 Schieberegister der Länge 32 mit seriellem Eingang und

[29]**T**ransistor **T**ransistor **L**ogik

Ausgang, die in der Triggerschaltung nicht verwendet werden. Die Ausgänge der Synchronisatoren sind, wie oben beschrieben, die Eingänge von Schieberegistern mit seriellem Eingang und parallelem Ausgang, falls es sich um einen Draht einer Meßlage handelt. Als Schieberegister wird ein Baustein vom Typ 74LS673 gewählt. Im anderem Falle, d.h. wenn es sich um einen Referenzdraht handelt, wird auch der Synchronisatorausgang an einen solchen Chip angeschlossen, wobei dieser nur als getaktetes, synchrones Verzögerungsglied von 16 Taktzyklen dient. Anschließend werden die Ausgänge der Schieberegister mit den XILINX–Gatearrays entsprechend der Verbindungsliste miteinander verbunden. Der logische Inhalt der Gatearrays wird beim Start des Systems von der Festplatte geladen. Des weiteren ist auf jeder Karte ein VME–Bus Slave Interface[30], welches die Adressmodifier und die unteren 8 Adressbits nicht dekodiert.

Jede Triggerkarte besitzt 25 Eingangssignale und liefert 50 Ausgangssignale. Sowohl Eingangs– als auch Ausgangssignale werden über den Frontendbus zu und von der Karte geleitet. Der VME–Bus Anschluß wird nur zum Laden der logischen Funktion in die XILINX–Bausteine benötigt.

A.1.2 Die Auslese und Pufferkarte

Um die Triggerentscheidung späteren Triggern und der Off–line Analyse zugänglich zu machen, müssen die Triggerbits weitergeleitet werden. Wenn der Trigger auf Doppel Eurokarten realisiert werden soll, dann müssen zusätzlich zu den Triggerkarten drei Auslese– und Pufferkarten existieren. Diese haben die Aufgaben, sowohl die Triggerbits zwischenzuspeichern und dem VME–Bus zugänglich zu machen, als auch die Gesamttriggerentscheidung des CJC–Triggers zu fällen.

Dazu werden die Triggerausgänge der XILINX–Chips aus der Triggerkarte durch den Frontendbus auf die Auslese– und Pufferkarte geschrieben. Auf dieser Karte sind zwei funktionelle Blöcke realisiert. Zum einen wird die Gesamttriggerentscheidung gefällt durch die Veroderung sämtlicher Triggersignale bzw. durch eine spezielle Logik, die gewisse Topologien der Signale verlangt, zum anderen wird, falls ein L2 Keep Trigger kommt, die Triggerinformation so lange gespeichert, bis ein L3 Keep oder ein Reject kommt. Die Pufferung der Daten wird bis der L2 Keep Trigger kommt, in Schieberegistern realisiert (für Testsetup, FSP) oder in einem RAM, welches mit einer Adressierlogik versehen ist, die einen Ringspeicher realisiert. Das RAM bzw. die Schieberegister sind über ein VME–Interface im VME Adressbereich adressierbar, so daß die

[30]Siehe Spezifikation VME–Bus [30]

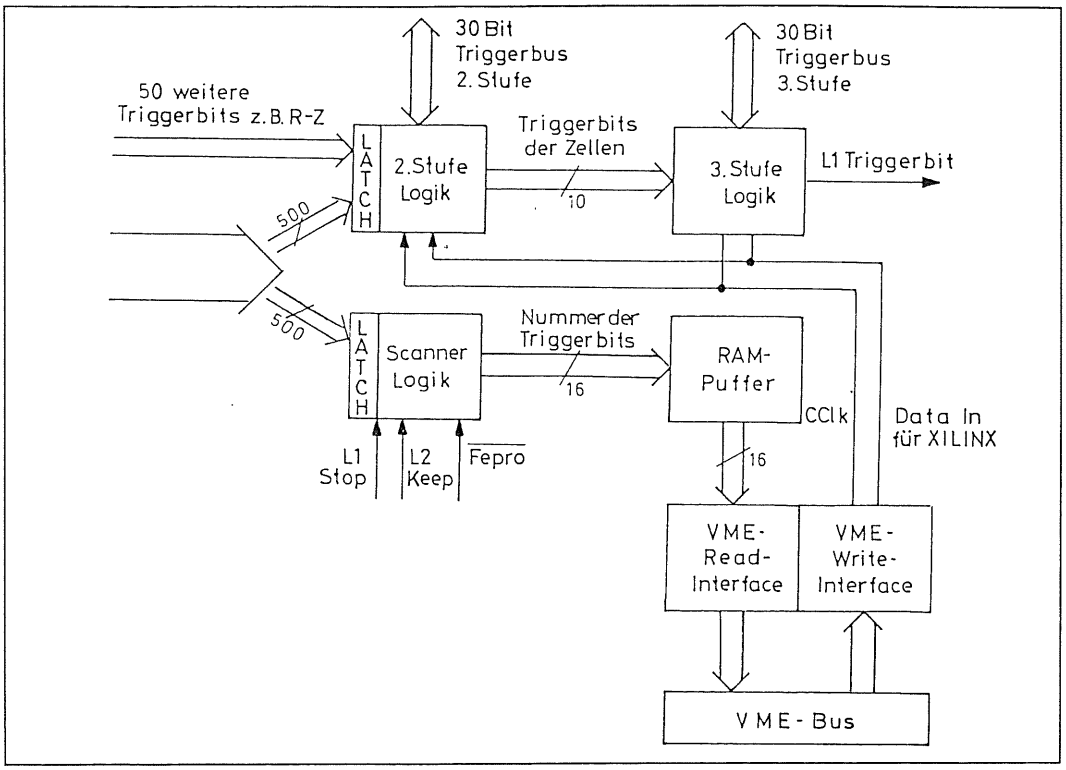

Abbildung 33: Blockschaltbild Auslesekarte

Triggerentscheidung über den Frontendprozessor mit ausgelesen werden kann. Der FSP–Trigger erzeugt pro Taktzyklus 150 Triggerbits, d.h. der H1 Trigger würde 1500 Triggerbits erzeugen. Damit nicht die kompletten 1500 Triggerbits gepuffert werden müssen, ist dem Puffer ein Scanner vorausgeschaltet, der die ankommenden Triggerdaten nach Einsen durchsucht und dann deren Adresse nur in den Puffer schreibt.

A.1.3 Die Triggertestkarte

Ein weiteres zentrales Problem ist der Test eines solchen Triggers. Um die Triggerkarten auf Funktionsfähigkeit zu testen, wurde ein Triggertestboard entwickelt. Dieses besitzt zwei große RAM–Blöcke, der eine 25 Bit, der andere 50 Bit breit organisiert. In dem 25 Bit breiten Speicher werden Testereignisse, die vorher auf einem Mikroprozessor berechnet wurden, über den VME–Bus eingeladen. Wenn dies geschehen ist, wird der VME–Bus vom Memory abgekoppelt, und der Speicher wird mit Bunchfrequenz ausgelesen. An den Datenausgängen der Speicherbausteine sind über ein Latch und einen Leitungstreiber

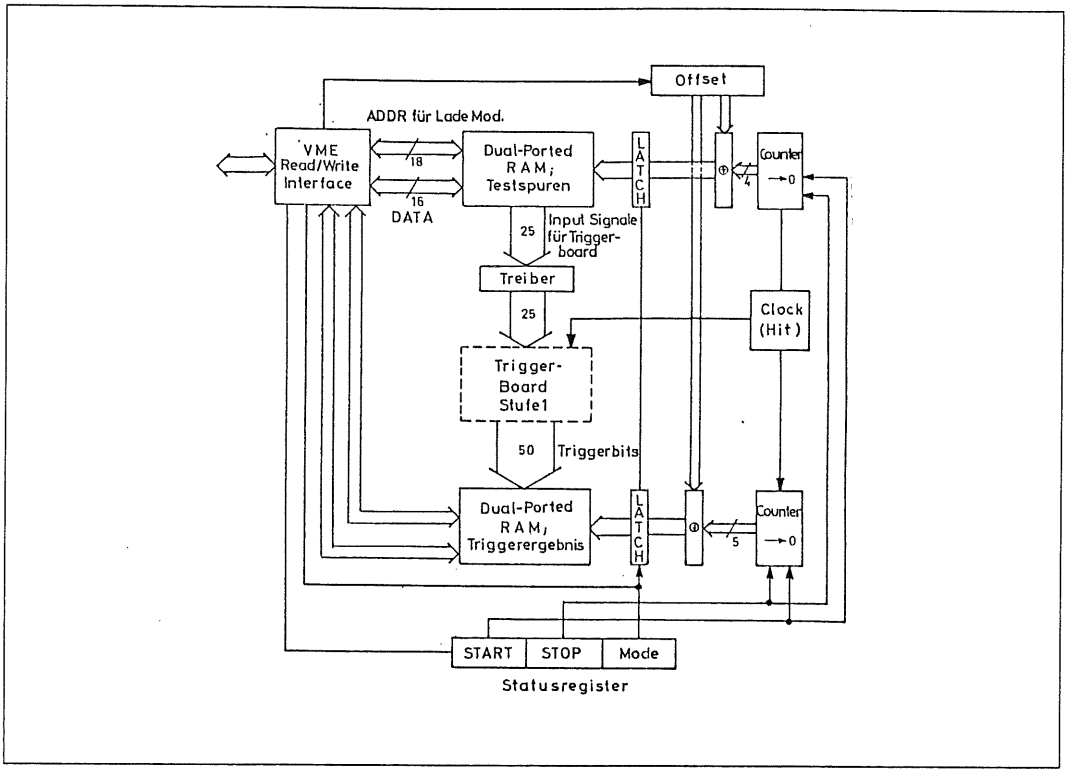

Abbildung 34: Blockschaltbild Triggertestkarte

die Eingänge der Triggerkarte verbunden. Die Ausgänge der Triggerkarte sind die Dateneingänge des 50 Bit breiten Memorys. In diesem Speicher werden die Triggerergebnisse einer Triggerkarte gesammelt. Nach einem Testlauf ist es möglich, diese Triggerbits mit einem Mikroprozessor zu analysieren. Bei diesem Speicher handelt es sich also um ein dual–ported–memory, das zum einem dem VME–Bus zugänglich ist, zum anderen der Triggerkarte. Dabei ist es, wenn man es von der VME–Seite aus anspricht, 16–Bit breit, wenn es die Triggerkarte testet, ist es 25–, 50–Bit breit.

Mit diesem Testboard hat man gerade in der Entwicklungsphase die Möglichkeit, Fehler auf den Triggerboards und in der Logik jedes einzelnen Logikarrays festzustellen.

A.1.4 Hardwarebedingte Seiteneffekte und das Timing des Triggers

Aufgrund der Gatterlaufzeiten in den Synchronisiergatearrays sowie in den Vorverstärkern ist die Genauigkeit der Position eines Bits in den Schieberegistern $(t-1)$Bit, d.h. das Bit ist im richtigen Zeitbit des Schieberegisters oder kommt

ein Bit zu spät. Die Logik in den Gatearrays muß gegenüber diesem Effekt fehlertolerant sein. Man erreicht dies, indem man die Grenzen der Triggerstraßen ein Bit überlappen läßt.

Wenn ein gesetztes Bit im Schieberegister auf eine Referenzposition trifft und die Zeitinformation in den Schieberegistern wegen der Überschneidung der Triggerstraßen auch eine wahre Triggerentscheidung liefern, liefert dieser Trigger eine wahre Triggerentscheidung . Dies ist in der Regel einen Taktzyklus später am Nachbardraht. Da die verkehrten Triggerimpulse immer nach dem eigentlichen Triggerpuls kommen, stellen sie für die Triggerbox kein weiteres Problem dar. Des weiteren besteht die Möglichkeit, in der Ausleselogik diese Pulse per Software zu unterdrücken.

A.2 Die Triggerbetriebssoftware

Die Betriebssoftware ist zweigeteilt, zum einen die Programme, die auf dem IBM–AT laufen, zum anderen die Programme, die auf dem VME–System laufen.

A.2.1 Die PC–Software

Auf dem PC laufen die Programme, um die Triggerlogik zu erzeugen und zu manipulieren. Dabei handelt es sich um ein CAD–System zur schematischen Eingabe der Triggerlogiken und um Dateikonvertier und Autorouteprogramme (Original XILINX Software), die die Dateien der Schemasoftware lesen und daraus den Inhalt der Gatearray RAMs berechnen, die den logischen Inhalt der Gatearrays speichern. Des weiteren sind auf dem PC die Programme installiert, um die Dateien vom PC auf das VME–System herunterzuladen , und zwar zum einen über eine serielle Verbindung (RS–232) oder zum anderen über ein PC–VME Parallelinterface. Wenn das PC–VME Interface zum Einsatz kommt, ist darauf zu achten, daß im VME–System ein Busarbiter existiert, der nicht auf der Mikroprozessor–Karte realisiert ist! Die PC–Programme sind in Turbo-Pascal geschrieben, da es in dieser Sprache möglich ist, Interruptroutinen zu schreiben und diese Eigenschaft für die Verbindungssoftware vonnöten ist. Die Programme laufen unter dem Betriebssystem MS–DOS 3.1 . Das Erzeugen der Triggerdaten soll in einem Programm, das aus den Triggerstraßengrenzen die Triggerlogik errechnet, geschehen und als PAL–Assemblerfile abspeichert werden, d.h. man gibt die Impulsgrenzen der Triggerstraßen ein, und das Programm errechnet die boolesche Funktion des Triggers. Zur Zeit ist dieser Teil nur durch Eingabe der Triggerlogik im schematischen Editor möglich. Danach wird das File mit der Triggerlogik durch teilweise selbstgeschriebene

Programme, teilweise durch original Xilinx– Software auf das Format vom Typ XNF (Xilinx Network File) gebracht. Dabei wird die Logik automatisch optimiert. Danach wird das Programm APR (Xilinx DN–23, Auto Place and Route) benutzt. Dieses hat die Aufgabe, die Verteilung der logischen Blöcke in dem Chip zu berechnen. Es benutzt dazu eine aus der Kristallzucht bekannte Methode. Zuerst werden die Blöcke stark in ihrer Position verändert, sie schwingen sich dann langsam auf ihre endgültige Position ein. Dieses Programm benötigt auf einem IBM–AT mit 8 Mhz Taktfrequenz ca. 1–2 Stunden CPU Zeit. Die dabei entstandenen Dateien werden durch ein weiteres Programm in einen Bitstream umgewandelt, welcher dann über das PC–VME–Interface oder die serielle Schnittstelle auf das VME–Zielsystem heruntergeladen werden kann. Auf der Festplatte dieses Systems werden die Dateien, die für den Triggerbetrieb notwendig sind, gelagert. Des weiteren gibt es auf diesem System die Gegenstücke zu der Transfersoftware auf dem IBM Rechner. Dieser Rechner dient weiter als Terminal für das VME–System.

A.2.2 Die VME–Software

Das VME–System besteht neben den Triggerkarten und den Auslesekarten aus einer Prozessorkarte E3 der Firma Eltec. Auf dieser Karte steht das Betriebssystem OS–9 zur Verfügung. OS–9 ist ein Echtzeitbetriebsystem, welches Multiuser– und Multiprozessingbetrieb zuläßt. Zur Programmentwicklung wurde die Programmiersprache OMEGA–SOFT Pascal benutzt. Diese hat die Möglichkeit, Variablen feste Adressen zuzuordnen. Die Gatearrays der Triggerkarten werden über ein Bit einer VME–Adresse adressiert. Einer Variablen des Typs hex (16 Bit) werden somit 10 Gatearrays, eine Taktleitung und eine Steuerleitung (siehe Anhang Schaltplan) zugeordnet. Es existieren die Programme, um die Gatearraydaten vom PC entweder über die serielle Schnittstelle oder das PC–VME Interface in Empfang zu nehmen. Des weiteren existieren die Programme, um die Triggerkarten zu laden, zu starten, zu stoppen und zu testen.

A.3 Aufbau und Test

Zur Zeit ist eine Triggerkarte fertig und getestet. Das komplette Transfersystem ist lauffähig, so daß dieser Trigger bei Fertigstellung des FSP für diese Testkammer zur Verfügung steht. Der FSP wird vermutlich nicht innerhalb des Zeitraums meiner Diplomarbeit fertiggestellt.

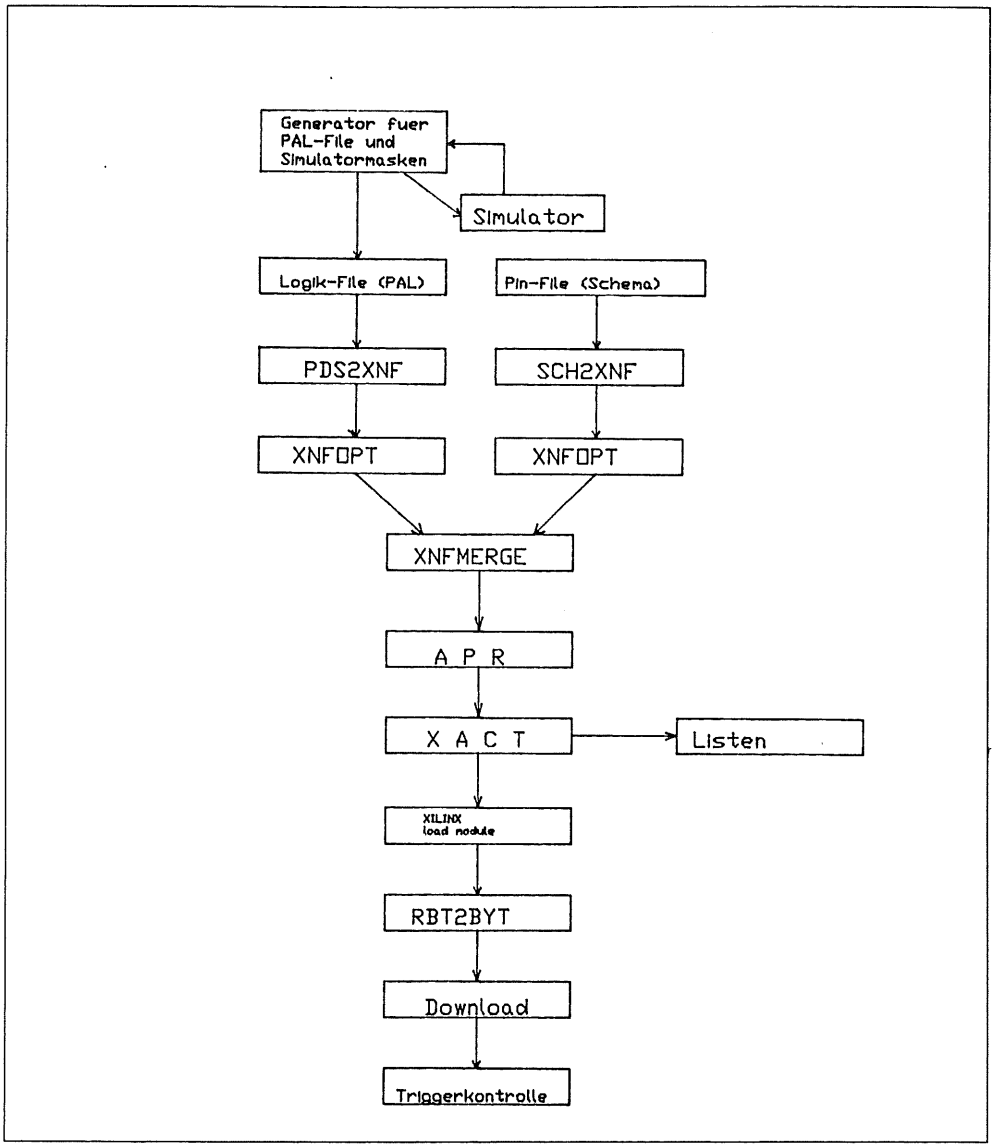

Abbildung 35: Datenfluß von Erzeugung der Triggerdaten bis zum Triggerlauf

B Die XILINX–Logic–Cell–Arrays

Da diese Bausteine relativ neu sind, sei hier eine kurze Beschreibung gegeben. Bei den XILINX–**L**ogic–**C**ell– **A**rrays (LCA) handelt es sich um softwareprogrammierbare Bausteine, deren Inhalt eine logische Funktion ist. Diese logische Function wird in konfigurierbaren Logikblöcken (CLB[31]) realisiert.

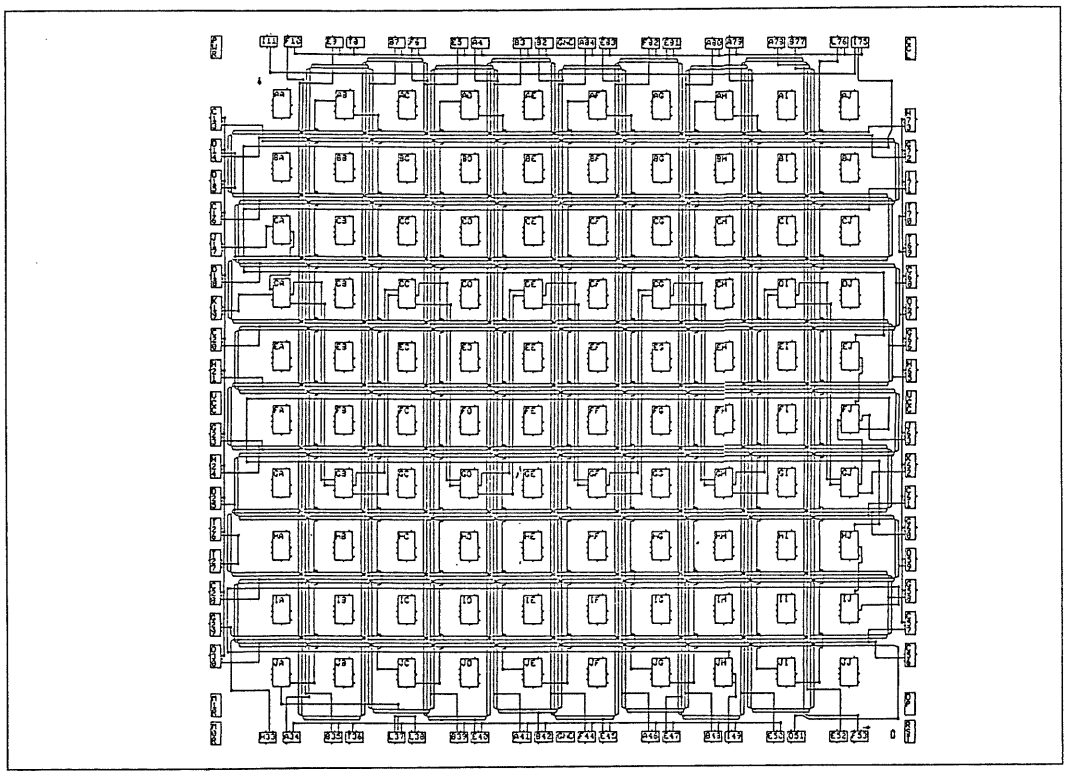

Abbildung 36: Anordnung der logischen Blöcke im XILINX–Chip

Diese Logikblöcke sind als Matrix auf dem Chip organisiert, siehe Abb. 36. Der Inhalt der Logikblöcke ist eine triviale logische Funktion und ein Steuerteil, der aus einem Multiplexer und einem Flip–Flop oder Latch besteht. Als triviale Functionen sind *UND, ODER, EXKLUSIV ODER*, sowie die *NEGATION* vorhanden. Die genaue Struktur der Logikblöcke ist in den Abbildungen 37 und 38 dargestellt.

Es sind zur Zeit zwei Chipserien mit unterschiedlichen Logikblöcken verfügbar. Die Serie 20xx, deren Logikblöcke eine logische Funktion von maximal 4 Variablen oder 2 logische Funktionen mit maximal 3 Variablen enthalten.

[31] Configurable **L**logic **B**locks

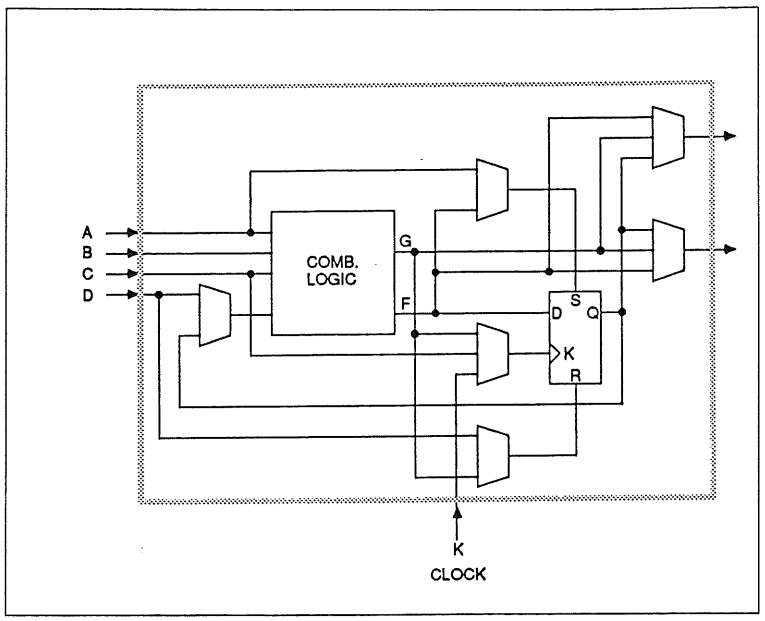

Abbildung 37: Aufbau eines Logikblocks von Chips aus der Serie 20xx

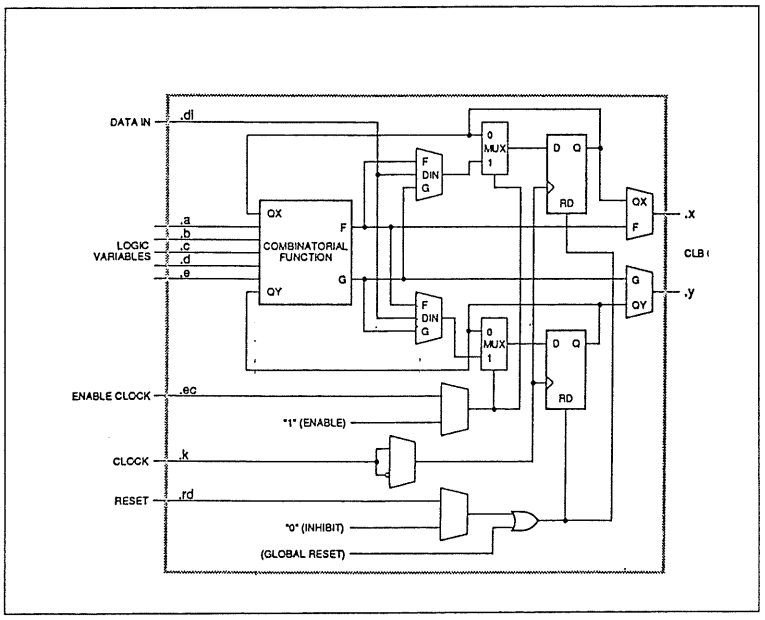

Abbildung 38: Aufbau eines Logikblocks von Chips aus der Serie 30xx

Weiter sind in den Logikblöcken dieser Serie ein Multiplexer und ein Flip–Flop konfigurierbar. Die Logikblöcke der Chips der 30xx Serie haben die Möglichkeit, entweder eine Funktion von 5 Variablen oder zwei Funktionen von 4 Variablen zu beinhalten. Dazu besitzen sie konfigurierbar einen Multiplexer und zwei Flip–Flops.

Neben konfigurierbaren Logikblöcken gibt es auch, je nach Chip, 58 bis 144 konfigurierbare Eingabe/Ausgabe Blöcke (IOB). Diese sind die Pins des Chips. Jeder dieser Blöcke ist konfigurierbar als *Eingang, Ausgang* oder *bidirektional.* Wenn ein Block als Ausgang konfiguriert wurde, steht optional ein Ausgangslatch zur Verfügung.

In jeder beiden Serien sind Chips unterschiedlicher Größe vorhanden. Eine Aufstellung über die verschiedenen Chiptypen ist in Tabelle 3 zu sehen.

Chipnummer	Kapazität (Gates)	Logikblöcke	User I/Os	≈ Preis in DM
XC2064	1200	64	58	50
XC2018	1800	100	74	100
XC3020	2000	64	64	130
XC3030	3000	100	80	
XC3042	4200	144	96	
XC3064	6400	224	120	
XC3090	9000	320	144	1000

Tabelle 3: Zusammenfassung der Logic–Cell– Arrays

Die Verbindung der Logikblöcke geschieht über verschieden kofigurierbare Leitungen innerhalb des Chips. Dabei gibt es zwei verschiedene Arten von Verbindungen, metallische Leitungen, die "keine" Impulsverzögerung (delay) haben und Leitungen aus Silizium (Interconnect) die bei der Verbindung von zwei benachbarten etwa eine Impulsverzögerung von 6ns haben [17].

Sowohl die Verbindungen als auch die Logikblöcke sind durch eine spezielle Software programmierbar. Die Logikfunktion erstellt man in einem schematischen Editor, mit den in der Elektronik üblichen Schaltsymbolen. Eine zweite Möglichkeit ist die Logikfunktion als Text einzugeben[32].

Vergleichbar ist solch ein Chip am besten mit einem RAM–Baustein. Der konfigurierbare Inhalt der XILINX–Chips ähnlich zu RAMs abgespeichert. Die

[32]Die Funktion muß im Format PAL–ASM eingegeben werden

Konfiguration geht bei abschalten der Stromversorgung verloren. Die Konfigureation dieser Chips ist entweder in EPROMs oder Datenträgern eines Computers gespeichert, also außerhalb des Chips.

Für den vorher beschriebenen Trigger bedeutet diese Eigenschaft, daß man verschiedene Algorithmen erzeugen und abspeichern kann und so ein hohes Maß an Freiheit und Testbarkeit des Algorithmuses hat.

C Programme

C.1 Transferprogram PC nach VME via PC–VME–Interface, VME–Seite

```
 1:0 program pc2vme(input,output);
 2:0
 3:1 var
 4:2    daten : array[1..32767] of integer at $400000;
 5:2    path  : string[30];
 6:2    term  : text;
 7:2
 8:2
 9:1 procedure cls;
10*2 begin
11*3  write(term,chr(27),'[1;1H',chr(27),'[2J');
12*2 end;
13:2
14:1 procedure pathname;
15*2 begin
16*3 cls;
17*3 writeln(term,'Change Path');
18*3 writeln(term);
19*3 write(term,'Enter Pathname (e.g. /h0/pas) :');
20*3 readln(term,path);
21*2 end;
22:2
23:1 procedure vme_to_disk;
24:1
25:2 var   datname : string[20];
26:3       outfile : file of byte;
27:3       loop,
28:3       hilf,
29:3       lop     : integer;
30:3       convert : boolean;
31:3       ch      : char;
32:3       savename: string[80];
33:3
34:3
35:2 function hi(a : integer) : integer;
36*3  begin
37*4   hi := a div 256 ;
38*3  end;
39:3
40:2 function lo(a : integer) : integer ;
41*3 begin
42*4  lo := a- (a div 256) * 256 ;
43*3 end;
44:3
45:2 procedure schreib(umlaut:char);
46*3 begin
47*4  write(outfile,umlaut,'e');
48*3 end;
49:3
50:2 procedure outbyte(zeichen:integer);
51*3 begin
52*4  if ((zeichen > 128) and convert) then
53:6   begin
54*7     case zeichen of
55*8       129 : schreib('u');
56*8       132 : schreib('a');
57*8       148 : schreib('o');
58*8       142 : schreib('A');
```

```
 59*8       153 : schreib('O');
 60*8       154 : schreib('U');
 61*8       225 : write(outfile,'s','s');
 62:7       end;        {case}
 63:6         end       {then}
 64*5       else write(outfile,chr(zeichen));
 65*3 end;              {proc}
 66:3
 67:3
 68:3
 69*2 begin
 70*3 cls;
 71*3 writeln(term,'Get Data from PC');
 72*3 writeln(term);
 73*3 datname:='                    ';
 74*3  loop := 1;
 75*3  lop := 1;
 76*3  repeat
 77*4   lop := lop+1;
 78*4   datname[loop] := chr(lo(daten[lop]));
 79*4   datname[loop+1] := chr(hi(daten[lop]));
 80*4   loop := loop+2;
 81*3  until lop = 8;
 82*3  writeln(term,datname);
 83*3  writeln(term);
 84*3  write(term,'Filename OK ? <Y/N> ');
 85*3  readln(term,ch);
 86*3  if ch in ['N','n'] then begin
 87*6                          writeln(term);
 88*6                          write(term,'Enter New Filename: ');
 89*6                          readln(term,datname);
 90:5                         end;
 91*3  writeln(term);
 92*3  write(term,'Convert Data ? <Y/N> ');
 93*3  readln (term,ch);
 94*3  if ch in ['y','Y'] then convert := true
 95*4                    else convert := false;
 96*3  hilf := 129 + (daten[1] div 2);
 97*3  savename:= concat(path,datname);
 98*3  create(outfile,savename,output);
 99*3  for loop := 129 to hilf do begin
100*5                 outbyte(hi(daten[loop]));
101*5                 outbyte(lo(daten[loop]));
102:4                 end;
103*3  close(outfile);
104:3
105*2 end;
106:2
107:2
108:2
109:1 procedure disk_to_vme;
110:1
111:2 var inname    : string[25];
112:3     infile    : file of char;
113:3     schreib,
114:3     scounter,
115:3     dcounter,
116:3     hilf      : integer;
117:3     by1,by2   : char;
118:3     loadname  : string[80];
119:3
120*2 begin
121*3 cls;
122*3 inname:='                    ';
123*3 writeln(term,'Transfer Data to PC');
124*3 writeln(term);
```

```
125*3 write(term,'Source Filename ==> ');
126*3 readln(term,inname);
127*3 loadname:=concat(path,inname);
128*3 open(infile,loadname,input);
129*3 dcounter := 129;
130*3 while not eof(infile) do
131:4       begin
132*5        read(infile,by1);
133*5        if not eof(infile) then read(infile,by2)
134*6                           else by2:=chr(0);
135*5        schreib:= ord(by1)*256+ord(by2);
136*5        daten[dcounter] := schreib;
137*5        dcounter := dcounter +1;
138:4       end;
139*3 close(infile);
140:3
141*3 daten[1] := (dcounter-129) * 2;
142*3 scounter := 1;
143*3 dcounter := 1;
144*3 repeat
145*4   dcounter := dcounter +1;
146*4   daten[dcounter]
:=((ord(inname[scounter+1])*256)+ord(inname[scounter])
147*4   scounter := scounter+2;
148*3 until dcounter = 8;
149:3
150*2 end;
151:2
152:1 procedure menu;
153:1
154:2 var  ch   : char;
155:2
156*2 begin
157*3 repeat
158*4  cls;
159*4  writeln(term);
160*4  writeln(term);
161*4  writeln(term);
162*4  writeln(term,' ':20,'1) Transfer Data To PC ');
163*4  writeln(term);
164*4  writeln(term,' ':20,'2) Receive Data from PC ');
165*4  writeln(term);
166*4  writeln(term,' ':20,'3) Change Pathname ');
167*4  writeln(term);
168*4  writeln(term,' ':20,'4) End Program ');
169*4  writeln(term);
170*4  writeln(term);
171*4  writeln(term,' ':20,'Your Choice ?:');
172*4  repeat
173*5    readln(term,ch);
174*4  until ch in ['1','2','3','4'];
175*4  if ch = '1' then disk_to_vme
176*5              else if ch = '2' then vme_to_disk
177*7                                else if ch = '3' then pathname;
178*3 until ch = '4';
179*2 end;
180:2
181*1 begin
182*2 path:='';
183*2 open(term,'/term',update);
184*2 menu;
185*1 end.
```

C.2 Transferprogram PC nach VME via PC–VME–Interface, PC–Seite

```
program PC_VME_datatransfer;

const pc_basis = $280;
type str80 = string[80];
var datname : str80;
{$U-}
{$C-}

{-------------------------------------------------------------------}

procedure head(schr: str80);
begin
clrscr;
textbackground(2);
gotoxy((80-length(schr)) div 2,1);
writeln(schr);
textbackground(0);
end;

{-------------------------------------------------------------------}

procedure request_bus;
var hilf1 : integer;
begin
port[pc_basis+1] := $80;  {request bus}
repeat
 hilf1:=port[pc_basis];
until (hilf1 and 2) <> 0;
port[pc_basis+2]:=$39;   {standard non-privileged data access}
end;

{-------------------------------------------------------------------}

procedure release_bus;
begin
port[pc_basis+1]:=0;
end;

{-------------------------------------------------------------------}

procedure set_addr(lbyte,hbyte: integer);
begin
 port[pc_basis+8]:=lbyte;
 port[pc_basis+9]:=hbyte;
 port[pc_basis+10]:=$40;
end;

{-------------------------------------------------------------------}

procedure vme_write(lbyte,hbyte:integer);
begin
  port[pc_basis+5]:=hbyte;
  port[pc_basis+4]:=lbyte;
  port[pc_basis]:=$0f;
end;

{-------------------------------------------------------------------}

procedure vme_read(var lbyte,hbyte : byte);
```

```
begin
   port[pc_basis]:=$0e;
   hbyte:=port[pc_basis+5];
   lbyte:=port[pc_basis+4];
end;

{-------------------------------------------------------------------------}

function disk_io_check : boolean;
var q : integer;
begin
 if ioresult <> 0 then begin
                       writeln('Disk Access Error !!!!!');
                       for q := 1 to 7 do begin
                                          sound(4400);
                                          delay(70);
                                          sound(3400);
                                          delay(170);
                                          end;
                       nosound;
                       disk_io_check:= false;
                       end
                  else disk_io_check := true;
{$i+}
end;

{-------------------------------------------------------------------------}

procedure press_key;
var ch : char;
begin
 gotoxy(30,22);
 write('Transfer completed !');
 gotoxy(24,24);
 write('Press any key to continue !');
 read(kbd,ch);
end;

{-------------------------------------------------------------------------}

procedure define_path;
var pfad : string[40];
begin
 head('Define Path');
 repeat
   {$i-}
   writeln;
   write('Pathname ? : ');
   readln(pfad);
   chdir(pfad);
 until disk_io_check;
end;

{-------------------------------------------------------------------------}

procedure vme_to_disk;
const lf : char = #10;

var outfile : file of char;
    laenge,
    adresse,
    hilf,
    loop,
    lauf   : integer;
    lb,hb   : byte;
```

```
    convert : boolean;
    ch1,
    ch2,
    ch      : char;

begin
head('Read Data From VME-Bus ! ');
writeln;
writeln;
datname:='';
 request_bus;
 set_addr(0,0);                      {get length}
 vme_read(lb,hb);
 laenge := (hb*256+lb) div 2  ;

hilf := 0;
repeat
   hilf:=hilf+2;
   set_addr(hilf,0);                    {get filename}
   vme_read(hb,lb);
   datname:= datname+chr(hb)+chr(lb);
until hilf=12;
release_bus;

 writeln(datname);
 writeln;
 writeln('Filename OK ? .Y/N/ ');
 writeln;
 repeat
   read(kbd,ch);
   ch := upcase(ch);
 until ch in ['Y','N'];
 if ch = 'N' then begin
                   write('Destination Filename : ');
                   readln(datname);
                  end;
 {$i-}
 assign(outfile,datname);
 rewrite(outfile);
 if not disk_io_check then exit;

 writeln;
 write('Insert LINEFEED's ? .Y/N/ ');
 writeln;
 repeat
   read(kbd,ch);
   ch :=upcase(ch);
 until ch in ['Y','N'];
 convert:= (ch = 'Y');

 request_bus;

 loop:=-2;
 lauf:=1;
 for hilf := 1 to (laenge) do
         begin
          loop:=loop+2;                          {read data}
          set_addr(loop,lauf);
          if loop = 254 then begin
                             loop:=-2;
                             lauf:=lauf+1;
                            end;

          vme_read(lb,hb);
          ch1 := chr(hb);
```

```
          ch2 := chr(lb);
          if ((hb=$0d) and convert) then begin
                                          write(outfile,ch1);
                                          write(outfile,lf);
                                         end
                                    else write(outfile,ch1);
          if ((lb=$0d) and convert) then begin
                                          write(outfile,ch2);
                                          write(outfile,lf);
                                         end
                                    else write(outfile,ch2);
        end;    {hilf}
 close(outfile);

 release_bus;
 press_key;
end;            {proc}

{-------------------------------------------------------------------}

procedure disk_to_vme;
var loop,
    lauf,
    laenge       : integer;
    infile       : file of byte;
    zw,zw2       : byte;
    ch           : char;
    noconvert    : boolean;

begin
 head('Write Data to VME-Bus !');
 writeln;
 write('Source Filename : ');
 readln(datname);
 writeln;
 write('Intercept LINEFEED ? .Y/N/ ');
 writeln;
 repeat
   read(kbd,ch);
   ch :=upcase(ch);
 until ch in ['Y','N'];
 noconvert := (ch='N');

 {$i-}
 assign(infile,datname);
 reset(infile);
 if not disk_io_check then exit;

 request_bus;
 laenge:=0;
 loop:=-2;
 lauf:=1;
 repeat
     repeat
       read(infile,zw);
     until ((zw<>$A) or noconvert);
                                          { intercept linefeed }
  repeat
  if not eof(infile) then read(infile,zw2)
                     else zw2 := 0;
  until ((zw2<>$A) or noconvert);

  loop:=loop+2;
  set_addr(loop,lauf);
  vme_write(zw2,zw);
   if loop = 254 then begin              {write data}
```

```
                    loop:=-2;
                    lauf:=lauf+1;
                  end;
  laenge := laenge+2;
 until eof(infile);
 release_bus;
 close(infile);

 request_bus;
 set_addr(0,0);                          {write length}
 vme_write(lo(laenge),hi(laenge));

 loop:=0;
 repeat
  loop:=loop+2;                            {write filename}
  set_addr(loop,0);
  vme_write(ord(datname[loop-1]),ord(datname[loop]));
 until loop=12;

 release_bus;
 press_key;
 end;

{-------------------------------------------------------------------}

procedure menu;
var ch : char;
begin
repeat
  clrscr;
  head('PC-VME DATA TRANSFER');
  gotoxy(1,7);
  writeln(' ':20,'1) Datatransfer  PC  ---/ VME ');
  writeln;
  writeln(' ':20,'2) Datatransfer  VME ---/ PC ');
  writeln;
  writeln(' ':20,'3) Define Pathname ');
  writeln;
  writeln(' ':20,'4) End Program');
  writeln;
  writeln;
  write(' ':20,'Your Choice ?');
  repeat
   read(kbd,ch);
  until ch in ['1','2','3','4'];
  if ch = '1' then disk_to_vme
              else if ch = '2' then vme_to_disk
                               else if ch = '3' then define_path;
  until ch = '4';
 end;

{-------------------------------------------------------------------}

begin
datname:='                                        ';
menu;
end.
```

C.3 Transferprogram PC nach VME via RS-232, PC–Seite

```
program Download;

{$K-U-C-}

uses CRT,DOS;

const
  comu1  = $3f8;
  comu2  = $2f8;
  comnr  = 1;
  Bufftop = 10000;
  IntContr : integer = $20;
  IRQ_Mask : integer = $21;
  EOI : byte = $20;

type
  ParTyp = (n,e,o);
  DBitTyp = 5..8;
  SBitTyp = 1..2;
  str2  =  string[2];

var
  regs        :  registers;
  Byteein,
  ComInt      :  byte;
  altcom      :  pointer;
  exitsave    :  pointer;
  com         :  word;
  BaudRate    :  real;
  inchar      :  char;
  infile      :  file of byte;
  inname      :  string[25];
  outs        :  str2;
  by          :  byte;
  zeb         :  byte;
  zei         :  byte;


var
  EinZeiger,
  AusZeiger   :  word;
  InBuff      :  array [0..BuffTop] of byte;
  keyout      :  char;
  NewByte     :  boolean;


procedure ReadCom;
  Interrupt;
  begin
    inc(Einzeiger);
    if (Einzeiger>Bufftop) then
      Einzeiger := 0;
    Inbuff[EInzeiger] := port[com];
    port[IntContr]:=EOI;
  end;


function InByte(var pb:boolean):byte;

  begin
    pb := (AusZeiger <> Einzeiger);
    if pb then begin
      inc(AusZeiger);
      if AusZeiger>Bufftop then
```

```
      AusZeiger := 0;
      InByte := InBuff[AusZeiger];
    end;
  end;


type  string8 = string[8];

function wbin(p:byte):string8;
var
  local :  byte;
  localstr : string8;

begin
  localstr := '';
  for local := 1 to 8 do begin
    if p>127 then localstr := localstr + '1'
      else localstr := localstr + '0';
    p := p shl 1;
    end;
  wbin := localstr;
end;



procedure SioInt_Aus;
begin
  Port[IRQ_Mask] := (Port[IRQ_Mask] or $18);
end;

procedure SioInt_ein;
begin
  Port[IRQ_Mask] := (Port[IRQ_Mask] and $e7);
end;


procedure initcom
          (BasisPort:integer;Baud:real;Databits:dbittyp;
           Stopbits:SBitTyp;parit:ParTyp);
var
  divisor  :  integer;
  local    :  byte;


begin
  divisor := round(115200.0 / Baud);
  port[Basisport+3] := $80;
  port[Basisport] := lo(divisor);
  port[Basisport +1] := hi(divisor);
  local := 0;
  case DataBits of
    8 :  local := 3;
    7 :  local := 2;
    6 :  local := 1;
  end;
  if (Stopbits = 2) then local := local or 4;
  case parit of
    e : local := local or $18;
    o : local := local or $08;
  end;
   port[Basisport + 3] := local;
   port[Basisport + 1] := 1;
   port[Basisport + 4] := $0B;
   local := port[Basisport];
 end;
```

```
procedure SioInfo;
begin
  writeln('Interr.-Maske  :',wbin(port[com+1]));
  writeln('Interr.-Status :',wbin(port[com+2]));
  writeln('Modem-Control  :',wbin(port[com+4]));
  writeln('Line-Control   :',wbin(port[com+3]));
  writeln('Line-Status    :',wbin(port[com+5]));
  writeln('Einzeiger      :',Einzeiger);
  writeln('Auszeiger      :',Auszeiger);
end;


procedure convert(n:byte ; var stc:str2);
var
  z : byte;

function ByteToHex(z0:byte):Char;
begin
  case z0 of
    0  :  ByteToHex := '0';
    1  :  ByteToHex := '1';
    2  :  ByteToHex := '2';
    3  :  ByteToHex := '3';
    4  :  ByteToHex := '4';
    5  :  ByteToHex := '5';
    6  :  ByteToHex := '6';
    7  :  ByteToHex := '7';
    8  :  ByteToHex := '8';
    9  :  ByteToHex := '9';
    10 :  ByteToHex := 'A';
    11 :  ByteToHex := 'B';
    12 :  ByteToHex := 'C';
    13 :  ByteToHex := 'D';
    14 :  ByteToHex := 'E';
    15 :  ByteToHex := 'F';
  end;{case}
end;{ByteToHex}

begin
  z := n div 16;
  n := n - ( z * 16 );
  stc[1] := ByteToHex(z);
  z := n div 1;
  stc[2] :=  ByteToHex(z);
end;




{$F+} procedure Myexit {$F-};
begin
  writeln;
  SioInt_Aus;
  Byteein := port[com];
  setintvec(comint,altcom);

{  sioinfo;}

  ExitProc := ExitSave;
end;
```

```
{********** Hauptprogramm **************}

begin
  writeln;
  write('Input TransferFileName ==> ');
  readln(inname);
  assign(infile,inname);
  reset(infile);


  baudrate := 9600;
  SioInt_Aus;
  com := comu1;
  ComInt := $0C;
  getintvec(comint,altcom);
  exitsave := exitproc;
  exitproc := @myexit;
  initcom(com,baudrate,8,1,n);
  setIntvec(ComInt,@readcom);
  einzeiger := 0;
  auszeiger := 0;
  SioInt_ein;
  clrscr;
  keyout := #$ff;
  outs := '  ';
  zeb := 0;

  while not eof(infile) do begin
    BYteEin := InBYte(newbyte);
    if newbyte then begin
      write(char(byteein));
    end;

    read(infile,by);
    convert(by,outs);

    BYteEin := InBYte(newbyte);
    if newbyte then begin
      write(char(byteein));
    end;
    while (port[com+5] and $20 = 0) do;
    port[com] := byte(outs[1]);

    BYteEin := InBYte(newbyte);
    if newbyte then begin
      write(char(byteein));
    end;
    while (port[com+5] and $20 = 0) do;
    port[com] := byte(outs[2]);

    zeb := zeb +2;

    if zeb=80 then begin

      BYteEin := InBYte(newbyte);
      if newbyte then begin
        write(char(byteein));
      end;
      while (port[com+5] and $20 = 0) do;
      port[com] := byte(chr(13));
      zeb := 0;
      delay(1000);
    end;
  end;
```

```
  for zei := zeb+1 to 79 do begin
    BYteEin := InBYte(newbyte);
    if newbyte then begin
      write(char(byteein));
    end;
    while (port[com+5] and $20 = 0) do;
    port[com] :=byte(chr(89));
  end;

  BYteEin := InBYte(newbyte);
  if newbyte then begin
    write(char(byteein));
  end;
  while (port[com+5] and $20 = 0) do;
  port[com] := byte(chr(90));

  BYteEin := InBYte(newbyte);
  if newbyte then begin
    write(char(byteein));
  end;
  while (port[com+5] and $20 = 0) do;
  port[com] := byte(chr(13));

  delay(100);
  BYteEin := InBYte(newbyte);
  if newbyte then begin
    write(char(byteein));
  end;
  delay(100);
  BYteEin := InBYte(newbyte);
  if newbyte then begin
    write(char(byteein));
  end;
  delay(100);
  BYteEin := InBYte(newbyte);
  if newbyte then begin
    write(char(byteein));
  end;
  delay(100);
  BYteEin := InBYte(newbyte);
  if newbyte then begin
    write(char(byteein));
  end;
end.
```

C.4 Transferprogram PC nach VME via RS-232, VME–Seite

```
 1:0 program dl(input,output);
 2:0
 3:1 type
 4:2   str3  = array [1..3] of char;
 5:2
 6:2
 7:1 var
 8:2   inname   :  string[25];
 9:2   outname  :  string[25];
10:2   termname :  string[25];
11:2   outfile  :  file of byte;
```

```
12:2   infile    :  text;
13:2   term      :  text;
14:2
15:2   ch        :  char;
16:2   cb        :  byte;
17:2
18:2   cfeld     :  array[1..2] of char;
19:2   zeile     :  string[80];
20:2
21:2   i         :  integer;
22:2   stp       :  boolean;
23:2
24:2
25:2
26:1 procedure convert(var by:byte);
27:1
28:2 var
29:3   j  :  integer;
30:3   z1 :  integer;
31:3   z2 :  integer;
32:3
33*2 begin
34*3   z1 := 0;
35*3   z2 := 0;
36*3   for j:=1 to 2 do begin
37*5     case cfeld[j] of
38*6       '1' : z1 := 1;
39*6       '2' : z1 := 2;
40*6       '3' : z1 := 3;
41*6       '4' : z1 := 4;
42*6       '5' : z1 := 5;
43*6       '6' : z1 := 6;
44*6       '7' : z1 := 7;
45*6       '8' : z1 := 8;
46*6       '9' : z1 := 9;
47*6       '0' : z1 := 0;
48*6       'A' : z1 := 10;
49*6       'B' : z1 := 11;
50*6       'C' : z1 := 12;
51*6       'D' : z1 := 13;
52*6       'E' : z1 := 14;
53*6       'F' : z1 := 15;
54:5     end; {case}
55*5     case j of
56*6       1   : z2 := z2 + (z1 * 16);
57*6       2   : z2 := z2 + z1;
58:5     end; {case}
59:4   end; {for}
60*3   by := chr(z2);
61*2 end ; {convert}
62:2
63:2
64*1 begin {dl}
65:1
66:1 {$i+,r+,c+}
67:1
68:1
69*2   termname := '/term';
70*2   open(term,termname,update);
71*2   writeln(term);
72*2   writeln(term);
73*2   writeln(term,'Download from IBM-PC via RS-232');
74*2   writeln(term,'Ver 2.01  16.09.1988');
75*2   writeln(term);
76*2   writeln(term);
77*2   write(term,'Infilename  ==> ');
78*2   readln(term,inname);
```

```
 79*2   if inname = '' then inname := '/term';
 80*2   write(term,'Outfilename ==> ');
 81*2   readln(term,outname);
 82*2   if outname = '' then outname := '/h0/daten/test';
 83*2   write(term,chr(1));
 84*2   close(term);
 85*2   open(infile,inname,input);
 86*2   create(outfile,outname,output);
 87*2   stp := false;
 88*2   cfeld := '  ';
 89:2
 90*2   while not stp do begin
 91*4     readln(infile,zeile);
 92*4     if zeile[80] <> chr(90) then begin
 93*7       for i:=1 to 40 do begin
 94*9         cfeld[1] := zeile[2*i -1];
 95*9         cfeld[2] := zeile[2*i];
 96*9         convert(cb);
 97*9         write(outfile,cb);
 98:8       end; {for}
 99:6     end {if}
100:5     else begin
101*7       i:=1;
102*7       while zeile[i] <> chr(89) do begin
103*9         cfeld[1] := zeile[i];
104*9         i:=i+1;
105*9         cfeld[2] := zeile[i];
106*9         i:=i+1;
107*9         convert(cb);
108*9         write(outfile,cb);
109:8       end; {while}
110*7       close(outfile);
111*7       stp := true;
112:6     end; {else}
113:3   end; {while}
114*1 end. {dl}
```

C.5 Triggerlogiken laden und starten

```
  1:0 program lt(input,output);
  2:0
  3:1 var
  4:2   i,k:  integer; {general variables}
  5:2   b  :  integer;
  6:2   bz :  byte;
  7:2   zw1,zw2 : integer;
  8:2
  9:2   b0  :  hex at $F04006; { Latches on the Triggerboard }
 10:2   b1 :  byte at $F04005;
 11:2   b2 :  byte at $F04003;
 12:2
 13:2   cou : integer;  { Bitcounter }
 14:2   error : boolean;
 15:2   ende  : boolean;
 16:2
 17:2   file0 : file of byte; { Devices, Dateien }
 18:2   fn0 : string[25];
 19:2
 20:2   term : text;
 21:2   tname : string[25];
 22:2
 23*1 begin
```

```
24:1
25:1   { Reset Trigger }
26:1
27*2   b0 := hex($C000);
28*2   b0 := hex($0300);
29*2   b0 := hex($0100);
30:2
31*2   fn0 := '/r0/dat0';
32*2   open(file0,fn0,input);
33:2
34*2   tname := '/term';
35*2   open(term,tname,output);
36:2
37*2   cou := 0;
38*2   error := false;
39*2   ende := false;
40*2   write(term,'Trigger is loading ==> ');
41*2   b2 := chr(127);
42:2
43*2   while not ende do begin
44*4     if eof(file0) then ende := true;
45*4     read(file0,bz);
46:4
47:4     { Daten auf L1 anlegen }
48:4
49*4     if bz = chr(48) then begin
50*7       b1 := chr(0);
51*7       b  := 0;
52:6     end else begin
53*7       b1 := chr(255);
54*7       b  := 3;
55:6     end;
56:6
57*4     zw1 := 127 or b;
58*4     zw2 := 128 or b;
59:4
60*4     b2 := chr(zw1);
61*4     b2 := chr(zw2);
62*4     b2 := chr(zw1);
63:4
64*4     cou := cou + 1;
65*4     if ((cou mod 1000) = 0) then write('.');
66:3   end;
67:3
68*2   for i:=1 to 10 do begin
69*4     b1 := chr(255);
70*4     b2 := chr(128);
71*4     b2 := chr(127);
72*4     cou := cou +1;
73:3   end;
74:3
75*2   b0 := hex($C100);
76:2
77*2   writeln(term,'done');
78*2   writeln(term,'Bitcount = ',cou);
79*2   close(file0);
80*2   close(term);
81*1 end.
```

C.6 Simulationsprogramme des CJC–Triggers

```
Program TrigGen;

{$N+}

{ ************************************************************************

  Version 2.4    20.02.1989

  Dieses Program erzeugt aus den Grenzimpulsen der Triggerstrassen
  eine (FORTRAN oder Pascal) Prozedure, die vom jeweiligen Trigsim-
  programm zur Triggersimulation benutzt werden kann, und es erzeugt
  10 PAL-ASM-Files, die zur XILINX-Software kompatibel sind, und aus
  denen die Chipfiles der Gatearrays errechnet werden koennen.
  Desweitern ermoeglicht es eine graphische Ausgabe der Triggerstrassen
  auf dem Monitor.
  Die Erzeugung der Triggerstrassen ist noch in Karthesischen Koordinaten!!

************************************************************************ }


uses crt,graph;

const
  vd      = 45000.0;          { Driftgeschwindigkeit in m/s }
  clk     = 96e-9;            { 96 ns }
  B       = 1.3;              { Feldstaerke in Tesla }
  c       = 2.998e8;          { Vakuumlichtgeschwindigkeit }
  p_min   = 420.0;            { Minimal Impuls }
  m_anz   = 6;                { Anzahl der Messlagen }
  l_anz   = 28;               { Anzahl der Doppellagen }
  ref_anz = 10;               { Anzahl der Referenzpukte }
  g_anz   = 6;                { Anzahl der Grenzspuren }


type
  real = double;

  str16 = string[16];

  Vektor = record
     r    : real;
     phi  : real;
   end;

   Track = record
             x0   : real;
             y0   : real;
             z0   : real;                { Start Koordinaten einer Spur }
             r    : real;                { Momentum, sign(r) = charge   }
             phi  : real;
             teta : real;                { Funktions Parameter einer Spur }
             x1   : real;
             y1   : real;
             z1   : real;                { End Koordinaten einer Spur }
```

```
  end;

  BinVektor = record
                l : integer;                { Doublelayer # }
                c : integer;                { Cell # }
                b : integer;                { Bin # }
  end;

  TrackBins = array [1..l_anz] of binvektor; { quantisierte Parameter einer
Spur


var                                         { Globale Variable }
  bins  : array[1..l_anz] of integer;       { bins pro Zelle in Lage in den
Mess
  anz_l : array[1..l_anz] of integer;       { anzahl der cell der Lage ( 30
oder




  r_l   : array[1..l_anz] of real;          { Radien der Doppellagen }
  phi_l : array[1..l_anz] of real;          { Startwinkel der Doppellagen }
  off_l : array[1..l_anz] of real;
  n_l   : array[1..m_anz] of integer;       { Nummern der Messlagen }
  p_g   : array[1..g_anz] of real;          { Grenzimpulse }
  r_g   : array[1..g_anz] of real;          { Grenzradien }
  bin_g : array [1..g_anz,1..l_anz,1..ref_anz] of binvektor; { Grenzen der
Trigg
  ref_p : array[1..ref_anz] of vektor;      { Koordinaten der Referenzpunkte }

  shift : array[0..59,1..28] of word;
  mask  : array[1..10,1..5,1..6,-5..5] of word;
  effcell  : array[-5..34] of integer;      { Kreisschluss }
  effcella : array[-5..64] of integer;
  relcell  : array[0..29] of integer;       { Relativ zu zell 0 }
  relcella : array[0..59] of integer;

  fn    : string[25];                       { filename des Fortran Files }
  ff    : text;                             { filetype des Fortran Files }
  outf  : text;                             { Protokollfile }
  lst   : text;                             { Printer Port }

{******************** Proceduren & Functions ****************************}


{$I trigo.sim}
> function arcsin(x:real):real;
> begin
> arcsin := arctan(x/sqrt(1-sqr(x)));
> end;
>
> function arccos(x:real):real;
> begin
> arccos:=(pi/2)-arcsin(x);
> end;
>
> function tan(x:real):real;
> begin
> tan:=sin(x)/cos(x);
```

```
> end;
>
> function cot(x:real):real;
> begin
> cot:=cos(x)/sin(x);
> end;
>
> function arccot(x:real):real;
> begin
> arccot:=(pi/2)-arctan(x);
> end;
>
> function sinh(x:real):real;
> begin
> sinh := 0.5*(exp(x)-exp(-x));
> end;
>
> function cosh(x:real):real;
> begin
> cosh := 0.5*(exp(x)+exp(-x));
> end;
>
> function tanh(x:real):real;
> begin
> tanh := (exp(x)-exp(-x))/(exp(x)+exp(-x));
> end;
>
> function coth(x:real):real;
> begin
> coth := (exp(x)+exp(-x))/(exp(x)-exp(-x));
> end;
>
> function arsinh (x:real):real;
> begin
```

```
> arsinh := ln(x+sqrt(sqr(x)+1));
> end;
>
>
> function arcosh(x:real):real;
> begin
> if x <= 0 then arcosh := ln(x-sqrt(sqr(x)-1));
> if x >  0 then arcosh := ln(x+sqrt(sqr(x)-1));
> end;
>
>
> function angle(x,y:real):real;
> begin
>
>   if x = 0 then
>     if y > 0 then
>        angle := pi/2
>     else
>       if y=0 then
>         angle := 0
>       else
>          angle := 3*pi/2
>   else begin
>     if ((x>0) and (y>=0)) then angle := arctan(y/x);
>     if ((x>0) and (y<0)) then angle := (2 * pi) + arctan(y/x);
```

```
>     if x<0 then angle := pi + arctan(y/x);
>   end;
> end;
>
>
> procedure RPhiToXY(r,phi:real;var x,y:real);
> begin
>   x := r * cos(phi);
>   y := r * sin(phi);
> end;
>
> procedure XYToRPhi(x,y:real;var r,phi:real);
> begin
>   r   := sqrt( sqr(x) + sqr(y) );
>   phi := angle(x,y);
> end;
{$I bit.sim}
> function bitset(byt,n:word):boolean;
> begin
>   bitset:=false;
>   bitset:=(byt and (1 shl n))=1 shl n;
> end;
>
> function clrbit(byt,n:word):word;
> begin
>   clrbit:=byt;
>   clrbit:=byt and not(1 shl n);
> end;
>
> function setbit(byt,n:word):word;
> begin
>   setbit:=byt;
>   setbit:=byt or 1 shl n;
> end;
>


procedure swap(var r,q:real);
var
 zw : real;
begin
  zw := r;
  r  := q;
  q  := zw;
end;

procedure swapi(var r,q:integer);
var
 zw : integer;




begin
  zw := r;
  r  := q;
  q  := zw;
end;


function Rs(p:real):real; { p in GeV/c, Kruemmunggsradius der Spur }
{ Diese Function berechnet aus dem Impuls einer Spur den Kruemmungsradius }
begin
  Rs := p * 1.0e6 / ( B * c );
```

```
end;

procedure ReadSpurparameter;
{ Diese procedure soll die Spurparameter einlesen,
  eigendlich sollte dies interaktiv geschehen, jedoch ist
  dies erst spaeter moeglich }
var
  i,j,k  :  integer;  { lv }
begin
  p_g[1] := -450;
  p_g[2] := -800;
  p_g[3] := -1600;
  p_g[4] :=  1600;
  p_g[5] :=  800;
  p_g[6] :=  450;

  for i:=1 to 6 do r_g[i] := Rs(p_g[i]);
  for i:=1 to 6 do writeln(lst,'==> Pg = ',p_g[i]:6:0);
  for i:=1 to 6 do writeln(lst,'==> Rg = ',r_g[i]:6:4);
  fn := 'trig.for';
  writeln;
end;


procedure set_vars;

var
  i : integer;

begin
  writeln(lst,'==> Variable werden definiert');

  for i:=1 to l_anz do r_l[i] := 0;          { loeschen der Felder }
  for i:=1 to l_anz do Phi_l[i] := 0;        { loeschen der Felder }

  for i:=1 to 12 do anz_l[i] := 30;          { Anzahl der Cellen im Layer }
  for i:=13 to 28 do anz_l[i] := 60;

  n_l[1] := 2;                               { Benutze Messlagen }
  n_l[2] := 4;
  n_l[3] := 7;
  n_l[4] := 9;
  n_l[5] := 11;
  n_l[6] := 15;

  r_l[1] :=  0.22225;                        { Radien der Doppellagen }
  r_l[2] :=  0.2385;
  r_l[3] :=  0.25535;
  r_l[4] :=  0.27265;
  r_l[5] :=  0.29035;
  r_l[6] :=  0.3084;
  r_l[7] :=  0.3267;
  r_l[8] :=  0.3452;
  r_l[9] :=  0.36385;
  r_l[10] := 0.3828;
  r_l[11] := 0.40185;
  r_l[12] := 0.42095;

  r_l[13] := 0.549;                          { aussen }
  r_l[14] := 0.5659;
  r_l[15] := 0.58295;
  r_l[16] := 0.6002;
```

```
r_l[17] := 0.6176;
r_l[18] := 0.6350;
r_l[19] := 0.6531;
r_l[20] := 0.6710;
r_l[21] := 0.6891;
r_l[22] := 0.7073;
r_l[23] := 0.7255;
r_l[24] := 0.7440;
r_l[25] := 0.7625;
r_l[26] := 0.7820;
r_l[27] := 0.7998;
r_l[28] := 0.8185;


phi_l[1] := -22.084;
Phi_l[2] := -19.045;                { Winkel der Zelle 0 }
Phi_l[3] := -17.68;
Phi_l[4] := -14.095;
Phi_l[5] := -12.065;
Phi_l[6] := -10.27;
Phi_l[7] := -8.67;
Phi_l[8] := -7.254;
Phi_l[9] := -5.965;
Phi_l[10] :=-4.81;
Phi_l[11] :=-3.765;
Phi_l[12] :=-2.818;

Phi_l[13] :=-13.62;                          { aussen }
Phi_l[14] :=-12.45;
Phi_l[15] :=-11.35;
Phi_l[16] :=-10.31;
Phi_l[17] :=-9.34;
Phi_l[18] :=-8.41;
Phi_l[19] :=-7.45;
Phi_l[20] :=-6.71;
Phi_l[21] :=-5.93;
Phi_l[22] :=-5.19;
Phi_l[23] :=-4.478;
Phi_l[24] :=-3.81;
Phi_l[25] :=-3.166;
Phi_l[26] :=-2.555;
Phi_l[27] :=-1.98;
Phi_l[28] :=-1.424;

for i:=1 to 28 do phi_l[i] := phi_l[i] * pi / 180.0;

for i:=1 to 28 do                              { Bins in den Messlagen }
  bins[i] := trunc( 2.0 * pi * r_l[i] / ( vd * clk * 2.0 * anz_l[i] ) ) + 1;

for i:=1 to m_anz do
  writeln(lst,'==> Anzahl bins in Lage ==> ',n_l[i],' ==> ',bins[n_l[i]]);

for i:=1 to ref_anz do begin                  { Definition der Referenzpukte }
  ref_p[i].r    := r_l[i+1];
  ref_p[i].phi  := phi_l[i+1];
end;

for i:=1 to 12 do off_l[i] :=  2 * pi / 60;
for i:=13 to 28 do off_l[i] := 2 * pi / 120;


for i:=-5 to -1 do effcell[i] := i+30;     { effzell schliesst Kreis }
for i:=-5 to -1 do effcella[i] := i+60;
```

```
  for i:=30 to 34 do effcell[i] := i-30;
  for i:=60 to 64 do effcella[i] := i-60;

  for i:=0 to 29 do effcell[i] := i;
  for i:=0 to 59 do effcella[i] := i;

  for i:=0 to 29 do relcell[i] := i;
  for i:=0 to 59 do relcella[i] := i;

  relcell[29] := -1;




  relcell[28] := -2;
  relcella[59] := -1;
  relcella[58] := -2;
  relcella[57] := -3;
  relcella[56] := -4;

end;



{ ************************************************************************}


procedure MakeLogic;

{ Die procedure Make-Logic ist das eigentliche Programm, sie
  beinhaltet zwei locale Procedure, naemlich Writelogik und
  endpunkt. }

var
  iref                      : integer;
  ip,im,iz                  : integer;
  a,b,q,p,r1,r2,xoff,yoff   : real;
  i,j,k                     : integer;
  r,phi,teta                : real;
  x1,y1,z1                  : real;
  x2,y2,x,y                 : real;
  phi1,phi2                 : real;
  cell,bit,layer            : integer;
  GraphDriver,
  GraphMode,
  ErrorCode,
  maxx,maxy                 : integer;
  ch                        : char;
  dxy,dphi                  : real;
  zoomx,zoomy               : real;
  xoffset,yoffset           : real;
  r0,phi0                   : real;
  sign                      : integer;
  zwmask                    : word;




  procedure makepoint(x,y   :real;
                      farbe   : integer);
  var
```

```
  ix,iy   : integer;
begin
   ix    := (maxx div 2) + trunc ( (maxx div 2) * x / 0.85 );
   iy    := (maxy div 2) - trunc ( (maxy div 2) * y / 0.85 );
   ix    := trunc( (ix + xoffset) * zoomx * dxy );
   iy    := trunc( (iy + yoffset) * zoomy);
   putpixel(ix,iy,farbe);
end;


procedure makepixel(r,phi   :real;
                    farbe   : integer);
var
  x,y     : real;
  ix,iy   : integer;
begin
   x     := r * cos(phi);
   y     := r * sin(phi);
   ix    := (maxx div 2) + trunc ( (maxx div 2) * x / 0.85 );
   iy    := (maxy div 2) - trunc ( (maxy div 2) * y / 0.85 );
   ix    := trunc( (ix + xoffset) * zoomx * dxy );
   iy    := trunc( (iy + yoffset) * zoomy);
   putpixel(ix,iy,farbe);




end;


procedure ClearRegister;
var
  i,j,k,l    : integer;
begin
  for i:=0 to 59 do
    for j:=1 to 28 do
      shift[i,j] := 0;
   for i:=1 to 10 do
     for j:=1 to 5 do
       for k:=1 to 6 do
         for l:=-5 to 5 do
           mask[i,j,k,l] := 0;
end;  { End ClearRegister }


procedure displayshift;
var
  i,j,m,b     : integer;
  w           : word;
  dphi        : real;
  phi         : real;


begin
  for i:=0 to 59 do begin
    for j:=1 to 28 do begin
      w := shift[i,j];
      if w<>0 then begin
        for m:=0 to 15 do begin
          if bitset(w,m) then begin
```

```
              b := 15-m;

              dphi :=  (2.0 * pi / ( 2.0 * anz_l[j] * bins[j])) * b;
              phi := phi_l[j] + (i * ( 2.0 * pi / anz_l[j]));
              makepixel(r_l[j],phi,lightred);    { Display Wire Position }

              phi := phi + dphi;
              makepixel(r_l[j],phi,lightmagenta); { Display Triggerbit
Position
              phi := phi - (2*dphi);
{              makepixel(r_l[j],phi,lightcyan);    { Display Triggerbit
Position

            end; { end if }
          end; { end for }
        end; { end if }
      end; { end for }
    end; { end for }
  end; { end displayshift }


   procedure ShiftGleichMask;

   var
     i,j,k,l,m  : integer;

   begin
     i:=1;

     for j:=1 to 5 do
       for m:=1 to 6 do begin

         k:=-5;
         if m<>6 then l:=29 else l:=3;
         shift[l,n_l[m]] := shift[l,n_l[m]] or mask[i,j,m,k];

         for k:=-4 to 5 do begin
             if m<>6 then l:=k+4 else l:=k+8;
           shift[l,n_l[m]] := shift[l,n_l[m]] or mask[i,j,m,k];
        end;









     end;
   end;



{ ********************* Schnittpunkt ********************************** }


  function SchnittPunkt(rl,rt,r0,phi0 : real;  l : integer) : real;
{ Diese Function liefert den Schnittpunkt von zwei Kreisen in
  Polarkoordinaten. }

  begin
    if ((r0 < rl+rt) and (rt<r0+rl)) then
      if l=1 then
```

```
        Schnittpunkt := arccos( ( sqr(r0) + sqr(rl) - sqr(rt) ) / ( 2 * r0 *
rl
      else
        Schnittpunkt := -arccos( ( sqr(r0) + sqr(rl) - sqr(rt) ) / ( 2 * r0 *
rl
    else
      Schnittpunkt := -100
  end;



  function bits(z:word):str16;
  var
    i : integer;
    bi  : str16;
  begin
    bi:='';
    for i:=15 downto 0 do
      if bitset(z,i) then bi := bi + '1' else bi := bi + '0' ;
    bits := bi;
  end;


  procedure WriteLogic;

  var
    iref,im,ip,ig,i       : integer;                    { Laufvariable }
    bit,layer,cell        : integer;
    bit1,cell1,layer1     : integer;
    bit2,cell2,layer2     : integer;
    triggerbit            : integer;
    refcell               : integer;
    start,stop            : integer;
    icell,cstart,cend     : integer;
    zw1,zw2               : integer;
    zwmask                : word;

  begin  { WriteLogic }


    triggerbit := 0;

    writeln(outf,'procedure setmask;');
    writeln(outf);
    writeln(outf,'begin');
    writeln(outf);

    for iref := 1 to 1 {ref_anz} do begin  { loop ueber referenzpunkte }
                                           { 10 }
      for ig := 1 to  g_anz-1 do begin      { loop ueber 5 triggerstrassen }
                                           { 5 }
        for im := 1 to m_anz do begin       { loop ueber messlagen einer
trigger
                                           { 6 }

          cell1  := bin_g[ig,n_l[im],iref].c;




          cell2  := bin_g[ig+1,n_l[im],iref].c;
          bit1   := bin_g[ig,n_l[im],iref].b;
```

```
          bit2   := bin_g[ig+1,n_l[im],iref].b;
          layer  := bin_g[ig,n_l[im],iref].l;
          zwmask := 0;

          if (cell1 = cell2) then begin
            cell := cell1;
            if (bit1 > bit2) then swapi(bit1,bit2);
              start := bit1;
              stop  := bit2;
              for ip := start to stop do begin
                zwmask := setbit(zwmask,15-abs(ip));
              end; { end for ip }
{               if im=6 then cell:=cell-8 else cell:=cell-4;    {}
              zw1:=0;
              zw2:=15;
              while not bitset(zwmask,zw1) do zw1:=zw1+1;
              while not bitset(zwmask,zw2) do zw2:=zw2-1;
              zwmask := setbit(zwmask,zw1);
              zwmask := setbit(zwmask,zw2);
              writeln(outf,'  mask[',iref,',',ig,',',im,',',cell,'] :=
',{bits{}
              mask[iref,ig,im,cell] := zwmask;
            end { end if cell1 = cell2 }
            else begin

              if (cell1 > cell2) then begin
                swapi(cell1,cell2);
                swapi(bit1,bit2);
              end;

              cstart := cell1;
              cend   := cell2;

              for icell := cstart to cend do begin


                if (icell=cstart) then begin
                  start := bit1;
                  stop  := 15;
                  for ip := start to stop do begin
                    zwmask := setbit(zwmask,15-abs(ip));
                  end; { end for ip }
{                  if im=6 then cell:=icell-8 else cell:=icell-4;{}
                  zw1:=0;
                  zw2:=15;
                  while not bitset(zwmask,zw1) do zw1:=zw1+1;
                  while not bitset(zwmask,zw2) do zw2:=zw2-1;
                  zwmask := setbit(zwmask,zw1);
                  zwmask := setbit(zwmask,zw2);
                  writeln(outf,'  mask[',iref,',',ig,',',im,',',cell,'] :=
',{bi
                  mask[iref,ig,im,cell] := zwmask;
                end; { end if icell = cstart }


                if (icell=cend) then begin
                  start := 15;
                  stop  := bit2;
                  for ip := start downto stop do begin
                    zwmask := setbit(zwmask,15-abs(ip));
                  end; { end for ip }
{                  if im=6 then cell:=icell-8 else cell:=icell-4;{}
                  zw1:=0;
                  zw2:=15;
                  while not bitset(zwmask,zw1) do zw1:=zw1+1;
```

```
                  while not bitset(zwmask,zw2) do zw2:=zw2-1;
                  zwmask := setbit(zwmask,zw1);
                  zwmask := setbit(zwmask,zw2);
                  writeln(outf,'  mask[',iref,',',ig,',',im,',',cell,'] :=
',{bi
                  mask[iref,ig,im,cell] := zwmask;
                end; { end if icell=cend }


                if ((icell<>cstart) and (icell<>cend)) then begin




                  start := 0;
                  stop  := 15;
                  for ip := start to stop do begin
                    zwmask := setbit(zwmask,15-abs(ip));
                  end; { end for ip }
{                  if im=6 then cell:=icell-8 else cell:=icell-4;{}
                  zw1:=0;
                  zw2:=15;
                  while not bitset(zwmask,zw1) do zw1:=zw1+1;
                  while not bitset(zwmask,zw2) do zw2:=zw2-1;
                  zwmask := setbit(zwmask,zw1);
                  zwmask := setbit(zwmask,zw2);
                  writeln(outf,'  mask[',iref,',',ig,',',im,',',cell,'] :=
',{bi
                  mask[iref,ig,im,cell] := zwmask;
                end; { end if <> <> }


              end; { end for icell }

            end; { end else }

        end; { end im }

        triggerbit := triggerbit + 1 ;
        writeln(outf);

      end; { end ig }

      writeln(outf);
      writeln(outf,'{ ********************************** Lage
****************
      writeln(outf);
      writeln(outf);

    end; { end iref }

    writeln(outf,'end;');

  end;  { end writelogic }


begin     { MakeLogic }
  iz      :=  0;
  zoomx   :=  5.0;
  zoomy   :=  4.0;
  xoffset :=  -350;
  yoffset :=  -170;
```

```
GraphDriver := Detect;
InitGraph (GraphDriver,GraphMode,'');
ErrorCode := GraphResult;
if ErrorCode <> grOK then
begin
  writeln ('Grafikfehler');
  Halt(1)
end;
maxx := getmaxx;   maxy := getmaxy;   dxy := maxy / maxx;
ClearDevice;


                        { Kreise der Lagen }
 for i:=1 to 6 do
   for j:=1 to 2 * bins[n_l[i]] * anz_l[n_l[i]] do
     makepixel(r_l[n_l[i]],(2*pi/350)*j,red); {}


 for i:=1 to 28 do begin
   for j:=1 to anz_l[i] do begin            { Sensdraehte zeichnen }
     r := r_l[i];
     phi := phi_l[i] + j * ( 2.0 * pi / anz_l[i]);
     makepixel(r,phi,lightgray);
   end;




 end;

 for i:=1 to 28 do begin
   for j:=1 to anz_l[i] do begin            { Zellgrenzen }
     r := r_l[i];
     phi := phi_l[i] + j * ( 2.0 * pi / anz_l[i]) + off_l[i];
     makepixel(r,phi,darkgray);
   end;
 end;


 for i:=1 to 28 do begin                      { Cell# 0 makieren }
   r := r_l[i];
   phi := phi_l[i];
   makepixel(r,phi,lightmagenta);
 end;


                                          { Koordinatensystem }
for i:=-maxx to maxx do makepoint(i*(0.85/maxx),0,white);
for i:=-maxy to maxy do makepoint(0,i*(0.85/maxy),white);

for iref := 1  to 1 {ref_anz} do begin    { Loop ueber die Referenzpunte }
                                          { 10 }
  for ip := 1 to g_anz do begin           { Loop ueber die Grenzen }
                                          { 6 }
    r0  := r_g[ip];                       { GrenzRadius }
    r   := abs(ref_p[iref].r);            { Basis (R/Phi) von Referenz Layer
```

```
}
      if r < 0 then sign := -1              { Ladungs Vorzeichen }
      else sign := 1;
      phi := ref_p[iref].phi;               { Die Bedingung fuer die
Triggergren
                                              ist, dass diese Spur durch den
Ver
                                              und durch den Ursprung geht, bei
v
                                              gegebenen Radius }

      phi0 := -arccos( r / (2 * r0)) + phi;  { Winkel zum Mittelpunkt der
Grenzs
      if phi0 < 0 then phi0 := phi0 + 2 * pi;

      yoff := r_g[ip] * sin(phi0);
      xoff := r_g[ip] * cos(phi0);
                                            { triggerstrassen }
      for i:=1 to 5000 do makepoint(r_g[ip] * cos((2*pi/5000)*i)+xoff,
                                    r_g[ip] * sin((2*pi/5000)*i)+yoff,blue);


      for im := 1 to l_anz do begin         { loop ueber alle Lagen einer Spur
}
                                            { 28 }

        phi := schnittpunkt(r_l[im],r0,r0,phi0,sign); { liefert Schnittpunkt
der

        if phi < 0 then phi := phi + 2 * pi;    { 0 --> 2c }

        makepixel(r_l[im],phi, green);      { zeichnet wahre Position einer
Gren

        phi2 := pi / anz_l[im];             { Winkel einer halben celle }

        dphi := abs(phi - phi_l[im]);
        if dphi>2*pi then dphi := dphi - 2*pi;

        cell := trunc ( (dphi + phi2) / ( 2 * phi2 ) );
                                            { Cell# of track in layer im}

        bit  := trunc ( ( dphi  -  cell * 2 * phi2) * r_l[im]
                     / ( vd * clk ) ) ;     { Bit# of track in cell }

        layer := im;

        if ( (layer=2) or
             (layer=4) or
             (layer=7) or




             (layer=9) or
             (layer=11) or
             (layer=15)) then
          writeln(lst,'==> Bit# = ',bit:2,'    Cell# = ',cell:2,'    Layer# =
',

        if im < 12 then cell := effcell[cell]
                   else cell := effcella[cell];
```

```
        if im < 12 then cell := relcell[cell]
                   else cell := relcella[cell];


        bin_g[ip,im,iref].l := layer;
        bin_g[ip,im,iref].c := cell;
        bin_g[ip,im,iref].b := bit;

        r    := r_l[layer];
        dphi :=  (2.0 * pi / ( 2.0 * anz_l[layer] * bins[layer])) * bit;
        phi1 := phi_l[layer] + (cell * ( 2.0 * pi / anz_l[layer]));

        makepixel(r,phi1,cyan);                { Wire Position }

        phi1 := phi1 + dphi;
        makepixel(r,phi1,yellow);              { Trackpostion }

      end;  { End im }

      writeln(lst,'==> pg   =',p_g[ip]:5:0,'  refl  = ',iref+1:2);
      writeln(lst,'==> ------------------------------------------------');

    end;  { End For ip }

  end;    { End For iref }

  clearregister;
  writelogic;
  shiftgleichmask;
  displayshift;

  repeat
    write(^G);
    ch := readkey;
  until ch <> '';

  ClearDevice;
  CloseGraph;
  RestoreCrtMode;


end;      { End MakeLogic }

{ ********************** MainProgramm ***************************** }


begin { Hauptprogramm }
  clrscr;
  writeln('==> running');
  assign(lst,'nul');              { nul == abgeschalted }
  rewrite(lst);
  assign(outf,'mask.sim');
  rewrite(outf);
  set_vars;
  ReadSpurparameter;
  MakeLogic;
  close(lst);
  close(outf);
  writeln('==> ready');
end.
```

```
Program SpurSimulation(input,output);

{$B-}  { Der Compiler bricht bei Eindeutigkeit die Berechnung eines
         boolschen Ausdruckes ab }
{$N+}



{ ************************************************************************

  Version 2.2   30.01.1989; Schnitpunktsbrechnung in Polarkoordinaten

  Dieses Program stellt die Softwareschnittstelle zu den
  Eventdaten auf der IBM bzw. zu frei waelbaren Spuren dar. Dabei
  soll es zwei Moeglichkeiten der Triggerung besitzen. Erstens kann
  es die durch das Prgramm TrigGen generierte TriggerProcedure
  benutzen koennen, zweitens soll selbststaendig durch benutzten der
  Triggergrenzen Triggern koennen. Ein graphisches Eventdisplay
  ist auch vorhanden.

  ************************************************************************ }



uses crt,graph;



const

  vd      = 45000.0;                    { Driftgeschwindigkeit in m/s }
  clk     = 96e-9;                      { 96 ns }
  B       = 1.3;                        { Feldstaerke in Tesla }
  c       = 2.998e8;                    { Vakuumlichtgeschwindigkeit }
  p_min   = 420.0;                      { Minimal Impuls }
  m_anz   = 6;                          { Anzahl der Messlagen }
  l_anz   = 28;                         { Anzahl der Doppellagen }
  ref_anz = 10;                         { Anzahl der Referenzpukte }
  g_anz   = 6;                          { Anzahl der Grenzspuren }


type

{  real = single;  {}


  Vektor = record
             r    : real;
             phi  : real;
   end;

   Track = record
             x0   : real;
             y0   : real;
             z0   : real;                { Start Koordinaten einer Spur }
             r    : real;                { Momentum, sign(r) = charge   }
             phi  : real;
```

```
            teta : real;                    { Funktions Parameter einer Spur }
            x1   : real;
            y1   : real;
            z1   : real;                    { End Koordinaten einer Spur }
  end;

  BinVektor = record
                l : integer;                { Doublelayer # }
                c : integer;                { Cell # }
                b : integer;                { Bin # }
  end;



  TrackBins = array [1..l_anz] of binvektor; { quantisierte Parameter einer
Spur

var                                         { Globale Variable }

  bins  : array[1..l_anz] of integer;       { bins pro Zelle in Lage in den
Mess
  anz_l : array[1..l_anz] of integer;       { anzahl der cell der Lage ( 30
oder
  r_l   : array[1..l_anz] of real;          { Radien der Doppellagen }
  phi_l : array[1..l_anz] of real;          { Startwinkel der Doppellagen }
  n_l   : array[1..m_anz] of integer;       { Nummern der Messlagen }
  p_g   : array[1..g_anz] of real;          { Grenzimpulse }
  r_g   : array[1..g_anz] of real;          { Grenzradien }
  bin_g : array [1..g_anz,1..l_anz,1..ref_anz] of binvektor; { Grenzen der
Trigg
  ref_p : array[1..ref_anz] of vektor;      { Koordinaten der Referenzpunkte }
  bits  : array[0..29,1..5,1..10] of boolean;       { Triggerbits }
  shift : array[0..59,1..28] of word;       { Schieberegister }

  mask     : array[1..ref_anz,1..g_anz-1,1..m_anz,-5..5] of word; {
Maskenregist
  effcell  : array[-5..34] of integer;       { Kreisschluss }
  effcella : array[-5..64] of integer;

  fn    : string[25];                       { filename des Fortran Files }
  ff    : text;                             { filetype des Fortran Files }
  outf  : text;                             { Protokollfile }
  lst   : text;                             { Printer Port }
  monout: text;                             { Console out }
  monin : text;                             { Console in }



{******************** Proceduren & Functions ****************************}



{$I trigo.sim}     { Includes the trigometrical functions }
> function arcsin(x:real):real;
> begin
> arcsin := arctan(x/sqrt(1-sqr(x)));
> end;
>
```

```
> function arccos(x:real):real;
> begin
> arccos:=(pi/2)-arcsin(x);
> end;
>
> function tan(x:real):real;
> begin
> tan:=sin(x)/cos(x);
> end;
>
> function cot(x:real):real;
> begin
> cot:=cos(x)/sin(x);
> end;
>
> function arccot(x:real):real;
> begin
> arccot:=(pi/2)-arctan(x);
> end;
>
> function sinh(x:real):real;
> begin
> sinh := 0.5*(exp(x)-exp(-x));
> end;
>
> function cosh(x:real):real;
> begin
> cosh := 0.5*(exp(x)+exp(-x));
> end;
>
> function tanh(x:real):real;
> begin
> tanh := (exp(x)-exp(-x))/(exp(x)+exp(-x));
```

```
> end;
>
> function coth(x:real):real;
> begin
> coth := (exp(x)+exp(-x))/(exp(x)-exp(-x));
> end;
>
> function arsinh (x:real):real;
> begin
> arsinh := ln(x+sqrt(sqr(x)+1));
> end;
>
>
> function arcosh(x:real):real;
> begin
> if x <= 0 then arcosh := ln(x-sqrt(sqr(x)-1));
> if x >  0 then arcosh := ln(x+sqrt(sqr(x)-1));
> end;
>
>
> function angle(x,y:real):real;
> begin
>
>   if x = 0 then
>     if y > 0 then
>        angle := pi/2
```

```
>      else
>        if y=0 then
>          angle := 0
>        else
>           angle := 3*pi/2
>    else begin
>      if ((x>0) and (y>=0)) then angle := arctan(y/x);
>      if ((x>0) and (y<0)) then angle := (2 * pi) + arctan(y/x);
>      if x<0 then angle := pi + arctan(y/x);
>    end;
> end;
>
>
> procedure RPhiToXY(r,phi:real;var x,y:real);
> begin
>    x := r * cos(phi);
>    y := r * sin(phi);
> end;
>
> procedure XYToRPhi(x,y:real;var r,phi:real);
> begin
>    r   := sqrt( sqr(x) + sqr(y) );
>    phi := angle(x,y);
> end;
      { Includes the trigometrical functions }
{$I bit.sim}       { Includes the bit operation on integers }
> function bitset(byt,n:word):boolean;
> begin
>    bitset:=false;
>    bitset:=(byt and (1 shl n))=1 shl n;
> end;
>
> function clrbit(byt,n:word):word;
> begin
>    clrbit:=byt;
>    clrbit:=byt and not(1 shl n);
> end;
>
> function setbit(byt,n:word):word;
> begin
>    setbit:=byt;
>    setbit:=byt or 1 shl n;
> end;
>
       { Includes the bit operation on integers }

procedure swap(var r,q:real);




var
 zw : real;
begin
  zw := r;
  r  := q;
  q  := zw;
end;


procedure swapi(var r,q:integer);
var
```

```
 zw : integer;
begin
  zw := r;
  r  := q;
  q  := zw;
end;


function Rs(p:real):real; { p in GeV/c, Kruemmunggsradius der Spur }
{ Diese Function berechnet aus dem Impuls einer Spur den Kruemmungsradius }
begin
  Rs := p * 1.0e6 / ( B * c );
end;


procedure set_vars;

var
  i,j,k,l : integer;

begin
  writeln('==> Variable werden definiert');

  for i:=1 to l_anz do r_l[i] := 0;          { loeschen der Felder }
  for i:=1 to l_anz do Phi_l[i] := 0;        { loeschen der Felder }

  for i:=1 to 12 do anz_l[i] := 30;          { Anzahl der Cellen im Layer }
  for i:=13 to 28 do anz_l[i] := 60;

  n_l[1] := 2;                               { Benutze Messlagen }
  n_l[2] := 4;
  n_l[3] := 7;
  n_l[4] := 9;
  n_l[5] := 11;
  n_l[6] := 15;

  r_l[1] :=  0.22225;                        { Radien der Doppellagen }
  r_l[2] :=  0.2385;
  r_l[3] :=  0.25535;
  r_l[4] :=  0.27265;
  r_l[5] :=  0.29035;
  r_l[6] :=  0.3084;
  r_l[7] :=  0.3267;
  r_l[8] :=  0.3452;
  r_l[9] :=  0.36385;
  r_l[10] := 0.3828;
  r_l[11] := 0.40185;
  r_l[12] := 0.42095;

  r_l[13] := 0.549;                          { aussen }
  r_l[14] := 0.5659;
  r_l[15] := 0.58295;
  r_l[16] := 0.6002;
  r_l[17] := 0.6176;
  r_l[18] := 0.6350;
  r_l[19] := 0.6531;
  r_l[20] := 0.6710;
  r_l[21] := 0.6891;
  r_l[22] := 0.7073;
  r_l[23] := 0.7255;
```

```
  r_l[24] := 0.7440;
  r_l[25] := 0.7625;
  r_l[26] := 0.7820;
  r_l[27] := 0.7998;
  r_l[28] := 0.8185;


  phi_l[1] := -22.084;
  Phi_l[2] := -19.0450;                { Winkel der Zelle 0 }
  Phi_l[3] := -17.68;
  Phi_l[4] := -14.095;
  Phi_l[5] := -12.065;
  Phi_l[6] := -10.27;
  Phi_l[7] := -8.67;
  Phi_l[8] := -7.254;
  Phi_l[9] := -5.965;
  Phi_l[10] :=-4.81;
  Phi_l[11] :=-3.765;
  Phi_l[12] :=-2.818;

  Phi_l[13] :=-13.62;                        { aussen }
  Phi_l[14] :=-12.45;
  Phi_l[15] :=-11.35;
  Phi_l[16] :=-10.31;
  Phi_l[17] :=-9.34;
  Phi_l[18] :=-8.41;
  Phi_l[19] :=-7.45;
  Phi_l[20] :=-6.71;
  Phi_l[21] :=-5.93;
  Phi_l[22] :=-5.19;
  Phi_l[23] :=-4.478;
  Phi_l[24] :=-3.81;
  Phi_l[25] :=-3.166;
  Phi_l[26] :=-2.555;
  Phi_l[27] :=-1.98;
  Phi_l[28] :=-1.424;

  for i:=1 to 28 do phi_l[i] := phi_l[i] * pi / 180.0;

  for i:=1 to 28 do                          { Bins in den Messlagen }
    bins[i] := trunc( 2.0 * pi * r_l[i] / ( vd * clk * 2.0 * anz_l[i] ) ) + 1;

  for i:=1 to m_anz do
    writeln('==> Anzahl bins in Lage ==> ',n_l[i],' ==> ',bins[n_l[i]]);

  for i:=1 to ref_anz do begin               { Definition der Referenzpukte }
    ref_p[i].r    := r_l[i+1];
    ref_p[i].phi  := phi_l[i+1];
  end;

  for i:=1 to 10 do                        { setzt alle masken auf 0 }
    for j:=1 to 5 do
      for k:=1 to 6 do
        for l:=-4 to 4 do
          mask[i,j,k,l] := 0;

for i:=-5 to -1 do effcell[i] := i+30;     { effzell schliesst Kreis }
for i:=-5 to -1 do effcella[i] := i+60;

for i:=30 to 34 do effcell[i] := i-30;
for i:=60 to 64 do effcella[i] := i-60;

for i:=0 to 29 do effcell[i] := i;
for i:=0 to 59 do effcella[i] := i;
```

```
end; { End Set_Vars }

{$I mask.sim}
> procedure setmask;
>
> begin



>
>    mask[1,1,1,0] := 32768;
>    mask[1,1,2,0] := 2048;
>    mask[1,1,3,-1] := 7168;
>    mask[1,1,4,-1] := 61440;
>    mask[1,1,5,-1] := 49152;
>    mask[1,1,6,0] := 65024;
>
>    mask[1,2,1,0] := 32768;
>    mask[1,2,2,0] := 3072;
>    mask[1,2,3,-1] := 12288;
>    mask[1,2,4,-1] := 32768;
>    mask[1,2,5,-1] := 28672;
>    mask[1,2,6,-1] := 32767;
>    mask[1,2,6,-1] := 65535;
>
>    mask[1,3,1,0] := 32768;
>    mask[1,3,2,0] := 1024;
>    mask[1,3,3,-1] := 24576;
>    mask[1,3,4,-1] := 57344;
>    mask[1,3,5,-1] := 7936;
>    mask[1,3,6,-1] := 4095;
>    mask[1,3,6,-1] := 32767;
>
>    mask[1,4,1,0] := 32768;
>    mask[1,4,2,0] := 1024;
>    mask[1,4,3,-1] := 49152;
>    mask[1,4,4,-1] := 14336;
>    mask[1,4,5,-1] := 448;
>    mask[1,4,6,-2] := 63488;
>
>    mask[1,5,1,0] := 32768;
>    mask[1,5,2,0] := 1536;
>    mask[1,5,3,-1] := 32768;
>    mask[1,5,4,-1] := 3584;
>    mask[1,5,5,-1] := 511;
>    mask[1,5,5,-1] := 65535;
>    mask[1,5,6,-1] := 8191;
>    mask[1,5,6,-1] := 65535;
>
>
> { ********************************** Lage ************************}
>
>
> end;

{***************************  Trigger ******************************}


procedure triggern;

var
```

```
 i,j,k : integer;

begin
  for i:=0 to 29 do          { Loop ueber Cellen (der Referenzpunkte) }
   for k:=1 to 10 do         { Loop ueber Lagen der R.-Punkte }
     for j:=1 to 5 do        { Loop ueber die 5 Triggerstr. eines R.-Punktes }

   { Erst Oder in den Lagen,
     dann Verundung der Lagen,
     dann ( letzte Zeile Verundung mit Referenzpunkt }

       bits[i,j,k] :=

        (  ( not ((shift[effcell[i-5],n_l[1]] and mask[k,j,1,-5])  = 0 ) )
        or ( not ((shift[effcell[i-4],n_l[1]] and mask[k,j,1,-4])  = 0 ) )
        or ( not ((shift[effcell[i-3],n_l[1]] and mask[k,j,1,-3])  = 0 ) )
        or ( not ((shift[effcell[i-2],n_l[1]] and mask[k,j,1,-2])  = 0 ) )
        or ( not ((shift[effcell[i-1],n_l[1]] and mask[k,j,1,-1])  = 0 ) )
        or ( not ((shift[effcell[i  ],n_l[1]] and mask[k,j,1, 0])  = 0 ) )
        or ( not ((shift[effcell[i+1],n_l[1]] and mask[k,j,1,+1])  = 0 ) )




        or ( not ((shift[effcell[i+2],n_l[1]] and mask[k,j,1,+2])  = 0 ) )
        or ( not ((shift[effcell[i+3],n_l[1]] and mask[k,j,1,+3])  = 0 ) )
        or ( not ((shift[effcell[i+4],n_l[1]] and mask[k,j,1,+4])  = 0 ) )
        or ( not ((shift[effcell[i+5],n_l[1]] and mask[k,j,1,+5])  = 0 ) )  )

        and

        (  ( not ((shift[effcell[i-5],n_l[2]] and mask[k,j,2,-5])  = 0 ) )
        or ( not ((shift[effcell[i-4],n_l[2]] and mask[k,j,2,-4])  = 0 ) )
        or ( not ((shift[effcell[i-3],n_l[2]] and mask[k,j,2,-3])  = 0 ) )
        or ( not ((shift[effcell[i-2],n_l[2]] and mask[k,j,2,-2])  = 0 ) )
        or ( not ((shift[effcell[i-1],n_l[2]] and mask[k,j,2,-1])  = 0 ) )
        or ( not ((shift[effcell[i  ],n_l[2]] and mask[k,j,2, 0])  = 0 ) )
        or ( not ((shift[effcell[i+1],n_l[2]] and mask[k,j,2,+1])  = 0 ) )
        or ( not ((shift[effcell[i+2],n_l[2]] and mask[k,j,2,+2])  = 0 ) )
        or ( not ((shift[effcell[i+3],n_l[2]] and mask[k,j,2,+3])  = 0 ) )
        or ( not ((shift[effcell[i+4],n_l[2]] and mask[k,j,2,+4])  = 0 ) )
        or ( not ((shift[effcell[i+5],n_l[2]] and mask[k,j,2,+5])  = 0 ) )  )

        and

        (  ( not ((shift[effcell[i-5],n_l[3]] and mask[k,j,3,-5])  = 0 ) )
        or ( not ((shift[effcell[i-4],n_l[3]] and mask[k,j,3,-4])  = 0 ) )
        or ( not ((shift[effcell[i-3],n_l[3]] and mask[k,j,3,-3])  = 0 ) )
        or ( not ((shift[effcell[i-2],n_l[3]] and mask[k,j,3,-2])  = 0 ) )
        or ( not ((shift[effcell[i-1],n_l[3]] and mask[k,j,3,-1])  = 0 ) )
        or ( not ((shift[effcell[i  ],n_l[3]] and mask[k,j,3, 0])  = 0 ) )
        or ( not ((shift[effcell[i+1],n_l[3]] and mask[k,j,3,+1])  = 0 ) )
        or ( not ((shift[effcell[i+2],n_l[3]] and mask[k,j,3,+2])  = 0 ) )
        or ( not ((shift[effcell[i+3],n_l[3]] and mask[k,j,3,+3])  = 0 ) )
        or ( not ((shift[effcell[i+4],n_l[3]] and mask[k,j,3,+4])  = 0 ) )
        or ( not ((shift[effcell[i+5],n_l[3]] and mask[k,j,3,+5])  = 0 ) )  )

{        and
```

```
        (  ( not ((shift[effcell[i-5],n_1[4]] and mask[k,j,4,-5])  = 0 ) )
        or ( not ((shift[effcell[i-4],n_1[4]] and mask[k,j,4,-4])  = 0 ) )
        or ( not ((shift[effcell[i-3],n_1[4]] and mask[k,j,4,-3])  = 0 ) )
        or ( not ((shift[effcell[i-2],n_1[4]] and mask[k,j,4,-2])  = 0 ) )
        or ( not ((shift[effcell[i-1],n_1[4]] and mask[k,j,4,-1])  = 0 ) )
        or ( not ((shift[effcell[i  ],n_1[4]] and mask[k,j,4, 0])  = 0 ) )
        or ( not ((shift[effcell[i+1],n_1[4]] and mask[k,j,4,+1])  = 0 ) )
        or ( not ((shift[effcell[i+2],n_1[4]] and mask[k,j,4,+2])  = 0 ) )
        or ( not ((shift[effcell[i+3],n_1[4]] and mask[k,j,4,+3])  = 0 ) )
        or ( not ((shift[effcell[i+4],n_1[4]] and mask[k,j,4,+4])  = 0 ) )
        or ( not ((shift[effcell[i+5],n_1[4]] and mask[k,j,4,+5])  = 0 ) )  )

        and

        (  ( not ((shift[effcell[i-5],n_1[5]] and mask[k,j,5,-5])  = 0 ) )
        or ( not ((shift[effcell[i-4],n_1[5]] and mask[k,j,5,-4])  = 0 ) )
        or ( not ((shift[effcell[i-3],n_1[5]] and mask[k,j,5,-3])  = 0 ) )
        or ( not ((shift[effcell[i-2],n_1[5]] and mask[k,j,5,-2])  = 0 ) )
        or ( not ((shift[effcell[i-1],n_1[5]] and mask[k,j,5,-1])  = 0 ) )
        or ( not ((shift[effcell[i  ],n_1[5]] and mask[k,j,5, 0])  = 0 ) )
        or ( not ((shift[effcell[i+1],n_1[5]] and mask[k,j,5,+1])  = 0 ) )
        or ( not ((shift[effcell[i+2],n_1[5]] and mask[k,j,5,+2])  = 0 ) )
        or ( not ((shift[effcell[i+3],n_1[5]] and mask[k,j,5,+3])  = 0 ) )
        or ( not ((shift[effcell[i+4],n_1[5]] and mask[k,j,5,+4])  = 0 ) )
        or ( not ((shift[effcell[i+5],n_1[5]] and mask[k,j,5,+5])  = 0 ) )  )

        and

          (  ( not ((shift[effcella[i-5],n_1[6]] and mask[k,j,6,-5])  = 0 ) )
        or ( not ((shift[effcella[i-4],n_1[6]] and mask[k,j,6,-4])  = 0 ) )
        or ( not ((shift[effcella[i-3],n_1[6]] and mask[k,j,6,-3])  = 0 ) )
        or ( not ((shift[effcella[i-2],n_1[6]] and mask[k,j,6,-2])  = 0 ) )
        or ( not ((shift[effcella[i-1],n_1[6]] and mask[k,j,6,-1])  = 0 ) )
        or ( not ((shift[effcella[i  ],n_1[6]] and mask[k,j,6, 0])  = 0 ) )
        or ( not ((shift[effcella[i+1],n_1[6]] and mask[k,j,6,+1])  = 0 ) )
        or ( not ((shift[effcella[i+2],n_1[6]] and mask[k,j,6,+2])  = 0 ) )
        or ( not ((shift[effcella[i+3],n_1[6]] and mask[k,j,6,+3])  = 0 ) )
        or ( not ((shift[effcella[i+4],n_1[6]] and mask[k,j,6,+4])  = 0 ) )
        or ( not ((shift[effcella[i+5],n_1[6]] and mask[k,j,6,+5])  = 0 ) ) )




        and ( not ((shift[effcell[i],j] and $8000) = 0 ))}

        ;
end;



procedure analyze;  { Multiplizitaet in Triggerbits }

var
  i,j,k : integer;
  m     : integer;

begin
```

```
  m := 0;
  for i:=0 to 29 do
   for j:=1 to 5 do
     for k:=1 to 10 do
       if bits[i,j,k] then m:=m+1;

end;



procedure Triggerloop;

var

  lastevent         : boolean;
  event_anz         : integer;
  i,j,l             : integer;
  spur              : track;
  GraphDriver,
  GraphMode,
  ErrorCode,
  maxx,maxy         : integer;
  ch                : char;
  dxy,zw,r,phi      : real;
  zoomx,zoomy       : real;
  xoffset,yoffset   : real;
  asx,asy           : word;


  procedure makepixel(r,phi   :real;
                      farbe   : integer);
  var
    x,y     : real;
    ix,iy   : integer;
  begin
     x      := r * cos(phi);
     y      := r * sin(phi);
     ix     := (maxx div 2) + trunc ( (maxx div 2) * x / 0.85 );
     iy     := (maxy div 2) - trunc ( (maxy div 2) * y / 0.85 );
     ix     := trunc( (ix + xoffset) * zoomx * dxy );
     iy     := trunc( (iy + yoffset) * zoomy);
    putpixel(ix,iy,farbe);
  end;


  procedure makepoint(x,y   :real;
                      farbe   : integer);
  var
    ix,iy   : integer;
  begin
     ix     := (maxx div 2) + trunc ( (maxx div 2) * x / 0.85 );
     iy     := (maxy div 2) - trunc ( (maxy div 2) * y / 0.85 );
     ix     := trunc( (ix + xoffset) * zoomx * dxy );
     iy     := trunc( (iy + yoffset) * zoomy);




    putpixel(ix,iy,farbe);
  end;
```

```
  procedure ClearRegister;
  var
    i,j,k,l    : integer;
  begin
    for i:=0 to 59 do
      for j:=1 to 28 do shift[i,j] := 0;
    for i:=0 to 29 do
      for j:=1 to 5 do
        for k := 1 to 10 do
           bits[i,j,k] := false;
{    for i:=1 to 10 do
      for j:=1 to 5 do
        for k := 1 to 6 do
          for l:=-5 to 5 do
            mask[i,j,k,l] := $FFFF;
 }
  end;  { End ClearRegister }


  procedure displaytrack(r,vx,vy:real);
  var
    i       : integer;
    x,y     : real;
    xl,xh   : real;
    dx      : real;

  begin
    xl := - r + vx;
    xh :=   r + vx;
    dx := abs(xl-xh) / 500.0;
    x := xl;
    for i:=1 to 500 do begin
      y := sqrt( sqr(r) - sqr(x - vx) ) + vy;
      makepoint(x,y,lightred);
      y := -sqrt( sqr(r) - sqr(x - vx) ) + vy;
      makepoint(x,y,lightred);
      x:=x+dx;
    end;
  end;


  procedure displayshift;
  var
    i,j,m,b     : integer;
    w           : word;
    dphi        : real;
    phi         : real;
  begin
    writeln(outf);
    for i:=0 to 59 do begin
      for j:=1 to 28 do begin
        w := shift[i,j];
        if w<>0 then begin
          writeln(outf,'shift[',i,',',j,'] = ',w);
          for m:=0 to 15 do begin
            if bitset(w,m) then begin
              b := 15-m;

              dphi :=  (2.0 * pi / ( 2.0 * anz_l[j] * bins[j])) * b;
              phi := phi_l[j] + (i * ( 2.0 * pi / anz_l[j]));
{              makepixel(r_l[j],phi,lightred);    { Display Wire Position }
```

```
              phi := phi + dphi;
              makepixel(r_l[j],phi,lightmagenta); { Display Triggerbit
Position
              phi := phi - (2*dphi);
              makepixel(r_l[j],phi,lightcyan);    { Display Triggerbit
Position

            end; { end if }




          end; { end for }
        end; { end if }
      end; { end for }
    end; { end for }
  end; { end displayshift }

{ ******************** Schnittpunkt ********************************** }

function SchnittPunkt(rl,rt,r0,phi0 : real;  l : integer) : real;
{ Diese Function liefert den Schnittpunkt von zwei Kreisen in
  Polarkoordinaten. }
begin
  if ((r0 < rl+rt) and (rt<r0+rl)) then
    if l=1 then
      Schnittpunkt := arccos( ( sqr(r0) + sqr(rl) - sqr(rt) ) / ( 2 * r0 * rl
)
    else
      Schnittpunkt := -arccos( ( sqr(r0) + sqr(rl) - sqr(rt) ) / ( 2 * r0 * rl
)
  else
    Schnittpunkt := -100;
end;



  procedure GetTrack(var spur:track);
  { Teta <> 0 oder pi !!! }
  begin
    spur.x0     := 0.0;
    spur.y0     := 0.0;
    spur.z0     := 0.0;
    spur.r      := 2.0;
    spur.phi    := 0.57;
    spur.teta   := pi/2;
    spur.x1     := 0.0;
    spur.y1     := 0.0;
    spur.z1     := 0.0;

  end; { End GetEvent }


{ ******************** Readout Track ******************************** }

  procedure ReadoutTrack(spur:Track);

  var
    sign,im,layer,cell,bit            : integer;
```

```
    x,y,vx,vy,vr,r,r0                  : real;
    phi,phi0,phi1,phi2,dphi,vphi       : real;

    rz,vz,teta                         : real;  { Erweiterungen z - Koordinate
}

  begin

    vx    := spur.x0;                          { Vertex der Spur x - Koordinate }
    vy    := spur.y0;                          { Vertex der Spur y - Koordinate }
    vz    := spur.z0;                          { Vertex der Spur z - Koordinate }
    vphi  := angle(vx,vy);                     { Winkel vom Vertex }
    vr    := sqrt( sqr(vx) + sqr(vy) );        { Radius vom Vertex }
    r     := abs(spur.r);                      { Radius der Spur }
    if abs(spur.r) = spur.r then               { Ladung der Spur }
      sign  := 1
    else
      sign  := -1;
    phi   := spur.phi - (pi /2);               { senkrechte zum Tangentenwinkel r
-
    teta  := spur.teta;                        { Winkel Track - z - Achse }
    x     := sign * r * cos(phi) + vx;         { Mittelpunktkoordinaten des
Tracks
    y     := sign * r * sin(phi) + vy;
{    makepoint(x,y,white);                     { Display Mittelpunkt }
    r0    := sqrt( sqr(x) + sqr(y) );          { Radius von Mittelpunkt }
    phi0  := angle(x,y);                       { Winkel von Mittelpunkt }




    displaytrack(r,x,y);                      { Display Track }

    for im := 1 to l_anz do begin              { loop ueber alle Lagen, eine Spur
}
                                               { 28 }
      phi := SchnittPunkt(r_l[im],r,r0,phi0,sign);
                                               { liefert Schnittpunkt der Kreise
}

      if (phi <> -100) then begin              { Abfrage ob Schittpunkt vorhanden
}
        if phi < 0 then phi := 2*pi+phi;       { Kreisschluss }
        rz := vz + r_l[im] * cot(teta);        { z - Koordnate in layer }
        if ((rz<1.1) and (rz>-1.1)) then begin
          phi2 := pi / anz_l[im];                { Winkel einer halben celle }

          dphi := abs(phi - phi_l[im]);
          if dphi>2*pi then dphi := dphi - 2*pi;

          cell := trunc ( (dphi + phi2) / ( 2 * phi2 ) );
                                               { Cell# of track in layer im}
          bit  := trunc( ( dphi  -  cell * 2 * phi2) * r_l[im]
                       / ( vd * clk ) ) ;  { Bit# of track in cell }
          shift[cell,im] := setbit(shift[cell,im],15-abs(bit));
                                               { Set Bit in Shift }
        end; { End if rz }
      end; { End if phi }
    end;  { End for im }

  end;  { End ReadoutTrack }
```

```
  procedure ShiftEvent;
  var
    i,j : integer;
  begin
    for i:=0 to 59 do
      for j:=1 to 28 do
        shift[i,j] := shift[i,j] shl 1;
  end;


{ ************************* Triggerloop ***************************** }

begin  { Begin TriggerLoop }

  zoomx   :=  1.40;
  zoomy   :=  1.40;
  xoffset :=  -50.0;
  yoffset :=  0.0;

  GraphDriver := Detect;
  InitGraph (GraphDriver,GraphMode,'');
  ErrorCode := GraphResult;
  if ErrorCode <> grOK then
  begin
    writeln ('Grafikfehler');
    Halt(1)
  end;
  maxx := getmaxx;
  maxy := getmaxy;
  getaspectratio(asx,asy);
  dxy := (maxy / maxx) * (asy/asx);
  ClearDevice;

                                                { Koordinatensystem }
  for i:=-maxx to maxx do makepoint(i*(0.85/maxx),0,white);
  for i:=-maxy to maxy do makepoint(0,i*(0.85/maxy),white);

                                                { Sensdraehte zeichnen }
  for i:=1 to 28 do begin
    for j:=1 to anz_l[i] do begin
      r := r_l[i];
      phi := phi_l[i] + j * ( 2.0 * pi / anz_l[i]);
      makepixel(r,phi,white);




    end;
  end; { End for }


  for i:=1 to 28 do begin
    r := r_l[i];
    phi := phi_l[i];
    makepixel(r,phi,yellow);
  end; { End for }

  lastevent := false;
  event_anz := 1;
```

```
  repeat
    ClearRegister;
    GetTrack(spur);
    ReadoutTrack(spur);
    displayshift;

    for l := 1 to 15 do begin
      triggern;
      analyze;
      ShiftEvent;
      displayshift;
    end;
    event_anz := event_anz - 1;
    if event_anz = 0 then lastevent:= true;
  until LastEvent; { Main Triggerloop }

  ch := readkey;
  ClearDevice;
  CloseGraph;
  RestoreCrtMode;

end; { Triggerloop }


{ ********************** MainProgramm ****************************** }


begin { Hauptprogramm }
  assign(monout,'con');
  rewrite(monout);
  assign(monin,'con');
  reset(monin);
  assign(outf,'trigger.log');
  rewrite(outf);
  clrscr;
  writeln('==> running');
  set_vars;
  setmask;
  TriggerLoop;
  close(outf);
  close(monout);
  close(monin);
end.
```

D Schaltpläne und Abbildungen

D.1 Schaltplan einer Triggerkarte

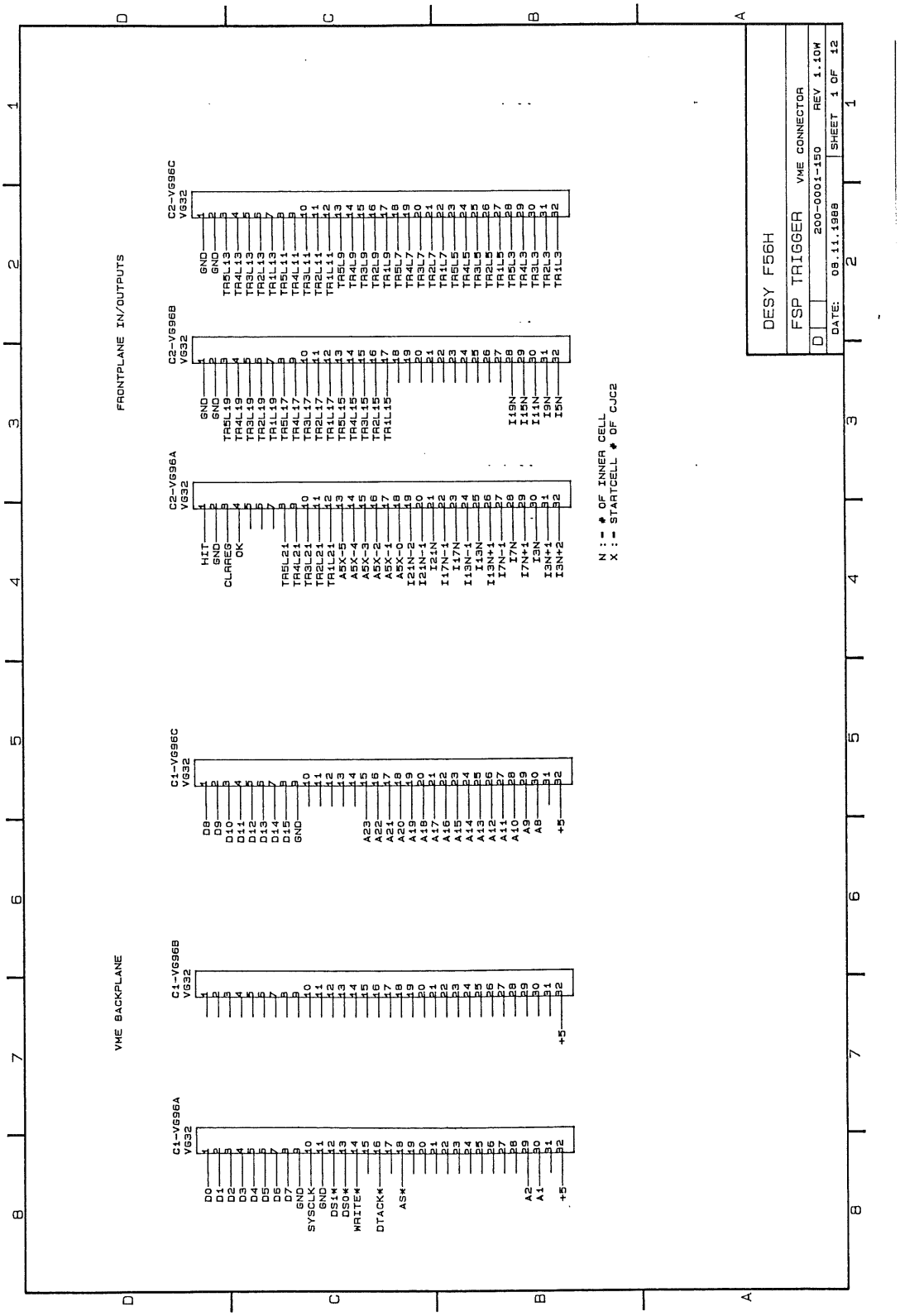

Abbildung 39: Schaltplan der Triggerkarte 1 von 12

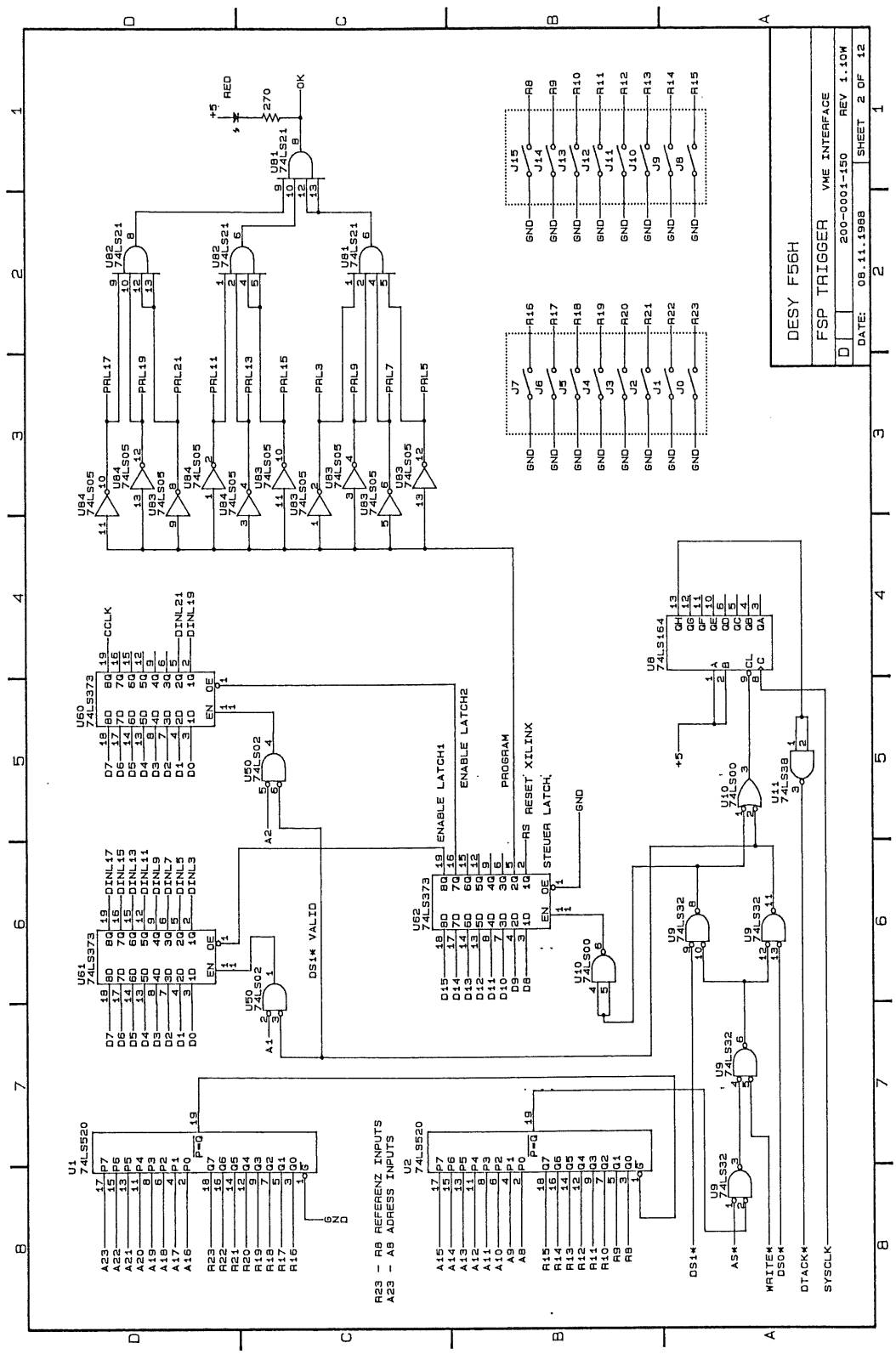

Abbildung 40: Schaltplan der Triggerkarte 2 von 12

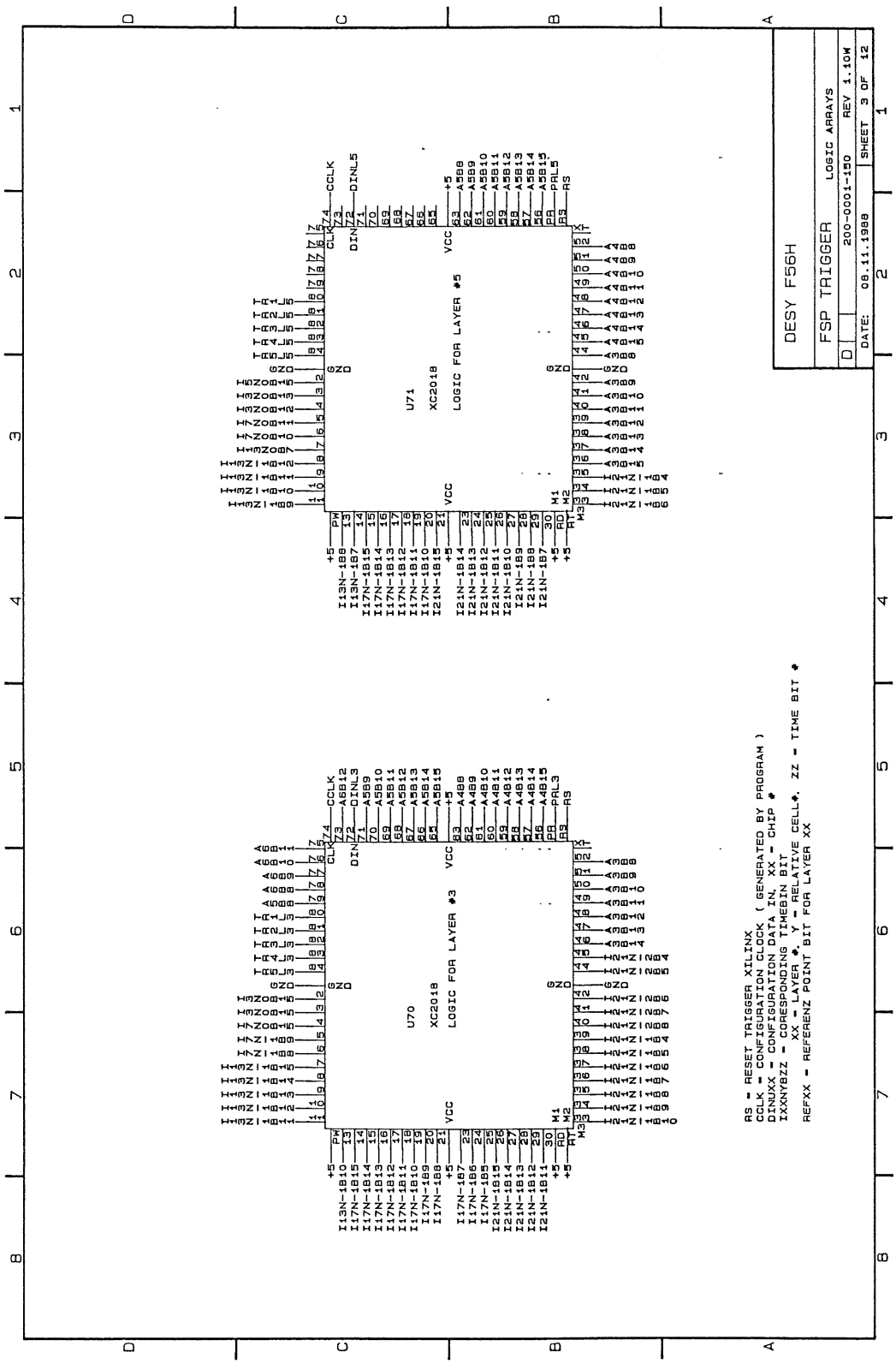

Abbildung 41: Schaltplan der Triggerkarte 3 von 12

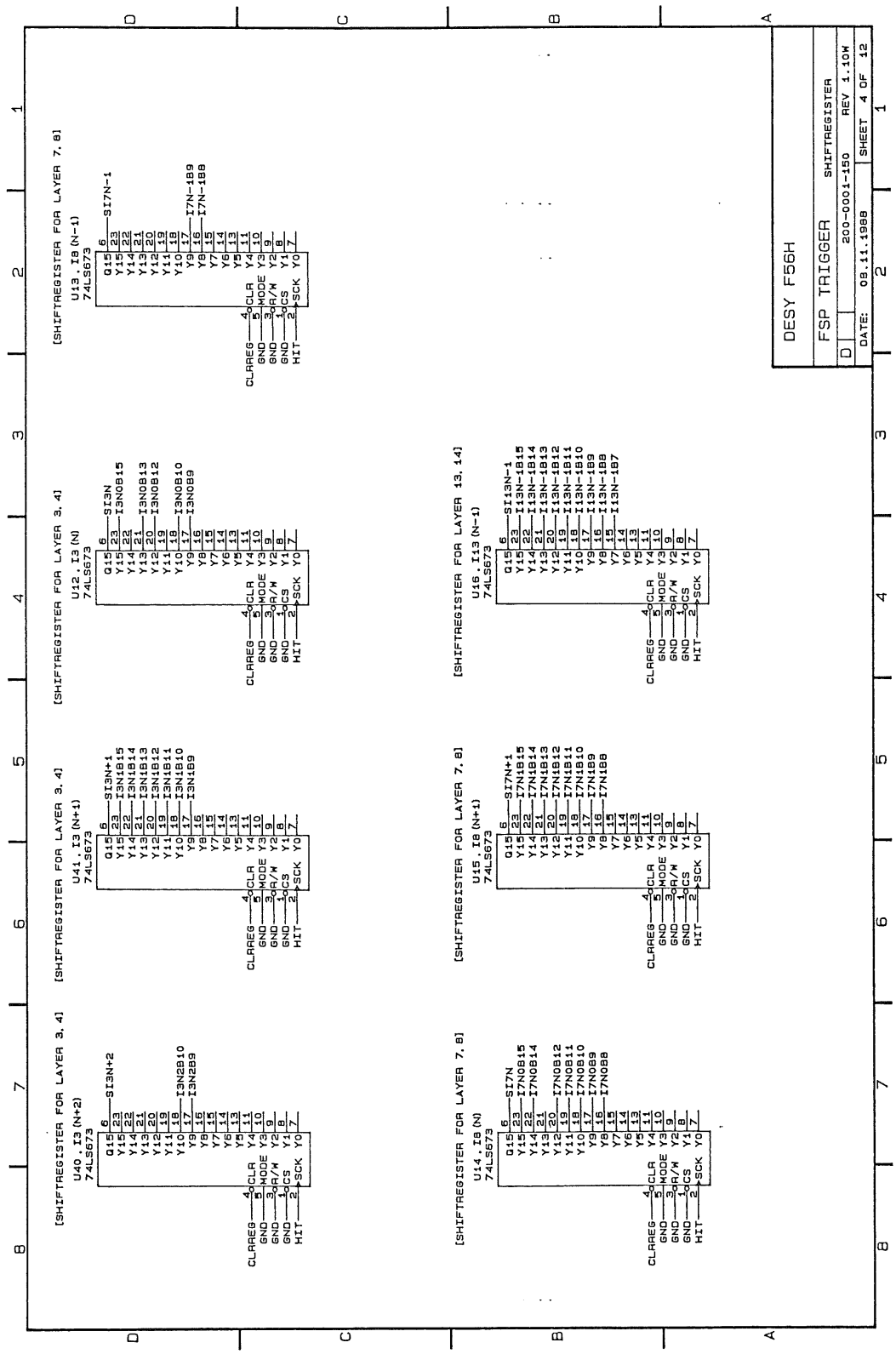

Abbildung 42: Schaltplan der Triggerkarte 4 von 12

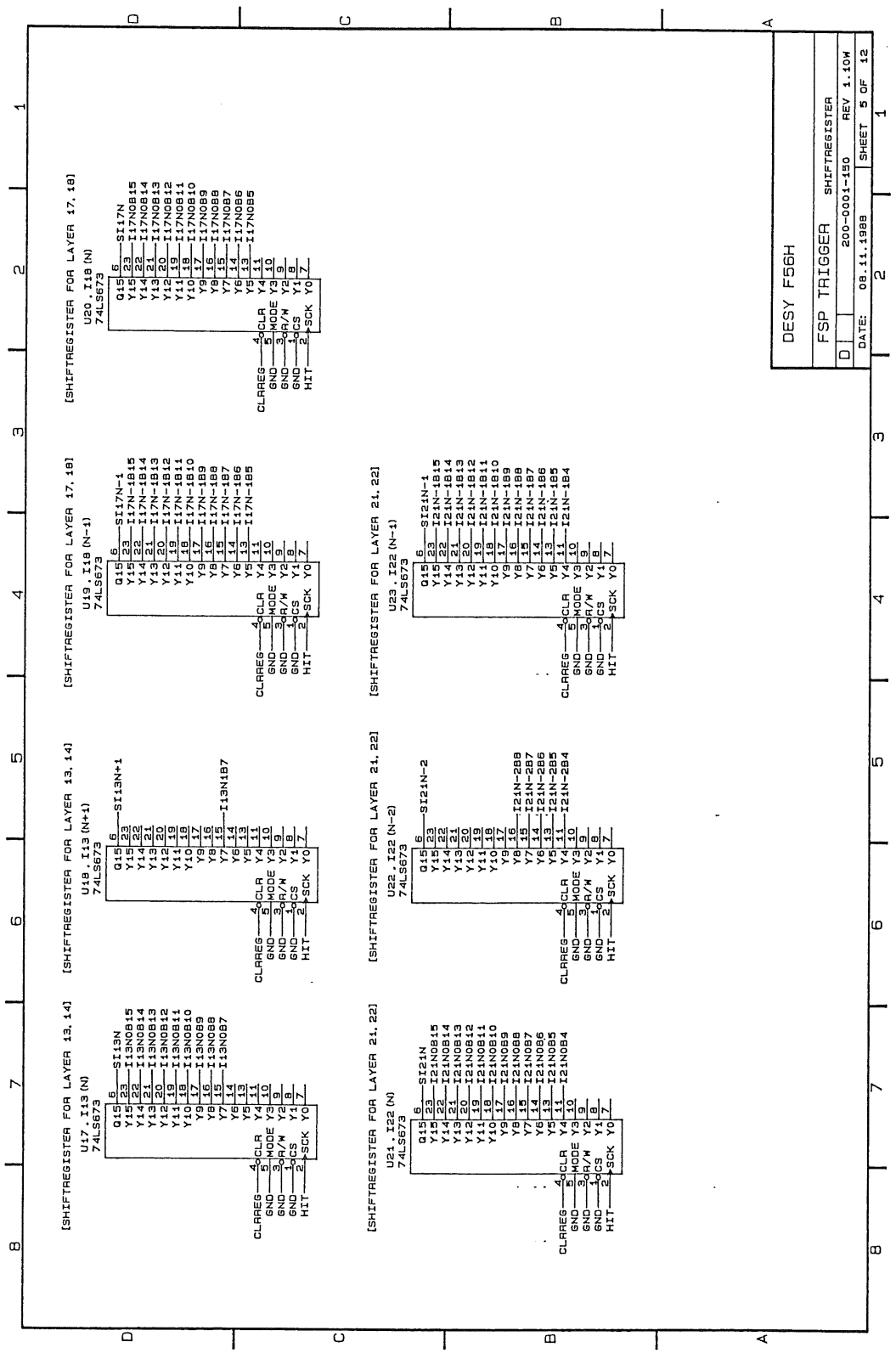

Abbildung 43: Schaltplan der Triggerkarte 5 von 12

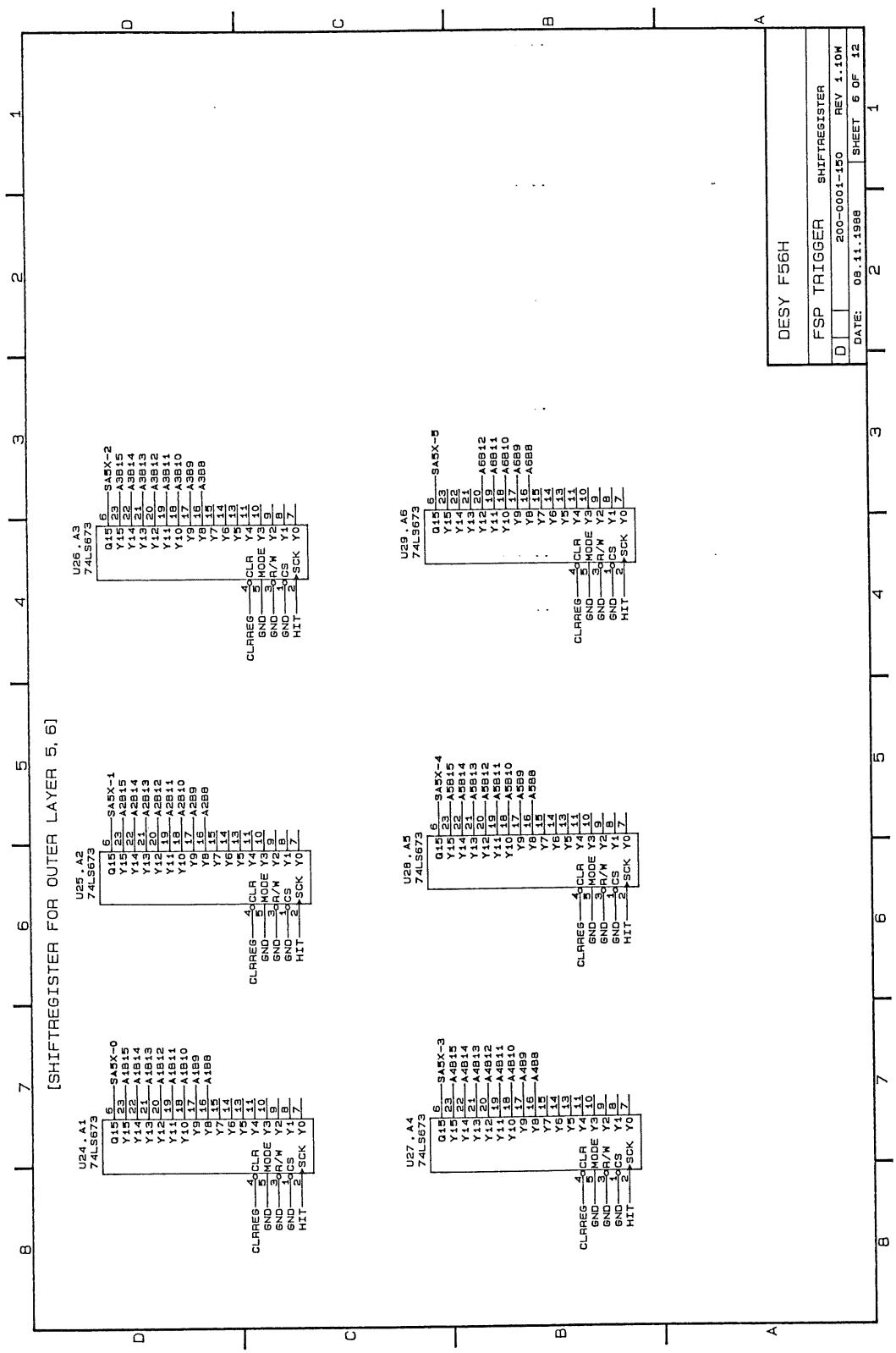

Abbildung 44: Schaltplan der Triggerkarte 6 von 12

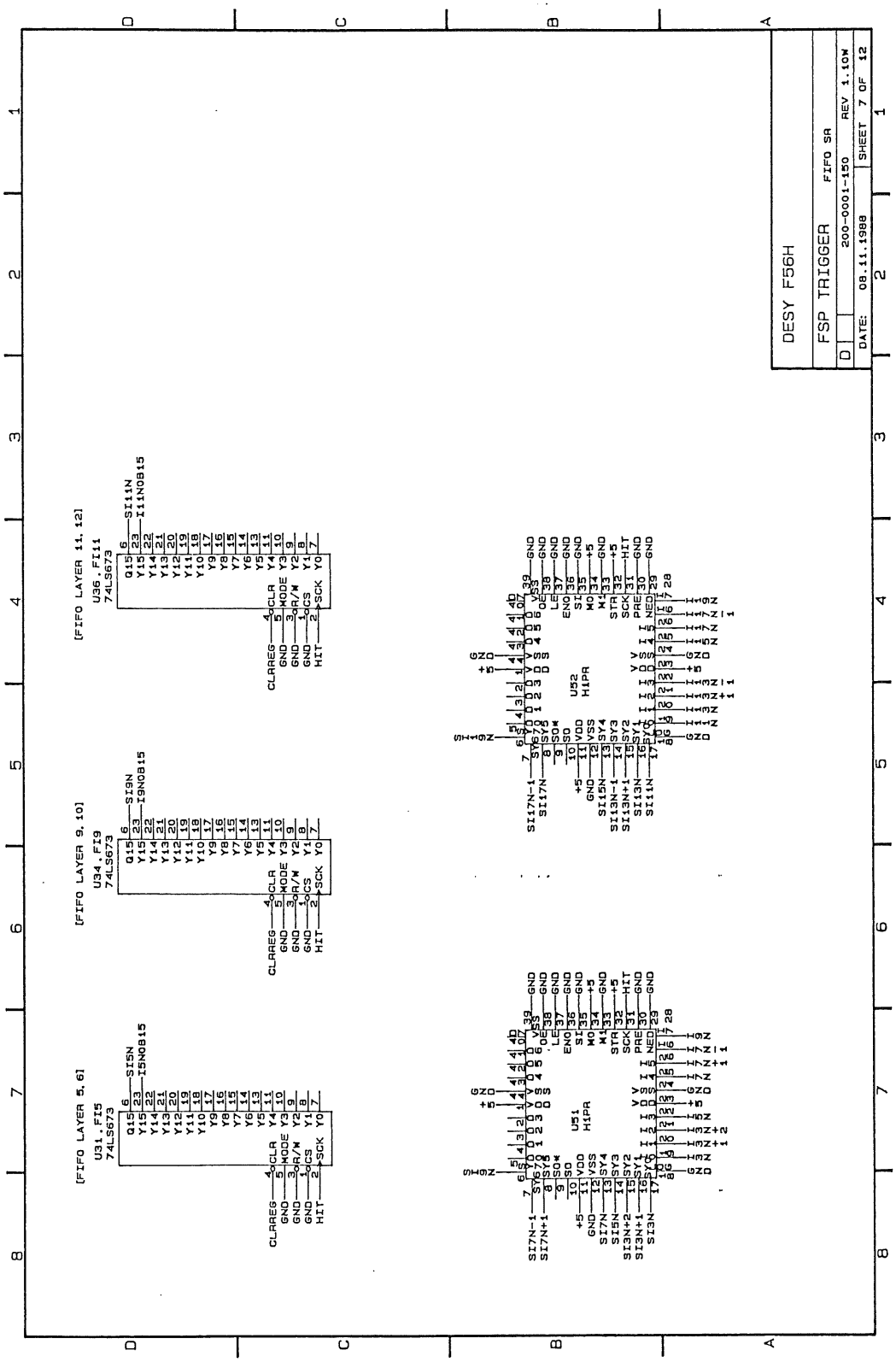

Abbildung 45: Schaltplan der Triggerkarte 7 von 12

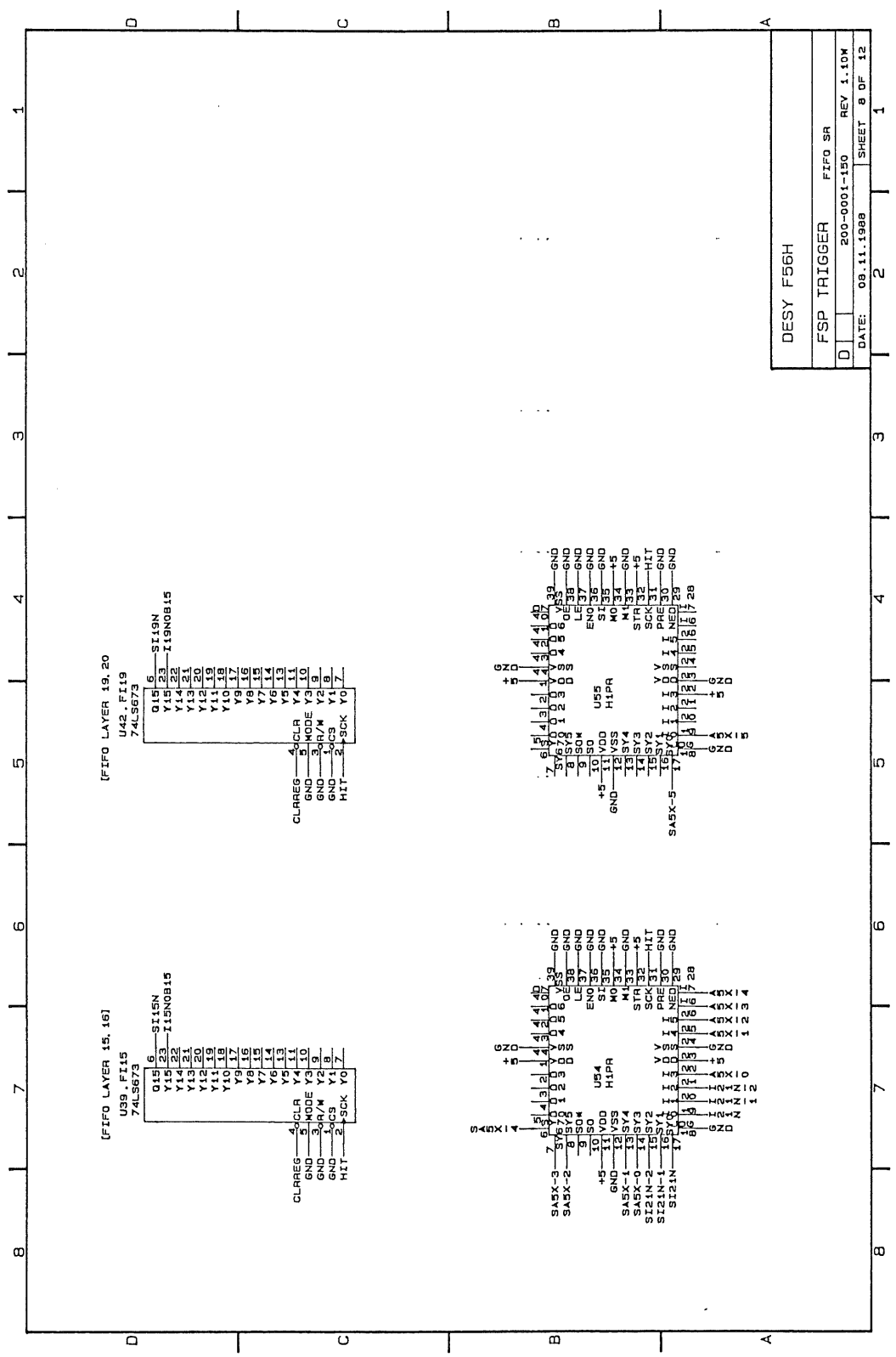

Abbildung 46: Schaltplan der Triggerkarte 8 von 12

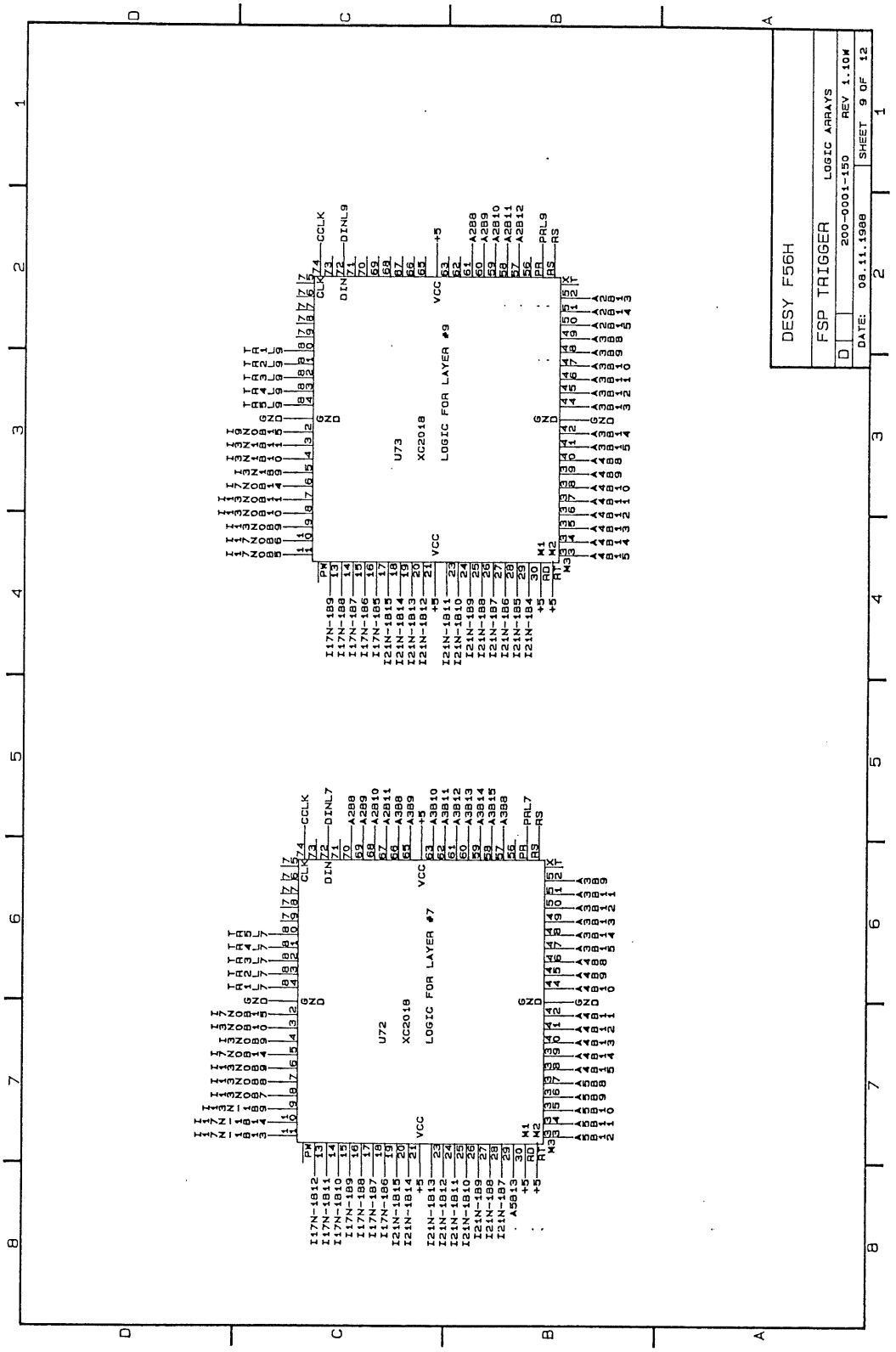

Abbildung 47: Schaltplan der Triggerkarte 9 von 12

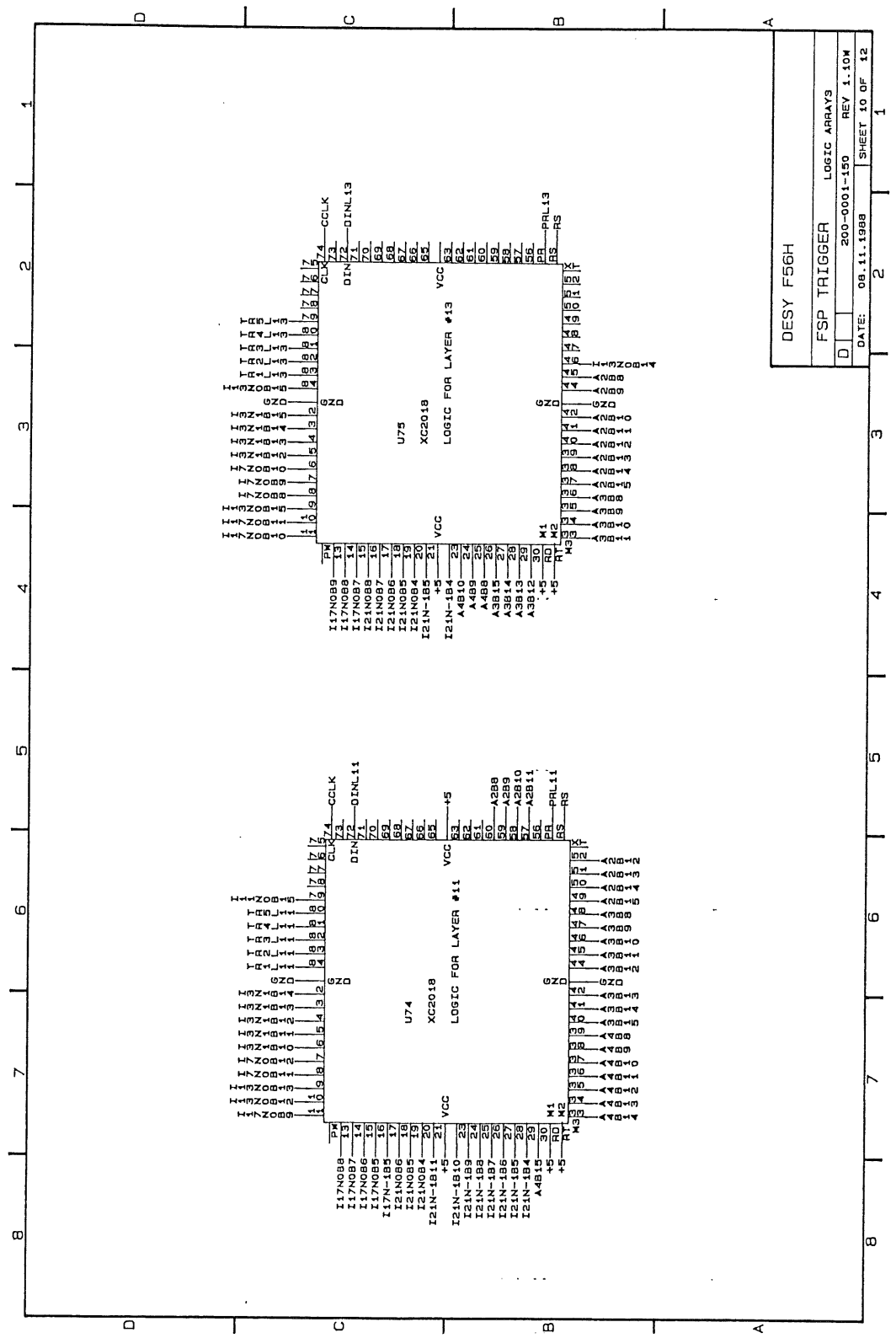

Abbildung 48: Schaltplan der Triggerkarte 10 von 12

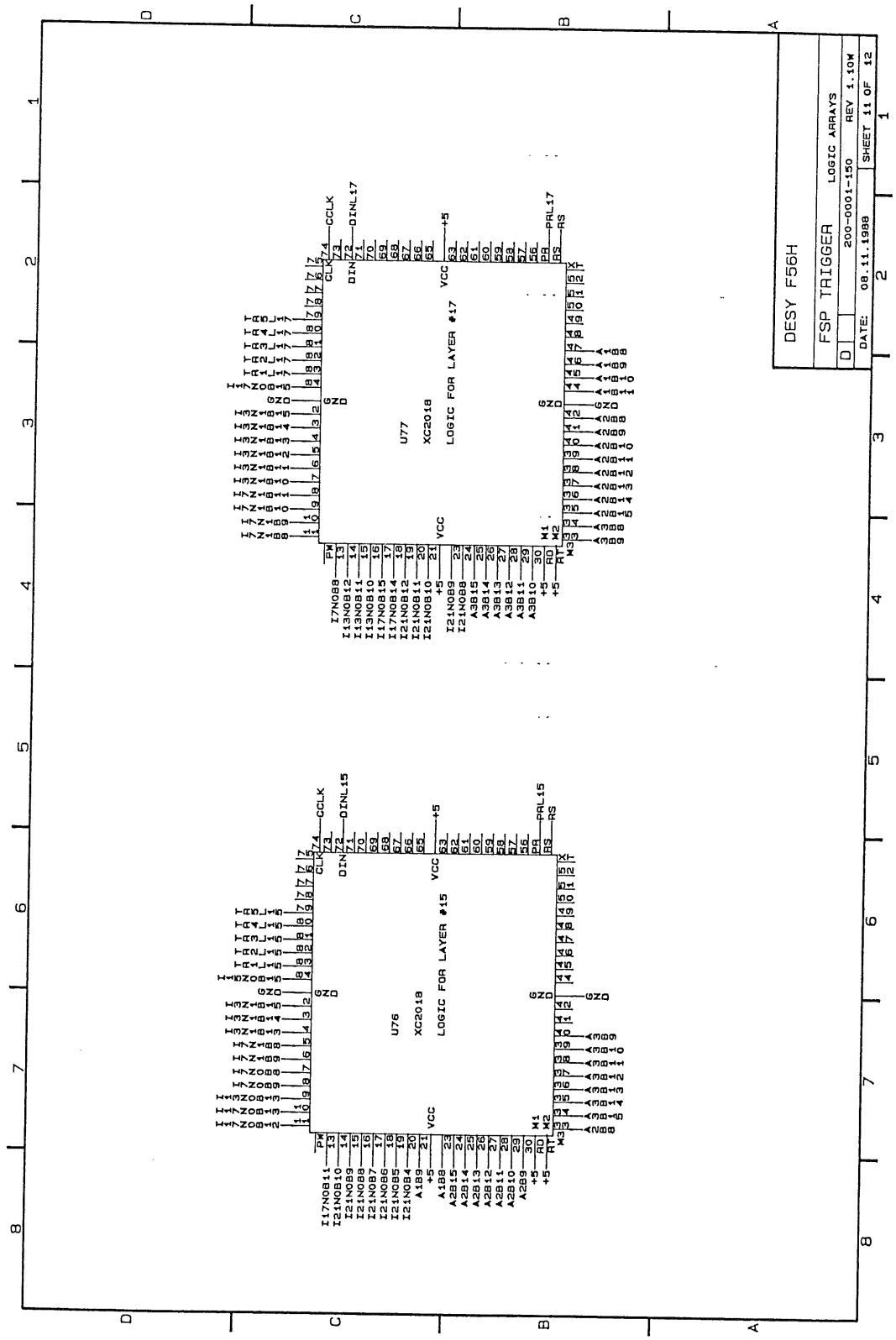

Abbildung 49: Schaltplan der Triggerkarte 11 von 12

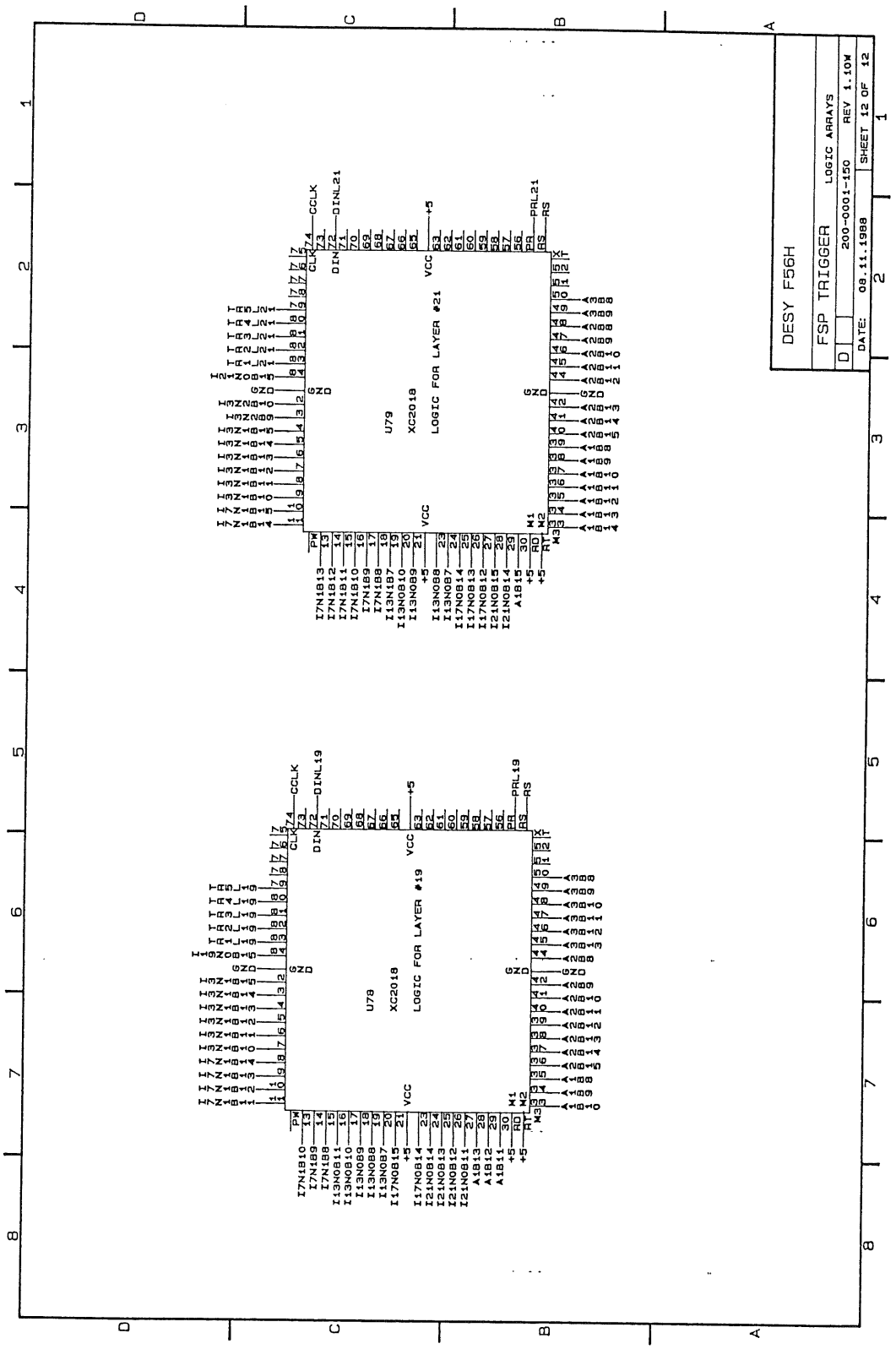

Abbildung 50: Schaltplan der Triggerkarte 12 von 12

D.2 Schaltplan der Testkarte

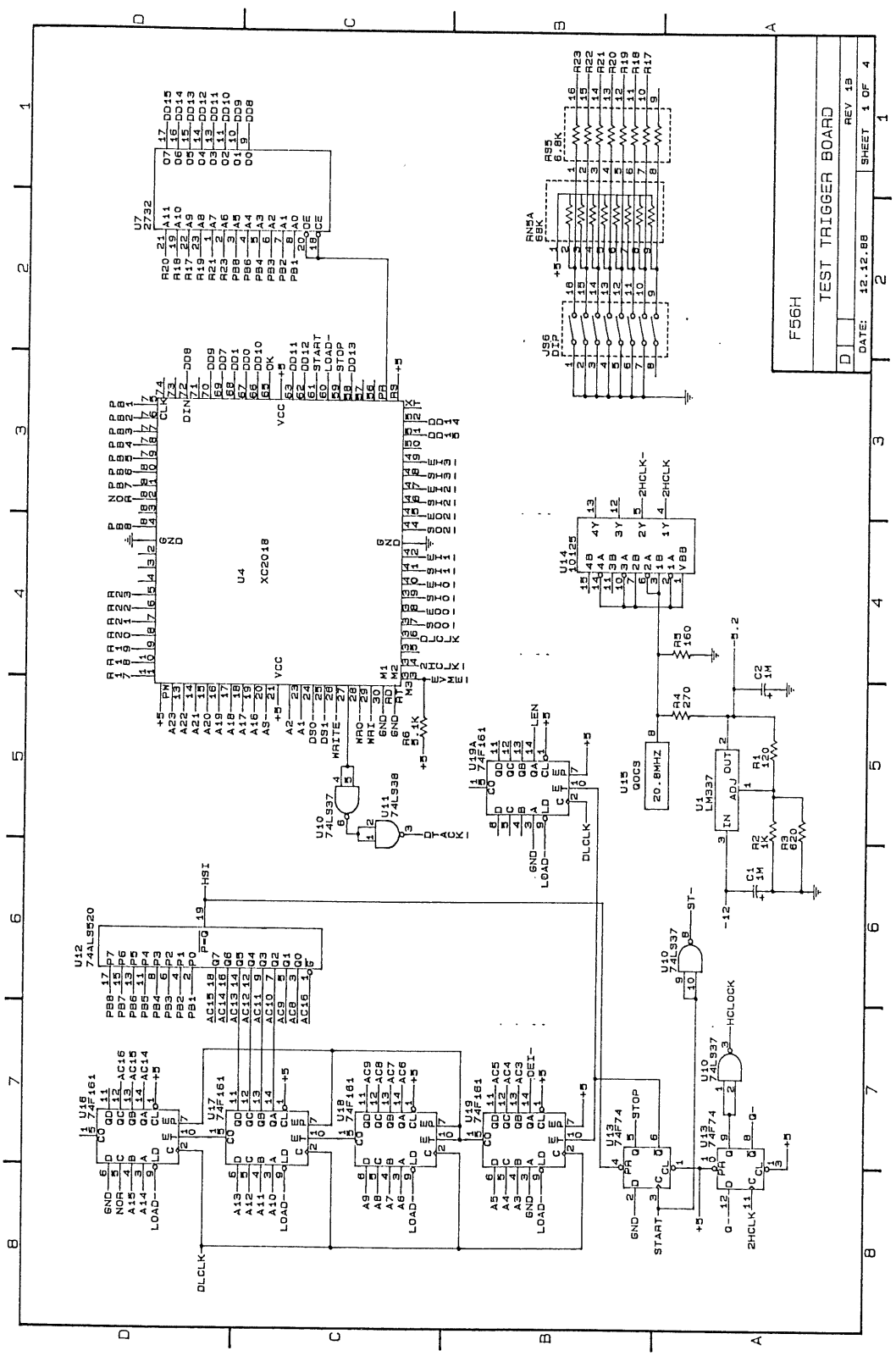

Abbildung 51: Schaltplan der Testkarte 1 von 3

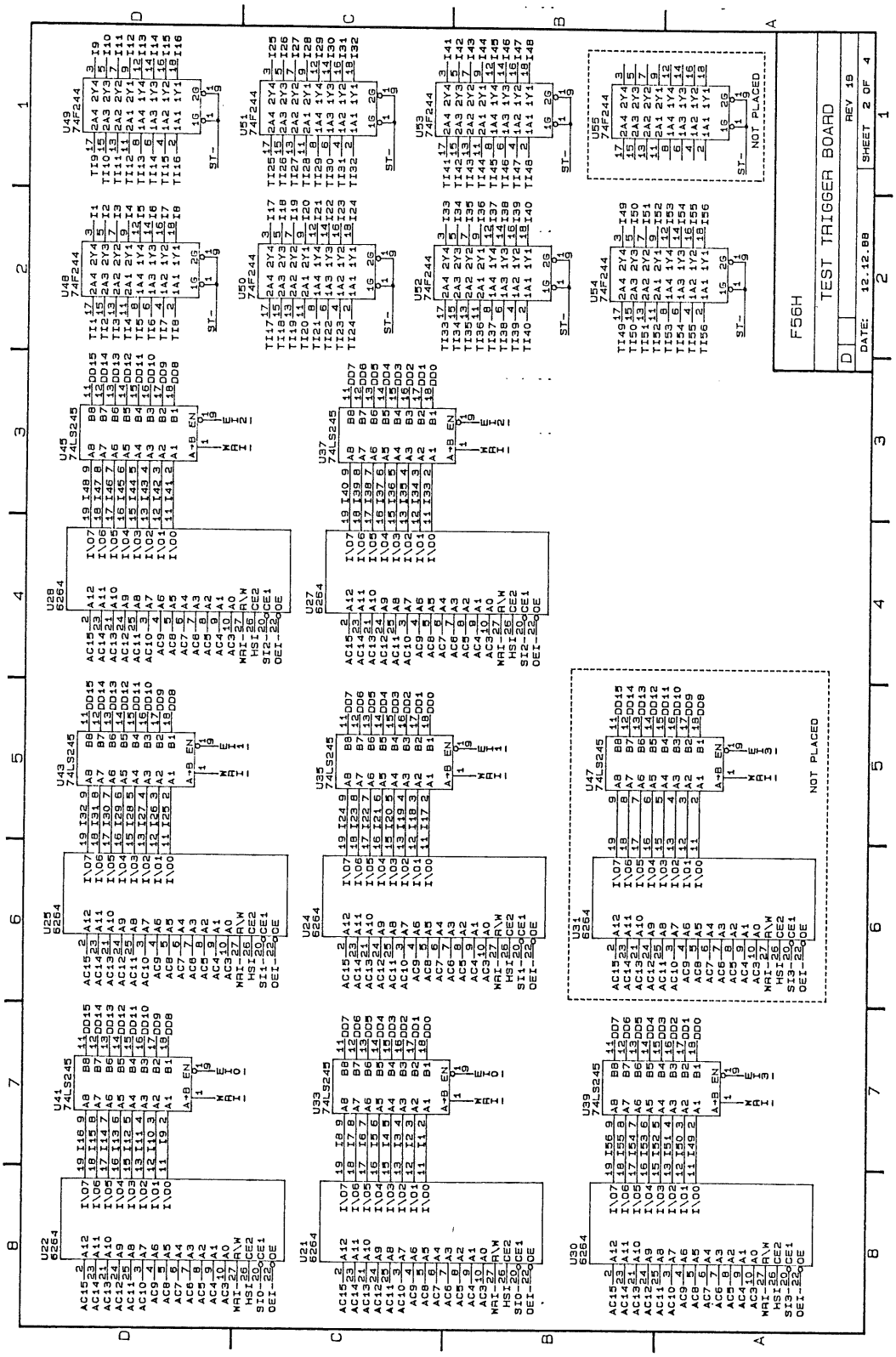

Abbildung 52: Schaltplan der Testkarte 2 von 3

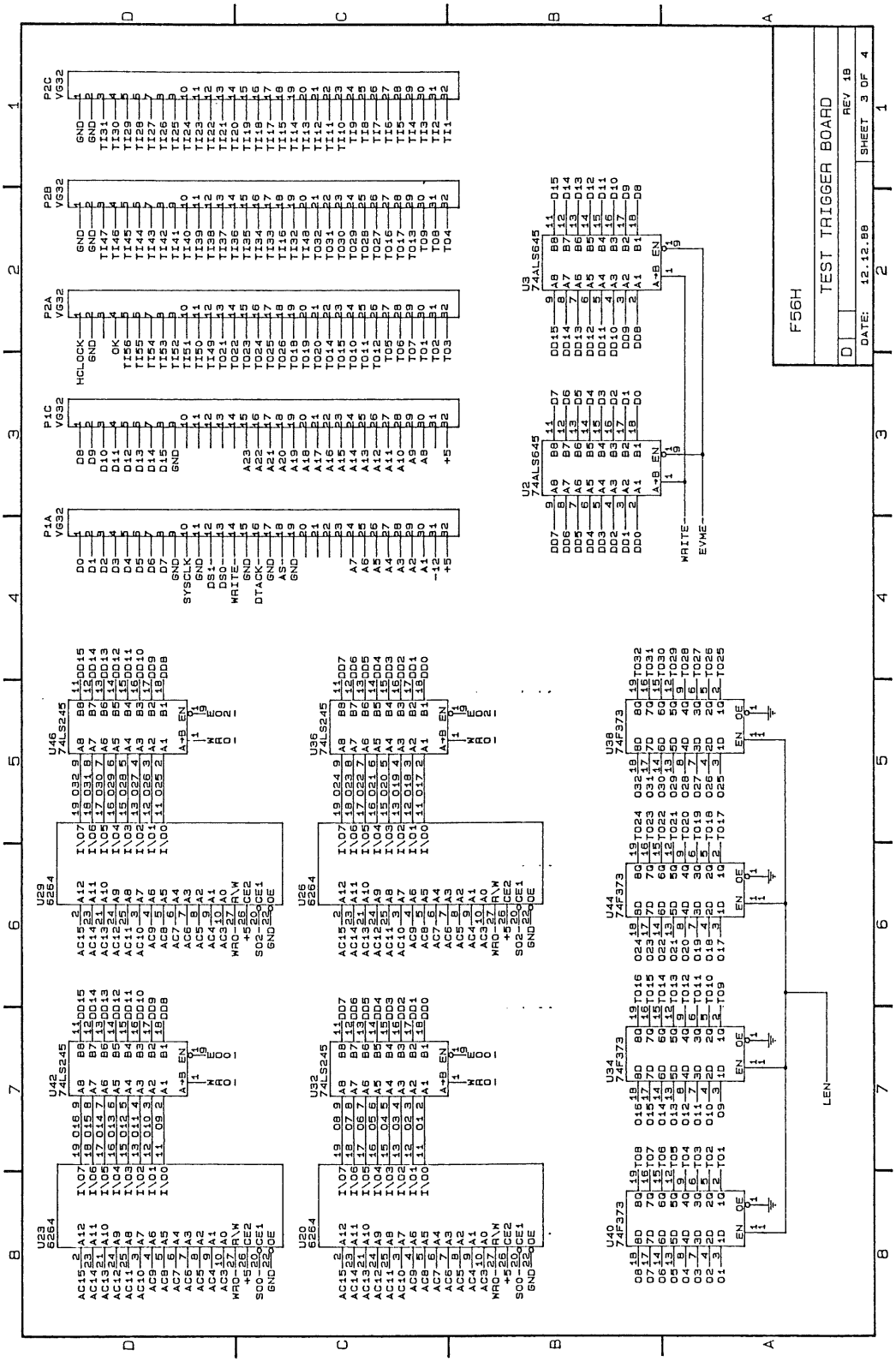

Abbildung 53: Schaltplan der Testkarte 3 von 3

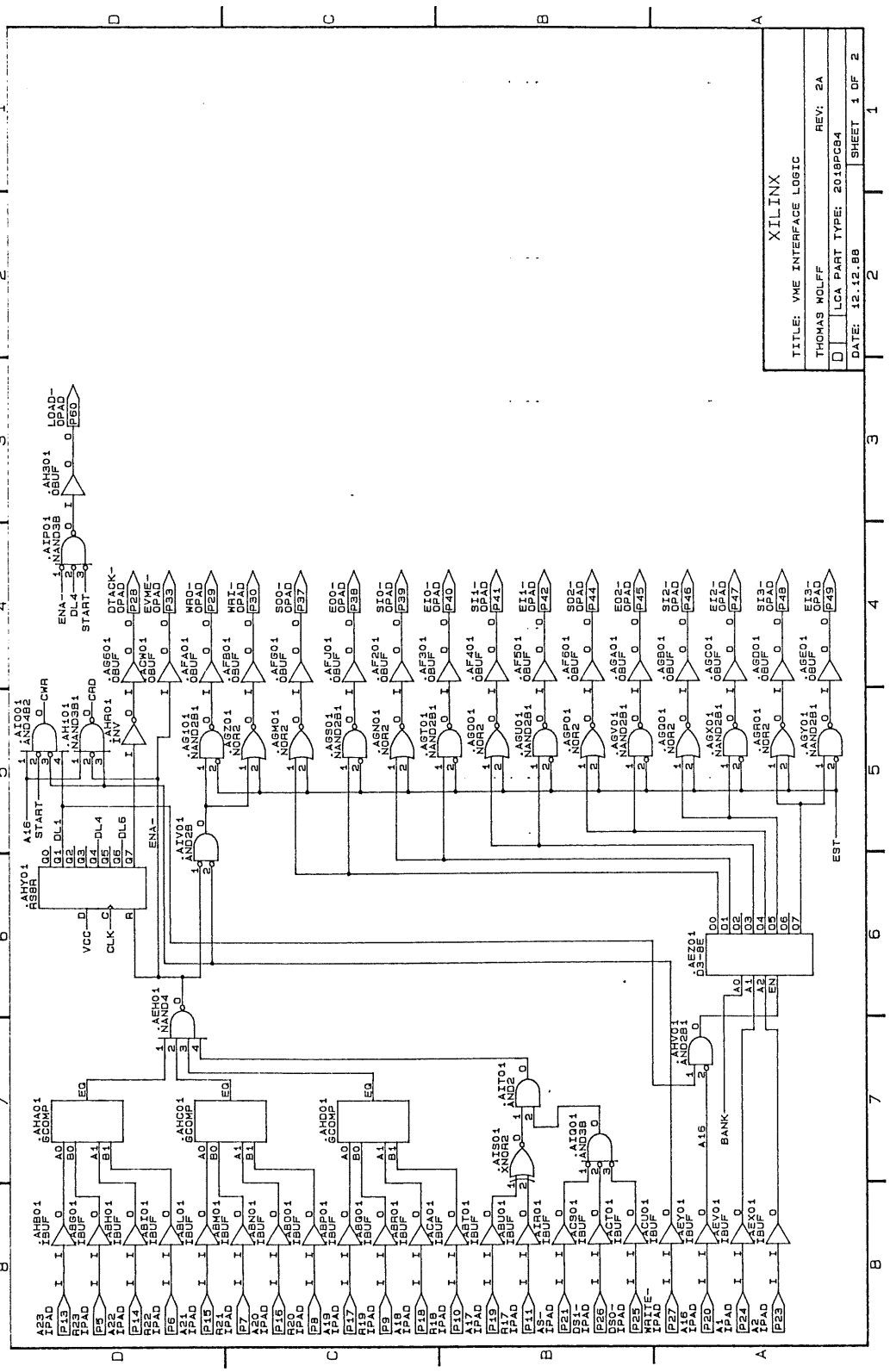

Abbildung 54: Xilinx–Diagramm des VME–Interfaces auf der Testkarte, 1 von 2

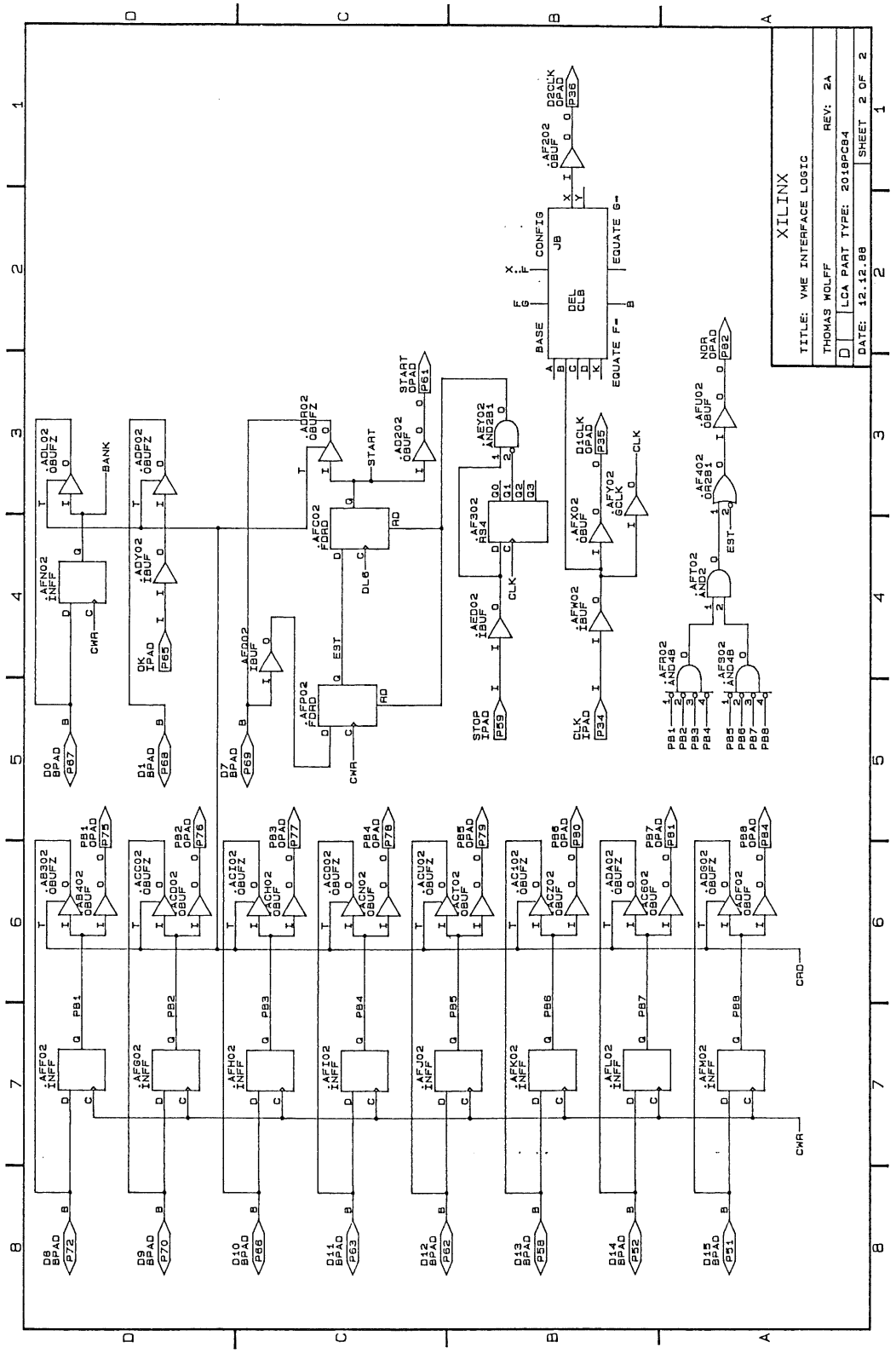

Abbildung 55: Xilinx–Diagramm des VME–Interfaces auf der Testkarte, 2 von 2

D.3 Abbildungen der Karten

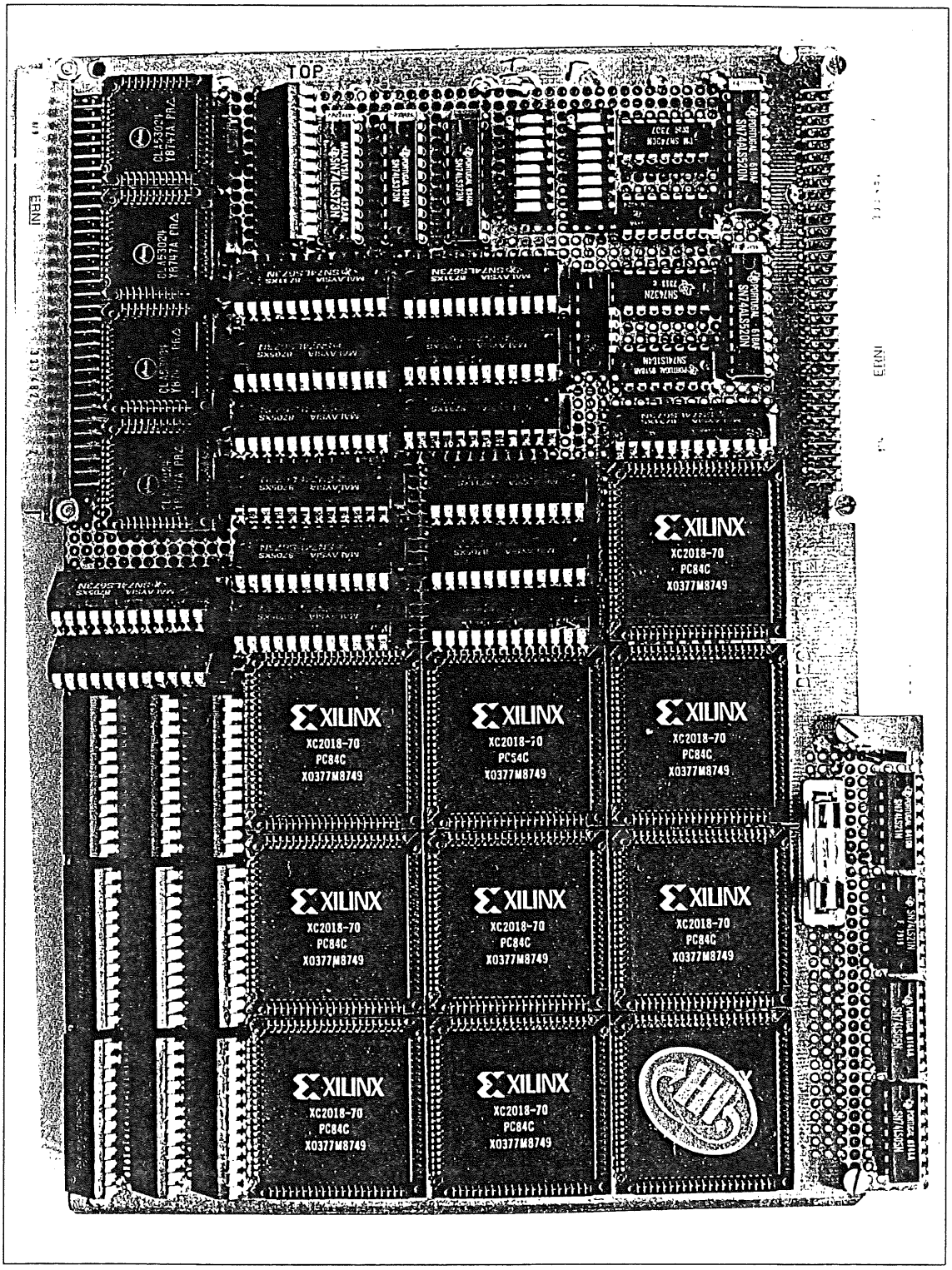

Abbildung 56: Abbildung der Triggerkarte, Komponentenseite

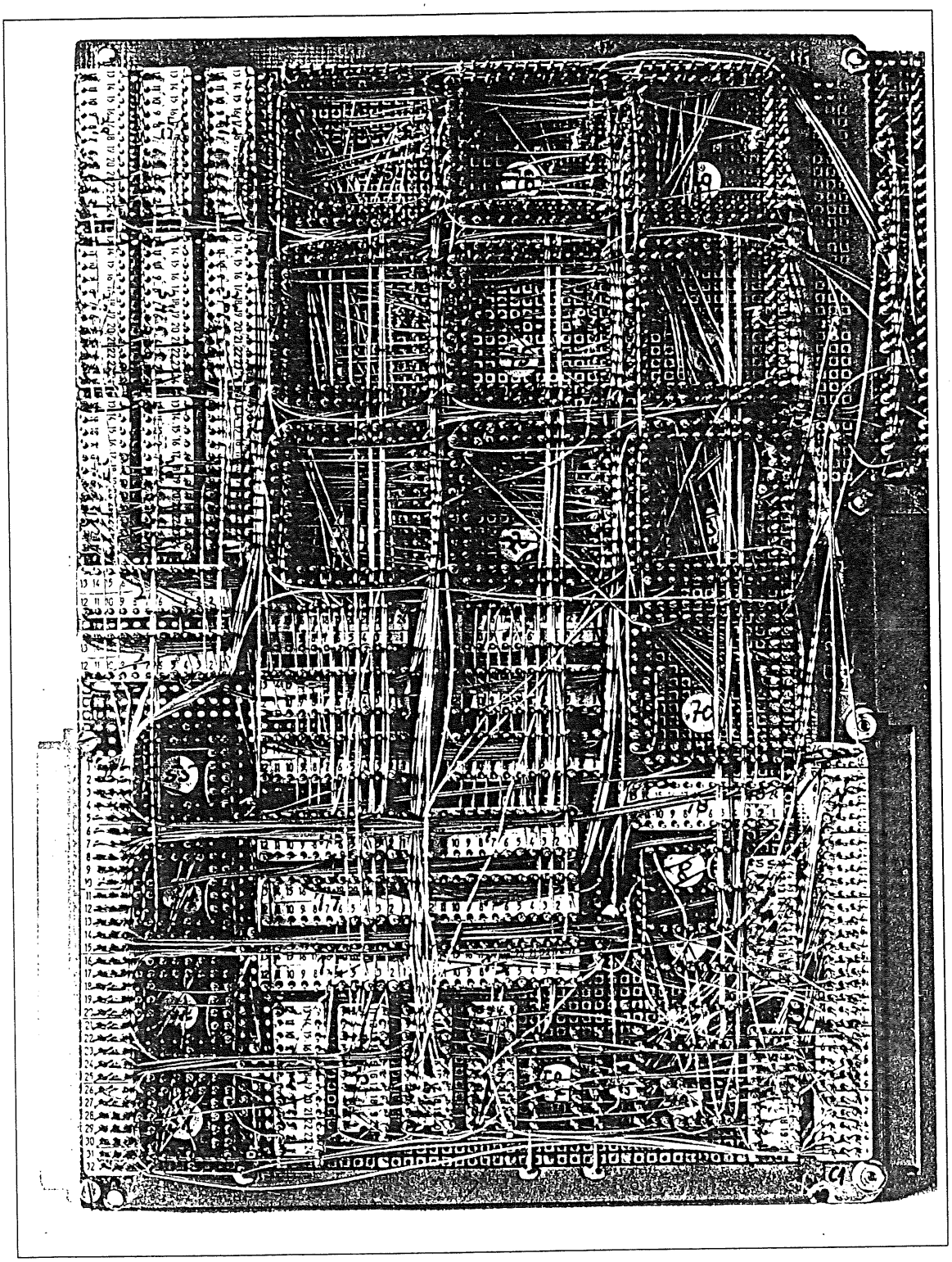

Abbildung 57: Abbildung der Triggerkarte, Wrapseite

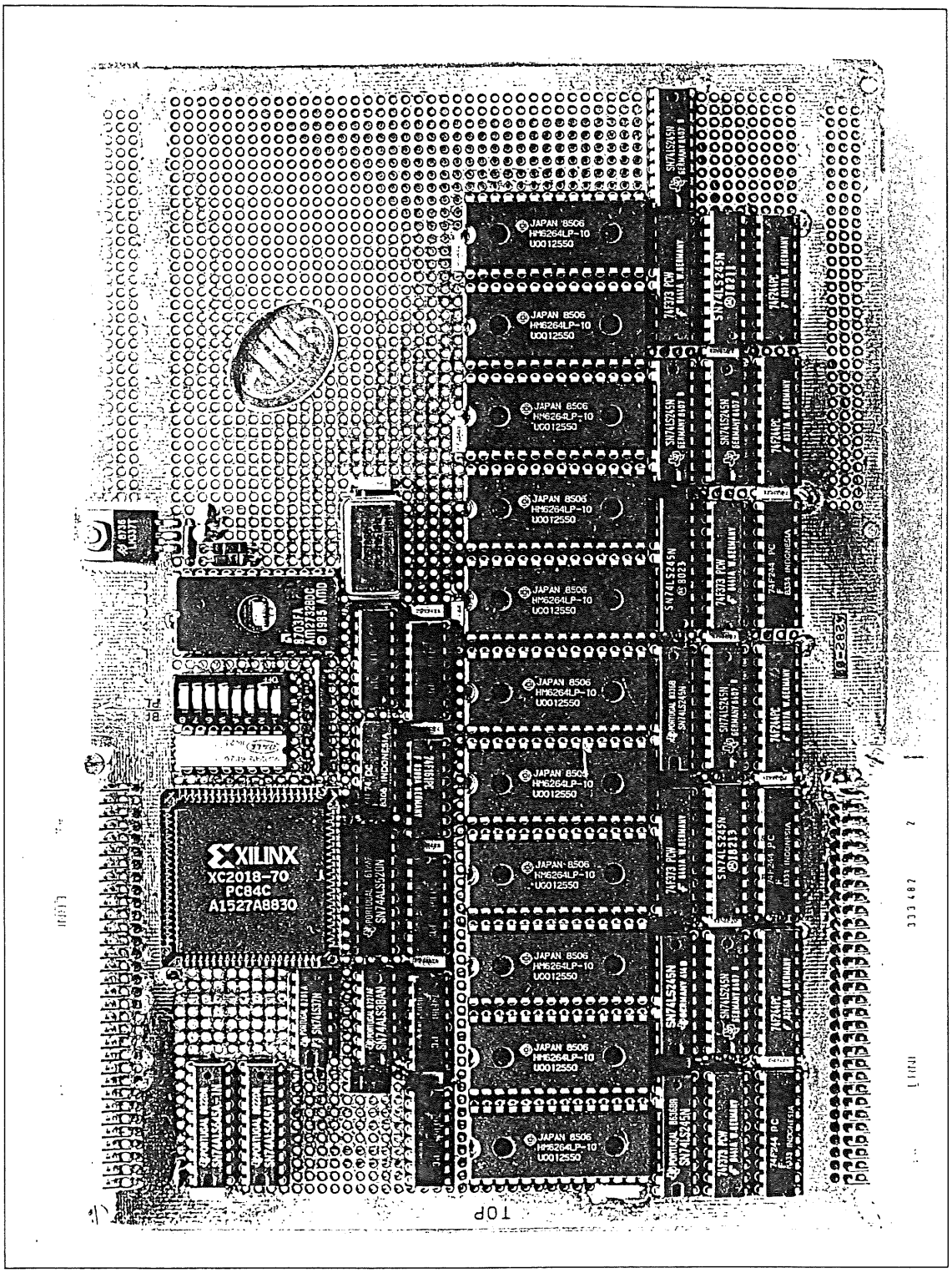

Abbildung 58: Abbildung der Triggertestkarte, Komponentenseite

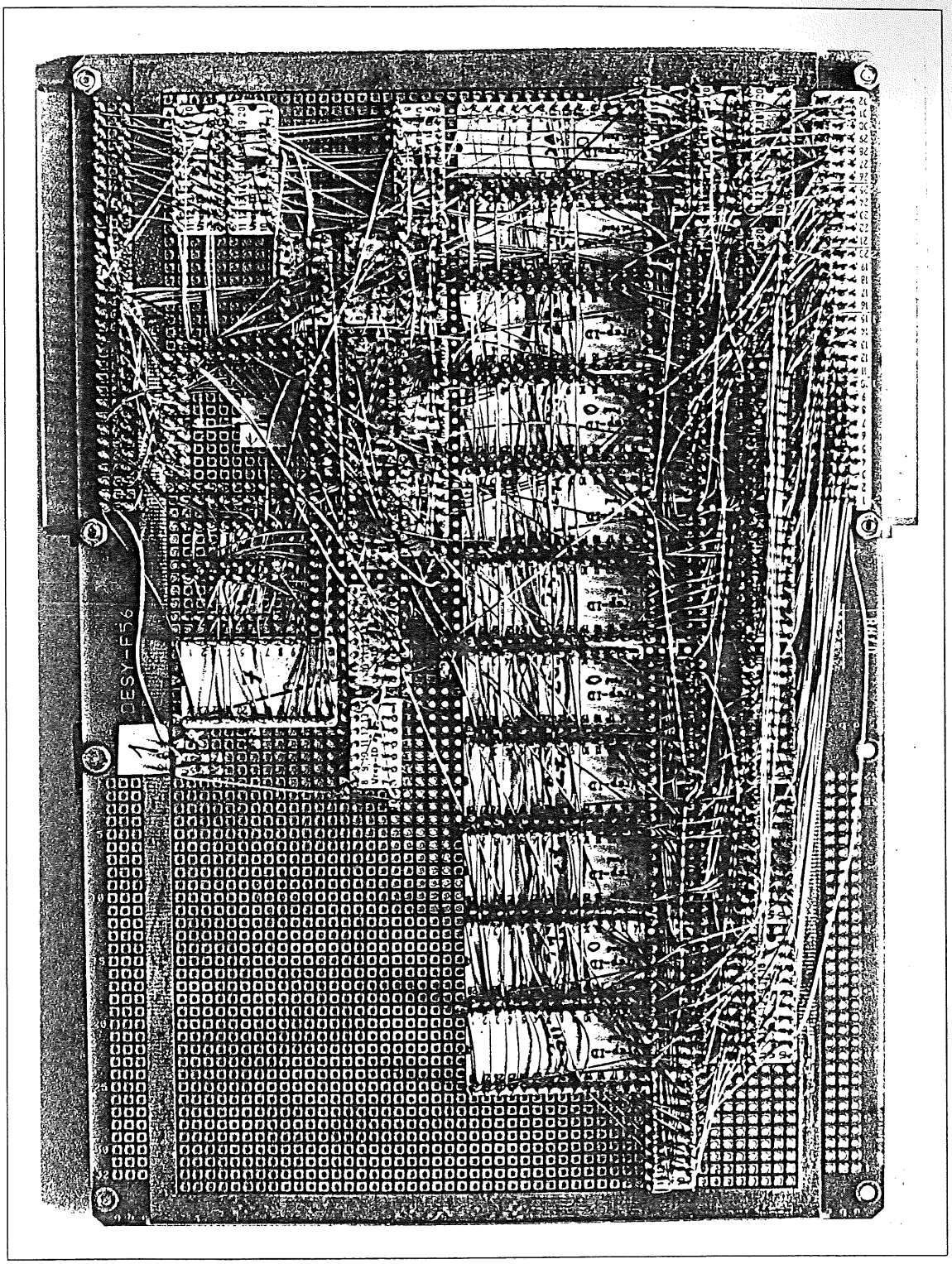

Abbildung 59: Abbildung der Triggertestkarte, Wrapseite

Literatur

[1] G. Wolf,
HERA: Physics, Machine and Experiments,
DESY preprint 86–089 (1986)

[2] B.H. Wiik,
Electron–Proton Colliding Beams, the Physics Programme and the mashine,,
proc. 10^{th} SLAC Summer Institute, ed.A. Mosher (1982)

[3] H1 Collaboration,
Technical Proposal for the H1–Detector,
Hamburg 1986

[4] H 1 Collaboration,
The central tracking system of the H1–Experiment,
DESY 88–293 (1988)

[5] P. Steffen,
persönliche Mitteilung,
DESY (1988)

[6] H.J. Stuckenberg,
persönliche Mitteilung,
DESY (1988)

[7] H.J. Behrend,
Hardware Triggers Using the Driftchamber of H1,
11/87-74 Internal Report Desy

[8] H. Krehbiel,
A Memo about the Sync Gate Array in H1 MWPC Circuits,
11/87-74 Internal Report Desy

[9] F. Sauli,
Principles of Operation of Multiwire Proportional and Drift Chambers,
CERN 77–09 (1977)

[10] A. Peisert und F. Sauli,
Drift and Diffusion of Elektrons in Gases: A Compilation,
CERN 84–08 (1984)

[11] K. Kleinknecht,
Detektoren für Teilchenstrahlung,
Teubner (1984)

[12] M.Schulz,
Aufbau und Test von Prototypen für eine hochauflösende Jetkammer,
Diplomarbeit Dortmund (1988)

[13] W.v.Wendorff,
Ein Trigger der dritten Stufe für das H1-Experiement,
Diplomarbeit Hamburg (1988)

[14] G.Westerkamp,
Aufbau und Test eines Prototypen für die zentrale Jetkammer des Detektors H1
Diplomarbeit Hamburg (1988)

[15] W. Zimmermann,
persönliche Mitteilung,
DESY (1988)

[16] MOTOROLA,
MC68000 16-Bit Microprozeesor User's Manual,
third edition (1982)

[17] XILINX,
The Programmable Gate Array Design Handbook,
Xilinx (1986)

[18] Bronstein, Semendiaiew,
Taaschenbuch der Mathematik,
Harri Deutsch(1981)

[19] D.H. Perkins,
Introduction to High Energy Physics,
Addision–Wesley (1982)

[20] A. Rost,
Grundlagen der Elektronik,
Akademie–Verlag Berlin (1988)

[21] G. Fischer,
Lineare Algebra,
Vieweg (1984)

[22] F. Schmidt,
Untersuchungen zur dynamischen Akzeptanz von Protonenbeschleunigern und ihre Begrenzung durch chaotische Bewegung,
DESY HERA 88-02

[23] XILINX,
XC3000 Logic Cell Array Family, Technical Data,
DESY HERA 88-02

[24] Borland corp. int.,
Turbopascal 5.0 USER's Manual,
Borland 1988

[25] Microsoft,
DOS USER's Manual,
Microsoft 1986

[26] IBM,
IBM AT Reference Manual,
IBM 1986

[27] Lutz Russek,
Diplomarbeit an der Universität Hamburg (1989)

[28] *H1TR300*

[29] V. Blobel *The Bos System, Dynamic Memory Management,*
DESY Internal Report, Desy R1-88-01 (1988)

[30] Motorola *VME-Bus Specification Manual,*
Motorola inc.(1981)

[31] Cern Data Handling Division *GEANT3*
Cern, DD/EE/84-1 (1984)

E Danksagung

Ich danke denen, die mir den Spaß an dieser Diplomarbeit ermöglicht haben. Dies gilt im besonderen für H.-J. Stuckenberg sowie für P. Steffen. Ich danke auch Prof. G. Heinzelmann für die Betreuung dieser Diplomarbeit. Dank sei auch an meinen Freund W. v.Wendorff gerichtet, der mir gerade in der Anfangsphase hilfreich zur Seite stand. Besonderer Dank gilt der Gruppe F56, die mich für die Zeit der Diplomarbeit ertragen mußte. Ferner sei auch M. Ernst gedankt, der mir für lange Zeit die XILINX–Entwicklungssoftware zur Verfügung gestellt hat.