Test und Simulation der Track-Finding-Unit des HERA-B-First-Level-Triggers

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von
Christian Hähnel
aus Ingelheim am Rhein

Mannheim, 2001

Dekan: Professor Dr. Guido Moerkotte, Universität Mannheim
Referent: Professor Dr. Reinhard Männer, Universität Mannheim
Korreferent: Professor dr.ir. Peter H.N. de With, University of Technology Eindhoven TU/e, Eindhoven, Niederlande

Tag der mündlichen Prüfung: 10. Mai 2001

Überblick

Zur Reduktion der hohen Datenrate des HERA-B-Experiments am DESY in Hamburg benötigt man ein leistungsfähiges, mehrstufiges Triggersystem. Die erste Stufe, ein asynchrones und paralleles Pipeline-Multiprozessorsystem, wurde vom Lehrstuhl Informatik V der Universität Mannheim hergestellt.

Damit der Produktionszeitplan für die Trigger-Komponenten, insbesondere für die 90 dicht bestückten Platinen der Track-Finding-Unit (TFU), eingehalten werden konnte, war man zur Diagnose der Produktionsfehler und zur Qualitätskontrolle auf effiziente Teststrategien angewiesen. Zur Design-Verifikation der Trigger-Hardware und für die Tests, zur Inbetriebnahme des Triggersystems und zur Berechnung der Effizienz des Experiments benötigte man eine exakte Verhaltenssimulation.

In der vorliegenden Dissertationsschrift werden die Anforderungen an die Simulation und die Tests untersucht. Ferner wird ihre Realisation beschrieben und diskutiert. Die Tests werden in den aktuellen Stand der Forschung und der Technik eingeordnet.

Die Berücksichtigung von Testbarkeitsaspekten beim Design der Hardware machte es möglich, die Kosten und den Zeitbedarf der Tests zu reduzieren. Die Integration von Scan-Pfaden erlaubte es, zur Diagnose statischer Fehler den TFU-Prozessor in einzelne Testklassen zu unterteilen und diese effizient zu testen.

Des weiteren werden Vorschläge zur Verbesserung der Testbarkeit bei künftigen, ähnlichen Designs unterbreitet und anhand von Simulationen diskutiert. Sie ermöglichen es, die Teststrategie zur Diagnose statischer Fehler auf dynamische Fehler auszuweiten.

Inhaltsverzeichnis

Kapitel 1

Einleitung und Motivation

In diesem Kapitel werden das HERA-B-Experiment und die Aufgabe des First-Level-Triggers vorgestellt und, in diesem Kontext, die Motivationsgründe der Dissertation dargelegt.

1.1 Das HERA-B-Experiment

Mit dem HERA-B-Experiment am Deutschen Elektronen-Synchrotron (DESY) in Hamburg wird die Verletzung einer Symmetrie physikalischer Gesetze bezüglich Materie und Antimaterie bei Zerfällen massereicher Elementarteilchen, die ein b-Quark enthalten, untersucht, die CP-Verletzung. Es handelt sich dabei um eine Verletzung von Symmetrien im Bereich der Quantenmechanik, wobei die P-Parität die Spiegelsymmetrie im Ortsraum und die C-Parität die Teilchen-Antiteilchen-Konjugation darstellen [Pov93]. Die CP-Verletzung wird von der theoretischen Physik vorhergesagt und äußert sich darin, daß bestimmte Zerfälle von B-Mesonen in ein Teilchen und sein Antiteilchen nicht gleich wahrscheinlich sind. Sie bietet eine mögliche Erklärung für das Fehlen von Antimaterie im Universum [Sak67].

Um die erforderlichen B-Mesonen zu erzeugen, werden Protonen des Elektron-Proton-Rings HERA mit einer Energie von 920 GeV auf ein ruhendes Target (*deutsch* Ziel) geschossen. Bei der Kollision der hochenergetischen Strahlteilchen mit dem Target entstehen neue Teilchen-Antiteilchen-Paare, die aufgrund der Impulserhaltung in Strahlrichtung weiterfliegen. Diese sind instabil und zerfallen sehr schnell in leichtere Teilchen. Durch geeignete, hinter dem Target angeordnete Detektoren (siehe Abbildung 1.1) lassen sich diese Teilchen nachweisen und ihre Spuren rekonstruieren.

Nach Abschätzungen werden nur bei einer einzigen von 10^{12} Kollisionen B-Mesonen erzeugt, welche dann bei ihrem Zerfall möglicherweise die

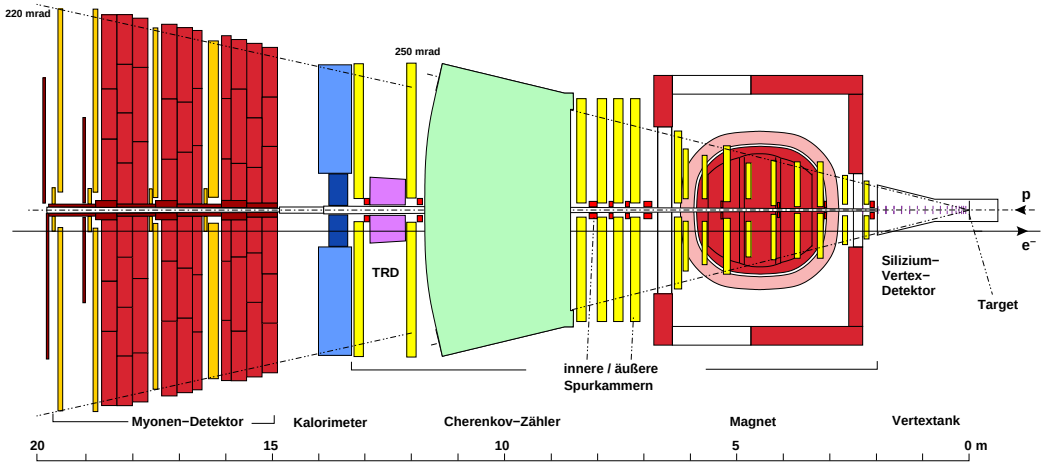


Abb. 1.1. Der Detektor: Die Protonen (p) fliegen von rechts auf das Target zu. Bei der Kollision mit dem Target entstehen neue Teilchen, die bei ihrem Weiterflug hinter dem Target in leichtere Teilchen zerfallen. Die Teilchen hinterlassen im Detektor Spuren und können dadurch nachgewiesen werden.

CP-Symmetrie verletzen. Zur Kompensation der geringen Wahrscheinlichkeit dieser Zerfälle arbeitet das Experiment mit einer hohen Ereignisrate von ca. 10 MHz, indem die Protonen alle 96 ns gebündelt auf das Target geschossen werden. Die Meßsysteme des gesamten Detektors erzeugen eine Datenrate von 10 TByte/s.[1]

Die geringe Wahrscheinlichkeit des gesuchten Zerfalls und die hohe Datenrate erfordern ein mehrstufiges Triggersystem, das hoch selektiv und effizient die physikalisch interessanten Teilchenspuren aus den Detektordaten herausfiltert.

1.2 Der First-Level-Trigger

Die Aufgabe des Triggersystems besteht in der Reduktion der Datenrate des Detektors bei hoher Nachweiseffizienz für den gesuchten Zerfall. Dazu sucht es aus allen Detektordaten nach vorher festgelegten Kriterien maximal so viele Daten heraus, wie die folgende Elektronik zur Aufzeichnung auf Magnetbändern verarbeiten kann.

Beim HERA-B-Experiment setzt man zur Reduktion der Datenrate einen mehrstufigen Trigger ein, dessen einzelne Stufen mit unterschiedlicher Geschwindigkeit und Genauigkeit arbeiten. Die erste Stufe, der First-Level-Trigger (FLT), analysiert nur einen Teil der Detektordaten, bei einem Voll-

[1] 1 T (Tera) = 10^{12}

ausbau des Detektors ca. 100.000 Spurkammerdrähte. Da ein Spurkammerdraht einem Datenbit entspricht, ergibt das mit der Ereignisrate von ca. 10 MHz eine Datenrate von ca. 10^{12} Bit/s. Die Auswahl des FLTs wird vom Second-Level-Trigger unter Berücksichtigung weiterer Detektordaten verfeinert. Die folgenden beiden Stufen überprüfen zum Schluß alle Daten.

Die Ereignisrate des Experiments von ca. 10 MHz muß der FLT um zwei bis drei Größenordnungen auf die Eingangsrate des Second-Level-Triggers von maximal 50 kHz reduzieren. Um eine Reduktion in dieser Größenordnung zu erlangen, sucht der FLT in den Detektordaten nach den Spuren bestimmter Teilchen, rekonstruiert diese und errechnet daraus die vektoriellen Impulse und die Ortsinformationen der Teilchen, die sog. Spurparameter. Die Spuren werden paarweise kombiniert, die invariante Masse der Teilchenpaare berechnet und daraus die Triggerentscheidung abgeleitet.

Der FLT hat maximal 9-10 μs Zeit, seine Entscheidung zu treffen, denn nach dieser Latenzzeit werden die Detektordaten, welche von der Elektronik des Detektors zwischengespeichert werden, überschrieben und gehen somit für eine weitere Auswertung durch den Second-Level-Trigger und die nachfolgenden Stufen verloren.

Diese Anforderungen können nur mit kommerziell nicht erhältlichen Spezialrechnern erfüllt werden, einem asynchronen Multiprozessorsystem, dessen Pipeline-Architektur für diese Aufgabenstellung optimiert ist. Der FLT wurde gemeinsam mit dem DESY vom Lehrstuhl Informatik V der Universität Mannheim entwickelt und gefertigt.

1.2.1 Funktionsweise des First-Level-Triggers

Zur Realisation der Funktion des FLTs wurden drei verschiedene Arten von Hardware-Prozessoren mit jeweils unterschiedlichen Aufgaben entwickelt. 70 Track-Finding-Units (TFU) sind für die Spurensuche zuständig, vier Track-Parameter-Units errechnen die Spurparameter und eine Trigger-Decision-Unit fällt die Triggerentscheidung.

Abbildung 1.2 zeigt im oberen Teil ein Beispiel für eine Teilchenspur durch den Detektor. Nun wird beschrieben, wie der FLT diese Spur vom Kalorimeter aus durch die verschiedenen Detektoreben in Richtung des Targets zurückverfolgt. Der Algorithmus zur Spurensuche wurde von der Kalman-Filterung abgeleitet [Har95].

Das Kalorimeter registriert den Ort und die Energie eines auftreffenden Teilchens. Wenn die Teilchenenergie innerhalb eines bestimmten Energiebereichs liegt, werden die Orts- und Energie-Informationen von einem angeschlossenen Pretrigger an den FLT weitergegeben. Aus dem Auftreffort des Teilchens im Kalorimeter kann man berechnen, welchen Bereich der vorheri-

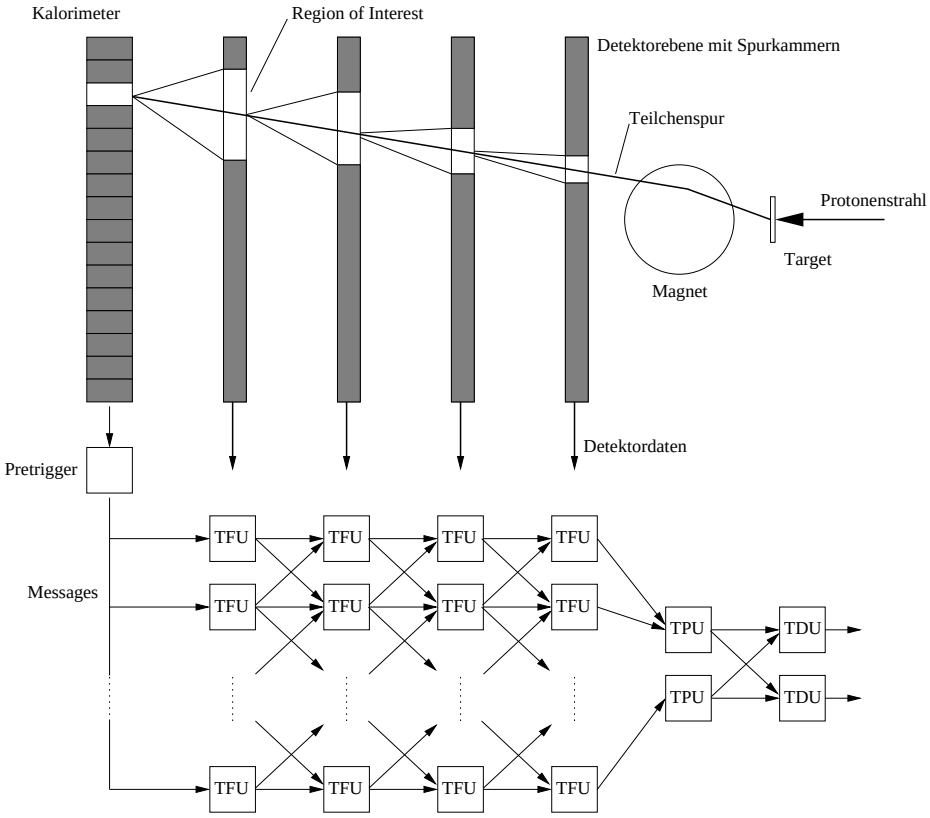


Abb. 1.2. Im oberen Teil sind vier Detektorebenen und das Kalorimeter skizziert. Die Spur eines Teilchens, das von rechts geflogen kommt, wird vom Kalorimeter aus bis zum Target rekonstruiert. Im unteren Teil sind die Prozessoren des First-Level-Triggers (FLT) skizziert. Der FLT sucht in den jeweiligen Suchbereichen oder Regions-of-Interest der einzelnen Detektorebenen die Spur des Teilchens. Die Track-Finding-Units (TFU) sind jeweils für eine bestimmte Region des Detektors zuständig, von der sie die Detektordaten zur Analyse erhalten. Aus den Spurinformationen des Kalorimeters generiert der Pretrigger eine Message und sendet diese an die zuständigen TFUs, welche sie über verschiedene Ebenen von TFUs an die Track-Parameter-Unit und Trigger-Decision-Units weiterleiten.

gen Detektorebene das Teilchen durchquert haben muß, diesen Suchbereich bezeichnet man als Region-of-Interest.

Für jeden Detektorbereich ist eine bestimmte TFU zuständig. Alle für den Suchbereich zuständigen TFUs erhalten von dem Pretrigger eine sog. Message *(deutsch* Nachricht), welche die zur Identifikation einer Teilchenspur notwendigen Informationen in einem fest definierten Format enthält. Diese Messages werden über eine ebenfalls speziell entwickelte Sender- und

Empfänger-Elektronik von einem zum nächsten FLT-Prozessor-Board[2] gesandt.

Die TFUs suchen in den Detektordaten ihres Suchbereichs nach der Spur des Teilchens. Wird die Spur gefunden, so kann die jeweilige TFU daraus den neuen Suchbereich in der nächsten Detektorebene berechnen. Sie aktualisiert die Spurkoordinaten in der Message mit den gefundenen Informationen und sendet sie an die für den neuen Suchbereich zuständigen TFUs. Dieser Vorgang wird jetzt in allen vom First-Level-Trigger ausgewerteten Detektorebenen wiederholt. Die Suchbereiche werden dabei zum Target hin immer kleiner, da immer genauere Kenntnisse über den Spurverlauf vorliegen.

Nach der Bearbeitung durch die TFUs wird die Message an eine Track-Parameter-Unit gesandt, welche aus den Spurinformationen die Parameter der Spur berechnet und Spuren von Teilchen mit einem Impuls außerhalb bestimmter Grenzwerte aussortiert. Anschließend erhält die Trigger-Decision-Unit die übriggebliebenen Messages, sammelt sie, berechnet die invariante Masse von Spurpaaren und fällt durch das Eingrenzen des Massebereichs und der Spurparameter die Triggerentscheidung.

Die einzelnen FLT-Boards wurden als Hardware-Prozessoren mit einer synchronen Pipeline realisiert. Jedes Prozessor-Board arbeitet mit einer Taktrate von 50 MHz, so daß sie alle 20 ns eine Message in ihre Pipeline übernehmen und bearbeiten können. Der Takt ist asynchron zu den anderen Boards und zum Beschleunigertakt.

Der Aufbau der TFU-Hardware wird in Kapitel 3 beschrieben.

1.3 Motivation der Dissertation

Diese Dissertation untersucht speziell die TFU. Die Track-Parameter-Unit ist ähnlich aufgebaut, so daß viele der für die TFU entwickelten Konzepte nahezu unverändert übernommen werden konnten. Die Trigger-Decision-Unit ist das Thema der Dissertation von Gröpl.

1.3.1 Simulation der TFU

Da alle Spuren, welche der FLT nicht innerhalb seiner Latenzzeit rekonstruieren und daher auch keiner Triggerentscheidung unterwerfen kann, für eine weitere Auswertung verloren gehen, hat das Zeitverhalten des Triggers einen direkten Einfluß auf die Effizienz des Experiments. Zur Berechnung der Effizienz benötigt man eine Simulation des genauen FLT-Zeitverhaltens [Wol98].

[2]Board: Abkürzung für: Printed circuit board, *deutsch* Platine

Anhand einer Simulation muß man darüber hinaus das Design der Trigger-Hardware in einem möglichst frühen Entwicklungsstadium überprüfen. Während der anschließenden Produktion benötigt man eine Simulation zum Test der Hardware, für die Inbetriebnahme des Triggers als Diagnoseinstrument und zur Bestimmung der Betriebsparameter.

Um doppelten Quellcode und die daraus folgenden Fehler zu vermeiden, benötigt man ein Simulationssystem, das alle diese unterschiedlichen Anforderungen, die in Kapitel 4 nochmals ausführlich beschrieben werden, erfüllt.

Die Simulation des gesamten Triggers baut auf die Simulationsmodelle der einzelnen FLT-Boards auf. In Kapitel 4 werden speziell die Entwicklung des TFU-Modells, die Schwierigkeiten, die sich dabei ergeben haben, und ihre Lösung beschrieben.

Das Simulationsmodell der TFU wurde in einem Vergleich mit der Hardware überprüft. Damit wurde nicht nur, soweit möglich, die exakte Übereinstimmung beider geprüft, sondern auch, ob bei der Umsetzung der Spezifikationen der TFU sowohl bei der Entwicklung der Simulationssoftware als auch der Hardware Fehler unterlaufen sind. Dieser Vergleich ist ebenfalls Thema des Kapitels 4.

1.3.2 Test der TFU-Hardware

Die Boards mit diesen Prozessoren bestehen jeweils aus einer 36×40 cm^2 großen Platine und sind doppelseitig dicht mit Bauteilen in der platzsparenden SMD-Technologie[3] bestückt. Eine TFU besitzt ca. 20.000 Lötstellen. Insgesamt wurden 90 TFUs produziert.

Bei der Produktion traten Hardware-Defekte auf, welche Fehler im Verhalten der TFU verursachten. Nur eine effektive Diagnose dieser Fehler ermöglichte eine schnelle Korrektur, sie war eine Voraussetzung zur Einhaltung des Produktionszeitplans von 3 TFUs pro Woche. Man benötigte Teststrategien, die diese effektive Diagnose unterstützten, und um die Qualität und das korrekte Verhalten der TFUs zu garantieren. Ein Fehlverhalten, das ohne Tests im ungünstigsten Fall nicht entdeckt wird, könnte die Effizienz des gesamten Hera-B-Experiments mindern. Auch nach der Produktion können beim Einsatz der TFU im FLT neue Hardware-Fehler auftreten, welche man durch wiederholtes Testen finden muß.

Es ein Ziel dieser Dissertation, Strategien zum Test der TFU zu entwickeln, die es ermöglichen, alle möglichen Hardware-Fehler der TFU zu diagnostizieren.

In Kapitel 2 wird zuerst der aktuelle Stand in Forschung und Technik beim

[3]SMD: *engl.* surface mounted device

Test von Hardware untersucht, bevor in Kapitel 5 die eingesetzten Teststrategien beschrieben und ihre Effizienz diskutiert.

Wegen seiner hohen Kosten wurde kein In-Circuit-Test eingesetzt, sondern auf der TFU wurden schon beim Design Möglichkeiten integriert, über welche man beim Test auf Knotenpunkte der Schaltung zugreifen kann. Man benötigt effiziente Software, die unter Einsatz dieser Möglichkeiten die Schaltung umfassend testet und die Fehler möglichst genau lokalisiert. Die entwickelte Software wird in Kapitel 5 beschrieben und diskutiert.

1.3.3 Verbesserungsvorschläge

Im Kapitel 6 werden Vorschläge zur Verbesserung der Testbarkeit bei künftigen, ähnlichen Entwicklungen gemacht, insbesondere im Hinblick auf den Test von dynamischen Fehlern, welche im Vergleich zu statischen Fehlern höhere Anforderungen an ihre Diagnose stellen.

1.4 Aufbau dieser Arbeit

Die drei genannten Motivationsgründe für diese Arbeit werden in je einem eigenen Kapitel diskutiert. Im Anschluß an jedes Kapitel wird an dieser Stelle auf den Status und Ausblick des jeweiligen Themengebiets eingegangen, um die inhaltliche Einheit der der Kapitel zu wahren.

1.5 Zusammenfassung

Das HERA-B-Experiment des DESY in Hamburg untersucht die CP-Verletzung beim Zerfall von B-Mesonen. Zur Reduktion der beim Experiment anfallenden hohen Datenrate von 10 TByte/s kommt ein mehrstufiges Triggersystem zum Einsatz. Die erste Stufe, der First-Level-Trigger (FLT), ein asynchrones Multiprozessorsystem, wurde gemeinsam mit dem DESY vom Lehrstuhl Informatik V der Universität Mannheim entwickelt und gefertigt.

Der FLT sucht in einem Teil der anfallenden Detektordaten nach den Spuren bestimmter Teilchen, rekonstruiert sie und leitet aus ihren Spurparametern die Trigger-Entscheidung ab. Der FLT hat maximal 9-10 μs Zeit, seine Entscheidung zu treffen, denn nach dieser Latenzzeit werden die Detektordaten, welche von der Elektronik des Detektors zwischengespeichert werden, überschrieben und gehen für eine weitere Auswertung durch den Second-Level-Trigger und die nachfolgenden Stufen verloren.

Zur Bestimmung der Effizienz des Experiments ist es notwendig, das genaue Zeitverhalten des Triggers mit einer Simulation zu berechnen, da-

mit seine Latenzzeit berücksichtigt werden kann. Da der FLT aus mehreren FLT-Prozessoren besteht, benötigt man zur Simulation die exakten Verhaltensmodelle dieser Prozessoren. Dies sind die Track-Finding-Unit (TFU), die Track-Parameter-Unit und die Trigger-Decision-Unit. Ein Ziel dieser Dissertation war die Entwicklung des Modells zur Simulation der TFU und dessen Überprüfung durch einen Vergleich mit der Hardware.

Diese Dissertation untersuchte speziell die TFU. Die Track-Parameter-Unit wurde ähnlich aufgebaut, so daß viele der für die TFU entwickelten Konzepte nahezu unverändert übernommen werden konnten.

Bei der Produktion der 90 dicht bestückten TFUs traten Hardware-Defekte auf, die Fehler im Verhalten der TFU verursachten. Nur eine effektive Diagnose dieser Fehler durch Tests ermöglichte eine schnelle Korrektur und war die Voraussetzung zur Einhaltung des Produktionszeitplans von drei TFUs pro Woche.

Diese Tests ermöglichten, daß man sowohl bei der Produktion als auch beim Betrieb des FLTs das korrekte Verhalten der TFUs prüfen und damit ihre Qualität garantieren konnte.

In der vorliegenden Dissertationsschrift werden die Anforderungen an die Tests untersucht, ihre Realisation beschrieben und diskutiert. Die Tests werden in Kapitel 2 in den aktuellen Stand der Forschung und der Technik eingeordnet und Verbesserungsvorschläge für künftige, ähnliche Entwicklungen aufgezeigt.

Kapitel 2

Theoretische Grundlagen und ihre Anwendung

Im ersten Teil des Kapitels werden Fehler und ihre Ursachen in der Hardware charakterisiert, im zweiten Teil die verschiedenen Strategien zur Diagnose der Fehler mit ihren Vor- und Nachteilen untersucht und im dritten Teil kurz die Grundlagen der Simulation digitaler Schaltungen dargelegt.

2.1 Fehler in digitalen Schaltungen

Eine Schaltung oder ein System ist fehlerhaft, wenn sie von ihrem spezifizierten Verhalten abweicht. Ein Hardware-Fehler ist ein physikalischer Defekt, der ein solches Versagen verursachen kann [Lal97].

Der Test ist ein Experiment an der Schaltung zur Suche eventuell vorhandener Fehler. Für eine gezielte und effektive Suche ist es notwendig, die Fehler und ihre Ursachen zu verstehen. In den folgenden Abschnitten werden die Fehler charakterisiert und verschiedene Fehlermodelle vorgestellt. Die Fehlermodelle ermöglichen eine mathematische Behandlung der Fehler und erlauben damit die Erzeugung von Tests, sowie Aussagen über die Fehlerdeckung oder Effizienz eines Test (siehe Kapitel 2.2).

2.1.1 Auftreten von Fehlern

Für physikalische Fehler können Fehler im Design der Schaltung oder Fertigungsfehler verantwortlich sein.

Designfehler treten auf, wenn z.B. die Spezifikation unvollständig oder inkonsistent ist, oder Designregeln verletzt wurden [Abr90, Kap.1]. Beispielsweise sind auf Platinen in der Regel gegenseitig nicht abgeschirmte Leiter-

bahnen nebeneinander angeordnet. Durch induktive und kapazitive Kopplung resultiert dann eine als Übersprechen bezeichnete gegenseitige Beeinflussung der Leitungsvorgänge [Rei97]. Man muß beim Design der Leiterbahnen diese so dimensionieren und anordnen, daß beim Betrieb der Schaltung keine Störungen der Signale durch Übersprechen enstehen können, welche zu Fehlern führen.

Zu den Fertigungsfehlern zählen Fehler durch defekte oder falsche bestückte Bauteile, Bauteile, welche ihre Spezifikation nicht einhalten, Unterbrechungen oder Kurzschlüsse von Leiterbahnen, Lötfehler oder auch Leiterbahntoleranzen, die zu Signalverzögerungen führen können [Lal97].

Nach der Fertigung können weitere Fehler auftreten [Woj88]. Durch elektrostatische Entladungen werden CMOS-Bauteile beschädigt. Auch eine thermische oder elektrische Überlastung von Bauteilen führt zu Fehlern, beides sind Einflüsse aus der Betriebsumgebung, die durch geeignete Maßnahmen wie z.B. Kühlung vermieden werden können. Eine häufige Fehlerursache sind Steckverbindungen mit mangelhaftem elektrischen Kontakt.

2.1.2 Fehlerarten

Ein Fehler wird charakterisiert nach seiner Eigenschaft, dem Spannungswert des fehlerhaften Signals, seiner Ausdehnung und seiner Dauer [Lal97].

Die Eigenschaft eines Fehlers kann als logisch oder nichtlogisch klassifiziert werden. Ein logischer Fehler führt zu einem invertierten logischen Signal. Nichtlogische Fehler beinhalten den Rest der Fehler, wie ein fehlerhaftes Taktsignal oder der Ausfall der Spannungsversorgung.

Der Spannungswert des fehlerhaften Signals weist darauf hin, ob der Fehler feste oder variierende Logikwerte produziert. So sind in der Digitaltechnologie bestimmte Spannungspegel undefiniert und lassen sich keinem festen Logikpegel zuordnen.

Die Ausdehnung eines Fehlers spezifiziert, ob der Effekt des Fehlers lokal oder verbreitet ist. Ein lokaler Fehler betrifft nur eine einzige Variable, der verbreitete Fehler dagegen mehrere, wie z.B. das fehlerhafte Taktsignal.

Ein Fehler kann permanent oder temporär sein, also von unterschiedlicher Dauer. Besonders temporäre Fehler, das sind z.B. Wackelkontakte oder Störungen von außen wie Einbrüche in der Spannungsversorgung, sind schwer zu finden.

Ein Fehler wird auch durch den Ort seines Auftretens charakterisiert, ob er z.B. in einem Bauteil, auf der Platine oder zwischen den einzelnen Modulen auftritt [Aue96]. Demnach können den Fehlern unterschiedliche Technologien zugrunde liegen, z.B. Fehler in integrierten CMOS-Schaltungen oder auf den Platinen.

Weiter lassen sich statische und dynamische Fehler unterscheiden. Unter dem Begriff des dynamischen Fehlers faßt man Effekte wie z.B. zu flache Signalflanken, Reflexionen der Signale am Leitungsende oder unterschiedliche Signallaufzeiten, welche zu Fehlschaltungen in der Logik führen, zusammen. Auch Fehler durch Übersprechen zwischen Signalleitungen fallen in diese Gruppe. Fehler dieser Art lassen sich zum großen Teil durch genaue Einhaltung der Designregeln für den logischen und den geometrischen Entwurf vermeiden [Woj88]. Mit zunehmender Taktfrequenz spielen diese Fehler eine immer größere Rolle und stellen hohe Anforderungen an den Designer der Schaltung.

2.1.3 Fehlermodelle

Generell wird der Effekt eines physikalischen Fehlers durch ein Modell repräsentiert, welches die Änderung der Signale der Schaltung, also den logischen Fehler, durch einen physikalischen Fehler beschreibt.

Fehlermodelle haben den Vorteil, daß

- verschiedene physikalische Fehlerursachen durch den selben logischen Fehler beschrieben werden können,
- einige Fehlermodelle technologieunabhängig sind,
- nach Fehlermodellen hergeleitete Tests auch für physikalische Fehler eingesetzt werden können, deren Effekt auf das Verhalten der Schaltung nicht komplett verstanden wird oder zu komplex für eine Analyse ist [Abr90, Kap.4].

Fehlermodelle finden ihren Einsatz bei der Fehlersimulation, der Simulation der zu testenden Schaltung anhand ihres Modells zusammen mit Fehlern aus einem Fehlermodell.[1] Die Ergebnisse der Fehlersimulation werden verwendet, um das bei einem Test beobachte Verhalten der Schaltung zu interpretieren, und zur Bestimmung der Effizienz eines Tests. Auf diese beiden Punkte wird in Kapitel 2.2 näher eingegangen.

Im folgenden finden sich eine kurze Beschreibung der für diese Dissertation wichtigen Fehlermodelle, nicht erwähnt sind Modelle wie das CMOS-spezifische I_{DDQ}-Fehlermodell.

[1] Zur Erklärung der Begriffe "Simulation" und "Modell" siehe Abschnitt 2.3.

Fehler durch konstante logische Werte

Das allgemeinste Modell für logische Fehler nimmt an, daß sich ein Fehler in einer Logikschaltung darin zeigt, daß einer ihrer Knoten auf den Logikwert 0 (*engl.* stuck-at 0) oder 1 (*engl.* stuck-at 1) fixiert ist. Im folgenden Text wurde für diese Fehler die englische Bezeichnung "Stuck-at" übernommen.

Das Stuck-at-Modell ist eine häufig eingesetzte Repräsentation für die gewöhnlichsten Fehlertypen, z.B. Kurzschlüsse nach Plus oder Masse in vielen Technologien.

Das Modell hat mehrere Vorzüge. Es ist relativ einfach, da jeder Knoten exakt zwei Fehler annehmen kann. Das Verhalten einer fehlerhaften Schaltung ist streng logisch und durch eine geänderte Bool'sche Gleichung beschreibbar. Weil die Zahl der möglichen Fehler einer Schaltung feststeht und das Fehlerverhalten so präzise ist, kann man durch eine Fehlersimulation bestimmen, ob beim Testen eine gegebene Folge von Eingangsmustern einen Fehler entdeckt oder nicht [Ait99].

Das Modell ist relativ erfolgreich bei geringer Integration, jedoch nicht sehr effektiv bei heutiger VLSI (Very Large Scale Integration), welche hauptsächlich in der CMOS-Technologie ausgeführt wird. Fehler in CMOS-Schaltungen produzieren nicht unbedingt logische Fehler, die durch Stuck-at-Fehler beschrieben werden können [Lal97].

Einzelfehlerannahme

Um Fehlermodelle wie den Stuck-at-Fehler einfacher mathematisch behandelbar zu machen, nimmt man an, daß nur ein einziger Stuck-at-Fehler (*engl.* Single Stuck-at Fault) in der Schaltung auftritt. Die Erweiterung auf mehrere Stuck-at-Fehler nimmt dann an, daß mehrere einzelne Stuck-at-Fehler gleichzeitig existieren. Entsprechendes gilt auch für andere Fehlermodelle.

Wenn die Intervalle zwischen Tests zur Überprüfung fertiger Schaltungen kurz genug sind, ist die Wahrscheinlichkeit gering, daß sich mehrere Fehler entwickelt haben. Jedoch manifestieren sich manche physikalische Defekte, wie z.B. ein Kratzer über mehrere Leiterbahnen, nicht als Einzelfehler, genauso wie in Schaltungen direkt nach der Fertigung eher mehrere Fehler auftreten. Dennoch, in den meisten Fällen kann ein Mehrfachfehler durch Tests gefunden werden, die für die einzelnen Fehler entwickelt wurden, aus welchen sich der Mehrfachfehler zusammensetzt [Abr90, Kap.4].

Kurzschlußfehler

Das Kurzschlußfehlermodell (*engl.* Bridging Fault Model) ist eine Erweiterung des Stuck-at-Modells. Es berücksichtigt zusätzlich Kurzschlüsse zur

Spannungsversorgung oder zwischen einzelnen Signalleitungen. Die Wahrscheinlichkeit für Kurzschlüsse steigt mit höherer Integrationdichte von ICs.

Die Ausgänge stärkerer Transistoren verschlucken die Signale von schwächeren (Wired-Logik). Ist der resultierende Signalpegel nicht definiert, so ist das Verhalten der Schaltung weniger klar.

Noch nicht erfaßt sind in diesem Modell Kurzschlüsse zwischen Aus- und Eingängen (*engl.* Feedback), welche die Schaltung zum Oszillieren bringen können oder in ein sequentielles Schaltwerk[2] verwandeln. Das gleiche gilt für Kurzschlüsse zwischen mehr als zwei Leitungen [Ait99][Lal97].

Unterbrechungen

Nicht gut verstanden ist, wie sich Brüche, fehlende oder nur teilweise vorhandene Verbindungen, oder Verbindungen mit einem größerem Widerstand bei einem CMOS-Schaltkreis verhalten. In manchen Fällen arbeitet die Schaltung korrekt, aber langsamer (Delay-Fehler). In anderen bewegt sich die Spannung zu einem definierten Logikpegel hin und wird durch das Stuck-at-Modell erfaßt [Ait99].

Zeitbezogene oder Delay-Fehler

Nicht alle Defekte verändern die Schaltung so, wie das Stuck-at-Fehlermodell es verlangt. Manche verändern das Zeitverhalten der Schaltung, so daß nur bei bestimmten Frequenzen Funktionsfehler auftreten. Ein Modell für diese dynamischen Fehler nennt man ein zeitbezogenes oder Delay-Fehlermodell [Ait99].

Kleinere Defekte treten während des Produktionsvorgangs aufgrund von statistischen Schwankungen mit höherer Wahrscheinlichkeit auf. Dazu zählen insbesondere bei der Chip-Produktion partielle Unterbrechungen oder Kurzschlüsse. Diese Defekte resultieren in dem Versagen der Schaltung, ihre zeitliche Spezifikation einzuhalten, ohne die eigentliche Logikfunktion zu ändern. Solch ein kleiner Defekt kann dazu führen, daß der Wechsel des Logiksignals von 0 nach 1, oder umgekehrt, verzögert wird [Lal97].

Genauso können Reflexionen an Leitungsenden, verursacht durch einen defekten Abschlußwiderstand, sich mit dem eigentlichen Signal überlagern und es dadurch verfälschen, wodurch es zu einem Delay des Signals kommt (siehe Abbildung 2.1) [Häh95].

Man unterscheidet Delay-Fehler in einem einzelnen Transistor (*engl.* Gate Delay Fault) oder entlang eines ganzen Pfads (*engl.* Path Delay Fault) wie

[2]Das ist eine Schaltung, deren logischer Zustand durch die Rückkopplung von früheren Zuständen abhängig ist.

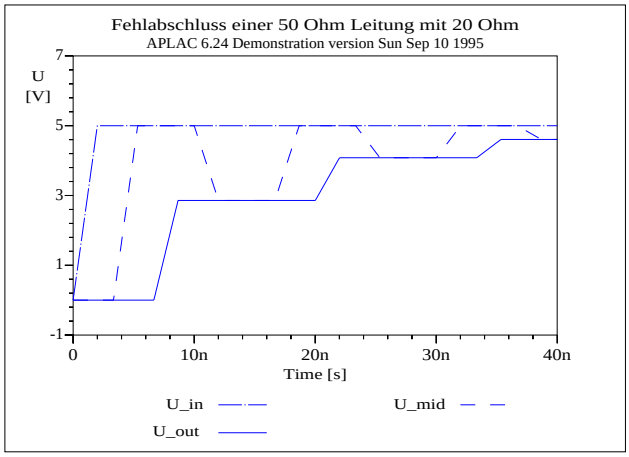


Abb. 2.1. Simulation einer mit einem zu kleinen Widerstand abgeschlossenen Leitung, an deren Eingang die Signalflanke von logisch-0 nach 1 wechselt (U_{in}). Durch die Überlagerung des Signals am Leitungsende U_{out} mit den Reflexionen wird der Signalanstieg verzögert [Häh95].

z.B. das Übersprechen von Signalen [Ait99].

Temporäre Fehler

Ein großer Teil der Fehlfunktionen von digitalen Systemen wird von temporären Fehlern verursacht. Diese Fehler sind schwer zu finden, da sie nicht immer in ihrem aktiven, fehlerverursachenden Zustand sind. Sie treten oft erst nach der Produktion während des regulären Betriebs auf und sind daher für über 90% der Wartungskosten verantwortlich. Man unterscheidet zwischen Aussetzern (*engl.* Intermittend Faults) und flüchtigen Fehlern (*engl.* Transient Faults).

Flüchtige Fehler wiederholen sich nicht, normalerweise werden sie durch α-Strahlung oder von Fluktuationen der Spannungsversorgung verursacht. Sie können nicht repariert werden, da der Hardware kein physikalischer Schaden zugefügt wurde. Sie sind eine Hauptursache für Fehler in Halbleiterspeicherbausteinen.

Aussetzer wiederholen sich mehr oder weniger regelmäßig. Solche Fehler können wegen losen Steckverbindungen, teilweise defekten Komponenten, Timing-Fehlern, metastabilen Zuständen oder schlechtem Design der Platine bzw. der Halbleiterbausteine auftreten. Aussetzer aufgrund der Alterung von Halbleitern können sich im Laufe der Zeit in permanente Fehler verwandeln. Genauso können auch Einflüsse aus der Umgebung wie Temperatur, Feuchtigkeit, Vibrationen u.s.w. zu temporären Fehlern führen, wenn das System

gegen diese Einflüsse nicht ausreichend geschützt ist wie z.B. durch Kühlung.

Weil die Aussetzer nur zufällig auftreten, müssen sie mit den Methoden der Wahrscheinlichkeitsrechnung modelliert werden [Lal97].

2.2 Test digitaler Schaltungen

Der Test digitaler Schaltungen dient zur Fehlerdiagnose. Darunter versteht man die Detektion und Lokalisierung der Fehler.

Zur Fehlerdiagnose werden in der Regel eine Reihe von Testsignalen an die Eingänge der Schaltung gelegt und deren resultierende Signale gemessen und überprüft (vergl. Abbildung 2.2). Zur Überprüfung der Ausgangsmuster werden diese mit den korrekten Ausgangsmustern, die vor oder während des Tests erzeugt werden, verglichen.

Der Fehler wird lokalisiert, indem er der jeweiligen defekten Komponente, Leiterbahn, Modul, etc., je nach den Anforderungen, zugeordnet wird. Dazu vergleicht man im allgemeinen die fehlerhaften Ausgangsmuster mit den Mustern, die in der Fehlersimulation erzeugt werden, um daraus auf die Fehlerursache zu schließen. Gegebenenfalls ist es möglich, den Fehler mit optischen oder visuellen Methoden zu lokalisieren, so kann man z.B. defekte Leiterbahnen oder Lötverbindungen mit bloßem Auge oder unter einem Mikroskop erkennen.

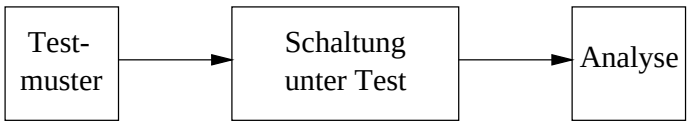


Abb. 2.2. Beim Test einer digitalen Schaltung werden eine Sequenz von Testmustern am Eingang der zu testenden Schaltung angelegt und deren resultierenden Ausgangsmuster überprüft.

Je nach Einsatz eines Tests unterscheidet man zwischen dem Prototypentest, dem Produktionstest und dem Wartungstest. Der Prototyp einer Schaltung wird getestet, um zu überprüfen, unter welchen elektrischen und sonstigen Randbedingungen ihre Funktion gewährleistet ist. Da der Test nur bei einer begrenzten Anzahl von Prototypen durchgeführt wird, ist die Testzeit von untergeordneter Bedeutung. Der Produktions- oder Herstellungstest dient zur Qualitätskontrolle, zur Prüfung der Funktionsfähigkeit. Hier kommt es auf eine kurze Testzeit an, die Lokalisierung der Fehlerursachen ist dabei von untergeordneter Bedeutung und braucht nicht zu erfolgen. Wichtig ist dagegen eine statistische Auswertung, insbesondere Angaben, die Rückschlüsse auf den Herstellungsprozeß erlauben. Der Wartungs- oder "Feld"-Test (*engl.*

field test) überprüft die Funktionsfähigkeit einer Schaltung bei ihrem Einsatz vor Ort, er soll einfach durchzuführen sein und ein schnelles Ergebnis liefern [Woj88].

Kosten eines Tests

Zu den Kosten eines Tests zählen die Erzeugung der Testsignale und die Durchführung des Tests. Man strebt an, mit möglichst wenigen Testmustern die maximal mögliche Fehlerzahl zu finden [Aue96].

Einer der Hauptaspekte bei der Erzeugung der Testsignale ist die Minimierung der Testsequenzlänge. Um das in einem Beispiel zu verdeutlichen, nehmen wir an, ein Schaltnetz mit n Eingängen werde getestet. Wenn man nun an alle Eingänge nacheinander jedes mögliche Testmuster (exhaustive Testmustersequenenz) anlegt, so erhält man eine Sequenz aus 2^n Testmustern. Bei 10 Eingängen und einer Eingangsrate von 50MHz würde ein solcher Test $2^{10}/50\text{MHz} \approx 20\text{ms}$ dauern, bei 50 Eingängen dagegen schon $2^{50}/50\text{MHz} \approx 260$ Tage. Glücklicherweise ist es nicht notwendig, alle möglichen Testmuster zu verwenden, man beschränkt sich auf die Eingangskombinationen, welche die meisten Fehler detektieren können [Lal97]. In den nächsten Abschnitten werden verschiedene Methoden zur Erzeugung von Testmustern vorgestellt.

Die Komplexität einer Schaltung ist ein wichtiger Faktor zum Bestimmung der Testkosten. Je komplexer eine Schaltung ist, um so teurer werden die Entwicklung und Anwendung eines Tests. Daher ist man bestrebt, die Komplexität durch geeignete Maßnahmen schon möglichst früh bei der Entwicklung der Schaltung, beim Design, zu reduzieren [Abr90, Kap.6]. Diesen Vorgang nennt man Design zur Verbesserung der Testbarkeit (DFT), welcher in den nächsten Abschnitten noch genauer behandelt wird.

Je früher ein Defekt in der Produktionsphase gefunden wird, um so weniger Kosten verursacht er. Für diesen Sachverhalt gibt es die 1:10:100:1000-Regel: wenn es DM 0,50 kostet, einen Defekt auf der Chipebene zu bannen, würde es DM 5,- auf der Leiterplattenebene kosten, DM 50,- auf der Systemebene, und schließlich DM 500,-, wenn das Problem erst beim Kunden auftritt [Aue96]. Folglich müssen die Kosten des Tests in Relation zu den Kosten eines nicht detektierten Fehlers gesetzt werden.

Die Kontrollierbarkeit und Observierbarkeit einer Schaltung

Zwei wichtige Faktoren, welche die Komplexität eines Tests bestimmen, sind die Kontrollierbarkeit (*engl.* Controllability) und die Observierbarkeit (*engl.* Observability) einer Schaltung.

Die Kontrollierbarkeit eines Knotens der Schaltung bestimmt, ob durch Anlegen bestimmter Eingangsmuster der Logikwert dieses Knotens kontrolliert werden kann. Ist ein Knoten nicht kontrollierbar, kann er nicht gezielt auf einen Fehler überprüft werden. Hat der Knoten z.B. einen Stuck-at-1-Fehler, so kann dieser nicht durch gezieltes Anlegen von logisch-0 überprüft werden.

Die Observierbarkeit eines Knotens hingegen sagt aus, ob aus dem Ausgangsmuster der Schaltung eindeutig auf den Logikpegel des Knotens rückgeschlossen werden kann. Ein Fehler in diesem Knoten kann sonst bei einem Test unentdeckt bleiben.

Generell kann die Kontrollierbarkeit und Observierbarkeit einer Schaltung erhöht werden, indem Kontroll- und Eingangs-, bzw. Ausgangsleitungen eingefügt werden, ein Aspekt des Designs for Testability (DFT) [Lal97]. Weniger "observierbar" sind z.B. Schaltungen mit redundanter Logik, es sei denn die redundanten Teile der Logik lassen sich durch zusätzliche Testknoten unterscheiden, oder sequentielle Schaltwerke, da ihre Ausgangssignale von den Zuständen früherer Takte abhängig sind [Abr90, Kap.9].

Qualität eines Tests

Die Qualität oder Effizienz eines Tests wird durch eine Fehlersimulation bestimmt, normalerweise im Kontext eines Fehlermodells.

Zur Fehlersimulation benötigt man neben einem Fehlermodell ein Modell der digitalen Schaltung. Wenn bei dem gleichen Eingangsmuster die Simulation des Schaltungsmodells mit und ohne Fehler aus dem Fehlermodell unterschiedliche Resultate liefert, so wird dieser Fehler durch das Muster entdeckt.

Wenn alle möglichen Fehlern mit den Eingangsmustern eines Tests simuliert werden, kann man die Fehlerdeckung (*engl.* Fault Coverage) des Tests bestimmen, das Verhältnis der entdeckbaren Fehler gegenüber allen möglichen Fehlern [Ait99].

$$\text{Fehlerdeckung eines Tests} = \frac{\text{entdeckbare Fehler}}{\text{alle potentiellen Fehler}}$$

Die Fehlerdeckung kann durch eine Erweiterung der Eingangsmustersequenz erhöht werden. Weil damit auch die Testzeit ansteigt, muß evtl. ein Kompromiß zwischen der Erhöhung der Fehlerdeckung und der Testzeit, beides Kostenfaktoren, gefunden werden, um die Effizienz des Tests zu maximieren.

Es ist zu beachten, daß diese Formel nur für solche Fehler gilt, welche in dem verwendeten Fehlermodell enthalten sind. Im allgemeinen sind darüber hinaus noch weitere Fehler möglich, welche nicht zwingend durch den

Test entdeckt werden. Somit gilt die Angabe einer Fehlerdeckung eines Tests immer nur im Kontext eines Fehlermodells, z.B. können, wenn ein Test für Stuck-at-Fehler eine Deckung von 100% besitzt, dennoch genügend andere Fehler vorhanden sein, die der Test nicht findet.

Es sollte nicht übersehen werden, daß in die Qualität auch die Zuverlässigkeit eines Tests mit einspielt. Es ist nicht immer einfach, zwischen Fehlern und Nicht-Fehlern zu unterscheiden. Fehlerhafte Instrumente, Probleme mit elektrischer Aufladung der Meßsonden, Fehler in der Testsoftware und Anomalien der Schaltung führen alle zu ungenauen Tests [Nee99].

2.2.1 Generierung von Tests

Es gibt verschiedene Strategien, um Testmuster zu generieren. Man kann drei Gruppen unterscheiden, strukturelle Tests, welche, wie der Name bereits sagt, auf der Struktur der Schaltung basieren, Funktionstests, bei welchen die korrekte Funktion der Schaltung geprüft wird, und Tests, die (pseudo-)zufällige Testmuster verwenden.

Die vorgestellten Strategien zur Testmustererzeugung haben ihre Grenzen. Testmusterbestimmungsalgorithmen sind nur auf die Gruppe der logischen Fehler beschränkt [Woj88]. Bei dem Einsatz von deterministischen und Zufallsmustern gibt es kein zuverlässiges Modell, um für eine Schaltung präzise vorhersagen zu können, wieviele Testmuster erforderlich sind, um eine bestimmte Größenordnung in der Fehlerdeckung zu erlangen [Abr90, Kap.9].

Strukturelle Tests

Für strukturelle Tests werden aus der Topologie der Schaltung Muster zur Sensibilisierung bestimmter Pfade extrahiert. Die Ausgangsbasis hierfür sind die logische Beschreibung in der Form von Funktionsgleichungen oder einer anderen äquivalenten Form, oder die Struktur der Schaltung [Woj88]. Es gibt verschiedene Verfahren zur Testmustererzeugung unter der Annahme, daß die Schaltung nicht redundant ist und nur ein Fehler gleichzeitig existiert [Lal97].[3] Die Schwäche der Verfahren liegt jedoch darin, daß bei komplexen ICs normalerweise keine detaillierten strukturellen Beschreibungen mitgeliefert werden, welche zur Testgenerierung benötigt werden.

[3] Der interessierte Leser findet diese Verfahren in der Literatur bei z.B. [Abr90], [Lal97] oder [Woj88].

Funktionstests

Funktionstests basieren auf einem funktionellen Schaltungsmodell. Sie sind unabhängig von der tatsächlichen Implementierung. Daher kann ein solcher Test nicht nur zur Suche nach physikalischen Fehlern, sondern auch zur Verifikation des Designs der Schaltung herangezogen werden [Abr90, Kap.8].

Tests mit exhaustiven oder pseudoexhaustiven Testmustern

Ein digitales Schaltnetz mit n Eingängen testet man exhaustiv, indem man alle möglichen 2^n Testmuster anlegt. Bei diesem Test hat man die Garantie, daß alle detektierbaren Fehler, die kein sequentielles Verhalten produzieren, entdeckt werden. Wie schon oben beschrieben, kann ein solcher Test je nach Anzahl der Eingänge und der Taktrate sehr lange dauern, oft zu lange. Außerdem ist dieser Test im allgemeinen nicht bei sequentiellen Schaltwerken anwendbar [Abr90, Kap.11].

Indem man den exhaustiven Test beschränkt, so daß nach wenigeren Fehlern gesucht wird, verkürzt er sich signifikant. Man nennt einen solchen Test einem pseudoexhaustiven Test. Die Beschränkung der Testmuster erreicht man durch verschiedene Formen der Segmentation, also der Unterteilung der Schaltung in Unterschaltungen, die man dann alle exhaustiv testen kann. Die Schaltung kann logisch, z.B. in Steuer- und Dateneingänge, oder physikalisch unterteilt werden, z.B. eine sequentielle Schaltung in die einzelnen Sequenzen. Die Ein- und Ausgänge der Unterschaltungen müssen kontrollierbar und observierbar sein [Abr90, Kap.11].

Tests mit Zufallszahlen

Recht einfach zu erzeugende Testmuster sind Zufalls- oder Pseudo-Zufallszahlen. Pseudo-Zufallszahlen haben viele Charakteristika von Zufallszahlen, sind aber wesentlich einfacher zu erzeugen. Sie haben den Vorteil, daß sie deterministisch sind, wodurch sich Tests wiederholen lassen, da eine Pseudo-Zufallssequenz reproduzierbar ist. Das kann sich bei der Lokalisierung eines Fehlers als eine große Erleichterung erweisen.

Der Hauptnachteil der Zufallszahlen ist, daß eine Testmustersequenz recht lange sein muß, d.h. nicht minimal ist, um einen Test mit hoher Fehlerdeckung zu erreichen. Nach [Abr90, Kap.6] ist eine solche Testmustersequenz viel länger, gewöhnlich zehnmal länger, als eine deterministisch generierte Sequenz zum Test der selben Fehler. Die Kosten für einen solchen Test sind also höher als bei deterministischen Testmustern.

Das zufällige Ansteuern mancher Eingänge einer Schaltung kann eine sinnvolle Funktion gänzlich verhindern, man denke an den zufälligen Wechsel

des Taktsignals oder des Reset-Eingangs. Daher legt man an solche Eingänge während der zufälligen Ansteuerung anderer Eingänge gleichzeitig deterministische Testmuster an. In diesem Kontext spricht man von einem semizufälligen Prozeß [Abr90, Kap.6].

Normalerweise liegt bei Zufallszahlen die Wahrscheinlichkeit, daß an einen Eingang eine logische 1 angelegt wird, bei 50%. Diese Wahrscheinlichkeit kann zur Erhöhung der Fehlerdeckung in 25% oder 75% geändert werden. Dieser Vorgang wird gewichtete Generierung von Testmustern genannt. Die adaptive Testgenerierung verwendet die Resultate einer Fehlersimulation, um die Gewichte zu modifizieren [Abr90, Kap.11].

Den Generator für Pseudo-Zufallsmustern kann man in Hardware z.B. durch ein rückgekoppeltes Schieberegister realisieren, das auch in die Schaltung selbst integriert werden kann. Damit wäre auch schon das Haupteinsatzfeld der Zufallszahlentests angesprochen, nämlich beim Selbsttest hochintegrierter Schaltungen (BIST, siehe unten).

Automatische Testgenerierung

Die Erzeugung von Tests kann sehr zeitaufwendig sein, manchmal erstreckt sie sich auf mehrere Monate für komplexe Designs. In den letzten Jahrzehnten wurden verschiedene Werkzeuge zur Automation der Prozesse entwickelt, um diesen Engpaß in der Zeit von der Entwicklung bis zur Marktreife eines Produkts zu entschärfen. Sie werden unter dem Begriff der automatischen Testgenerierung (ATG) zusammengefaßt. Die ATG umfaßt neben der automatischen Testmustergenerierung (*engl.* Automatic Test Pattern Generation, ATPG) und der Fehlersimulation, der Entwicklung von Testprogrammen und der Fehlerlokalisierung, auch automatisierte Prozesse zum Design zur Verbesserung der Testbarkeit (DFT) und zum Überprüfen von Design-Regeln [Che99][Kap99].

Ein Hauptziel der ATPG ist neben der Verkürzung der Test-Entwicklungszeit auch die Reduktion der Länge der Testsequenzen [Che99]. Bei der ATPG werden aus einem gegebenen Modell der Schaltung unter Zuhilfenahme eines Fehlermodell die Testmuster und, je nach Anforderung, auch die Diagnosemuster erzeugt (Abbildung 2.3) [Abr90, Kap.6].

2.2.2 Automaten zum Test einer Schaltung

Um die Testmuster an die Eingänge einer Schaltung anzulegen und deren Ausgangssignale auszulesen, benötigt man die entsprechenden Apparate.

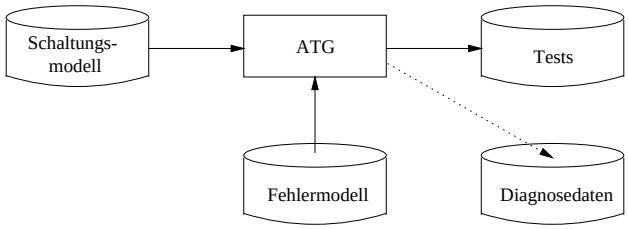


Abb. 2.3. Automatische Testgenerierung [Abr90, Kap.6]

In-Circuit

Beim In-Circuit-Test wird der elektrische Kontakt zu den Komponenten und Leiterbahnen auf einer Platine durch mechanische Prüfköpfe wie bei der Nagelbett-Technik (Abbildung 2.4) hergestellt.

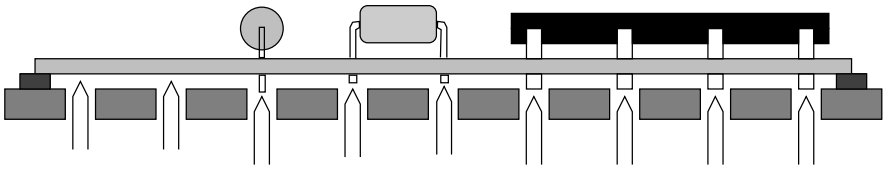


Abb. 2.4. Mechanische Prüfköpfe greifen bei der Nagelbett-Technik von unten auf Meßpunkte einer Platine zu

Durch die zunehmende Miniaturisierung der digitalen Schaltungen stellen sich bei der Nagelbett-Technik mehrere Nachteile ein. Die Abstände der ICs-Pins[4] werden pro IC immer kleiner ($\approx$ 0.3mm) und zahlreicher. Dadurch liegen die Leiterbahnen immer enger zusammen. Durch die geringen Abstände werden die Anforderungen an die Testmechanik immer höher und die Tester damit auch immer teurer[5]. Erschwerend kommt noch hinzu, daß Platinen heutzutage in der Regel zweiseitig in der SMD-Technologie bestückt werden, also nicht mehr alle Komponenten von einer Seite aus mit einem Nagelbrett erreichbar sind. Ebenso sind durch mechanische Probleme Fehlkontakte möglich. Die Technik stößt mit der zunehmenden Miniaturisierung an ihre mechanischen Grenzen [Ble93].

Flying-Probers

Die Flying-Probers-Technik (*deutsch* "fliegende Prüfköpfe") hat sich zunehmend zu einer ernstzunehmenden Konkurrenz der Nagelbett-Technik für be-

[4]Pin: engl. Bezeichnug der IC-Beinchen

[5]Große Tester können bis zu 6 Millionen Dollar kosten, da jedes Test-Pin zwischen 5.000 und 10.000 Dollar kostet.[Nee99]

stimmte Anwendungen entwickelt. Typischerweise besitzen moderne Flying-Probers-Roboter vier oder acht unabhängig bewegbare Prüfköpfe, die an ein In-Circuit-Meßsystem angeschlossen sind. Ihre prinzipiellen Vorteile gegenüber Nagelbetten sind niedrigere Kosten, kürzere Aufbauzeiten, größere Flexibilität und einfachere Wartung. Bei den Flying-Probes ersetzt eine Koordinatenliste die feste Nadelanordnung der Nagelbetten. Die Technik wird eher bei der Prototypentwicklung und bei Kleinserien eingesetzt, die Brauchbarkeit bei anderen Anwendungen wurde bisher noch nicht vollständig untersucht. Die Technik verspricht den Vorteil, daß sich mit zunehmender Miniaturisierung die Prüfköpfe nachführen lassen, um einen guten elektrischen Kontakt herzustellen. So ist sie geeignet, die TAP-Schnittstelle für den Boundary-Scan-Test (siehe unten) anzuschließen. [Rus98][Fer98].

2.2.3 Design zur Verbesserung der Testbarkeit

Wie im letzten Abschnitt beschrieben, wird es mit zunehmender Miniaturisierung immer schwieriger, bestimmte Knoten der Schaltung mit Prüfköpfen zu erreichen. Insbesondere werden immer vielfältigere und komplexere Logikfunktionen in einer einzigen integrierten Schaltung realisiert und wegen der begrenzten Anzahl von Pins können nicht mehr alle zum Test benötigten Schaltungknoten von außen zugänglich sein.

Eine Lösung des Problems ist, schon beim Design der Schaltung Testbarkeitsaspekte zu berücksichtigen, um ein vernünftiges Maß an Observierbarkeit und Kontrollierbarkeit zu erreichen. Man bezeichnet diesen Vorgang als Design zur Verbesserung der Testbarkeit (*engl.* Design for Testability). Im folgenden sind DFT-Techniken aus [Abr90, Kap.9] aufgeführt:

- Einführung zusätzlicher Testknoten (Abbildung 2.5).
  Diese Knoten werden im allgemeinen durch Scan-Pfad-Methoden zugänglich gemacht, welche im nächsten Abschnitt beschrieben sind.

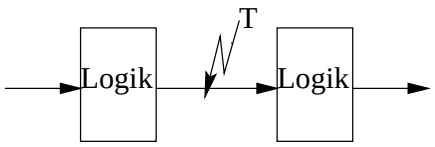


Abb. 2.5. Durch den zusätzlichen Testknoten T ist es möglich, zu unterscheiden, ob ein Fehler in der Logik vor oder nach dem Testknoten auftritt.

- Die Schaltung soll leicht initialisierbar sein, durch z.B. ein Reset-Signal. Die Schaltung kann somit einfach wieder in einen definierten Ausgangszustand gebracht werden, um einen Test zu wiederholen.

- Das Abschalten von internen Oszillatoren und Taktgeneratoren soll ermöglicht werden, um Synchronisationsprobleme zu vermeiden.
  In der Regel wird die Schaltung durch die Testlogik erst getaktet, wenn die Eingangssignale anliegen, und sofort nach einer bestimmten Anzahl von Takten wieder ausgeschaltet, um die Ergebnisse auslesen zu können.

- Die Partitionierung großer Schaltungen in kleinere Unterschaltungen, den sog. Testklassen, vereinfacht die Generierung von Tests und reduziert damit seine Kosten (vergl. Abbildung 6.1 auf Seite 92).
  Ein einfaches Beispiel ist eine Pipeline,[6] die sich in ihre einzelnen Pipelinestufen partitionieren läßt (Abbildung 2.6). Jede Pipelinestufe kann man dann unabhängig voneinander testen. Es ist sehr viel einfacher, für eine einzelne Pipelinestufe einen Test zu generieren, da die Einflüsse der vorhergehenden Pipelinestufen auf das gewünschte Eingangs-Testmuster nicht mehr berücksichtigt werden müssen, und auch die Ausgangsmuster direkt ausgelesen werden können, ohne daß die Änderungen nachfolgender Stufen zurückgerechnet werden müssen. Die Fehlerlokalisierung ist ebenfalls einfacher, da die Suche auf die jeweilige Pipelinestufe beschränkt ist, in welcher der Fehler aufgetreten ist.

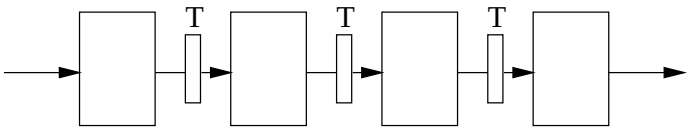


Abb. 2.6. Eine Pipeline wird durch Einfügen zusätzlicher Logik T partitioniert, so daß sich die einzelnen Pipelinestufen unabhängig voneinander testen lassen. Hierfür eignen sich die Scan-Zellen bestens.

- Logische Redundanzen sollten vermieden werden, z.B. durch Einfügung zusätzlicher Testknoten, welche die redundanten Zweige der Schaltung unterscheidbar machen (Abbildung 2.7).

- Verwendung zusätzlicher Logik, um globale Rückschlüsse (*engl.* Feedback) unterbrechen zu können.
  Dadurch vermeidet man ein sequentielles Verhalten der Schaltung, welches den Test erschwert. Am Beispiel mit der Pipeline läßt sich das verdeutlichen. Wenn die letzte Pipelinestufe mit einer früheren verbunden ist, so muß diese Rückschluß-Verbindung unterbrochen werden können, daß man beide Pipelinestufen unabhängig voneinander ohne Berücksichtigung früherer Zustände testen kann (Abbildung 2.8).

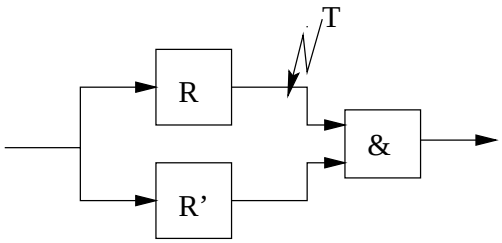


Abb. 2.7. Durch den zusätzlichen Testknoten T werden die redundanten Teile der Schaltung R und R' unterscheidbar gemacht.

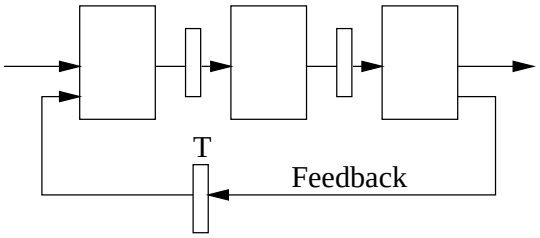


Abb. 2.8. Die Feedback-Leitung wird durch zusätzliche Logik T unterbrochen. Auch hier eignet sich eine Scan-Zelle bestens, die Feedback-Leitung zu kontrollieren und zu messen.

DFT-Techniken tragen zu einer Vermeidung von Testkosten bei, da sie die Komplexität eines Systems herabsetzen und damit sowohl die Zeit zum Entwickeln eines Tests drastisch reduzieren können als auch Testgeräte gespart werden können. Die Nachteile des DFT sind jedoch, daß gewöhnlich der Bedarf der Schaltungsfläche, der Energieverbrauch, die Anzahl der Ein- und Ausgangs-Pins und das Delay der Schaltung durch zusätzliche Register im Pfad steigen. Eine sorgfältige Balance muß gefunden werden [Abr90, Kap.9].

Zukünftig werden die Nachteile mehr und mehr in den Hintergrund treten, nach [Kap99] ist bei der projektierten Nanometer-Technologie zur Chip-Produktion Silizium-Fläche relativ frei verfügbar. Die Gate-Delay-Zeit, die Verzögerung eines Signals durch einen Transistor, ist vernachlässigbar gegenüber der Laufzeit eines Signals auf seiner Leitung.

Beim DFT kann man zwei weite Kategorien unterscheiden, die Scan-Techniken und die in die Schaltung integrierten Selbsttest-Techniken (*engl.* Build In Self Test, BIST), welche in den nächsten Abschnitten vorgestellt

[6] Eine Pipeline (*deutsch* Verarbeitungskette) besteht aus einer Kette von einzelnen Pipelinestufen. Der Ausgang einer Pipelinestufe ist dabei jeweils mit dem Eingang der nächsten verbunden. Jeden Takt verarbeitet eine Stufe ihre Eingangsmuster und legt die Ergebnisse an ihren Ausgang. Das Muster wird also mit jedem neuen Taktimpuls von der jeweils nächsten Stufe verarbeitet.

werden. Beide sind nach [Nee99] eher auf der Struktur der Schaltung basierende Methoden, die anstatt auf die korrekte Funktion auf korrekte logische Signale testen.

Scan-Technik

Die Scan-Technik ermöglicht es, interne, nicht unbedingt von außen über Prüfköpfe zugängliche Knotenpunkte einer Schaltung mit der Testeinrichtung zu verbinden.

An solch einem Knotenpunkt wird hierzu eine Scan-Zelle eingebaut. Die Scan-Zelle ist ein Register, welcher den logischen Wert am Eingang des Knotens zu einem gegebenen Zeitpunkt speichern, einen bestimmten Wert an den Ausgang des Knotens legen, oder den Eingang transparent auf den Ausgang des Knotens schalten kann. Entspricht der Kontenpunkt selbst schon einem in der Schaltung vorhandenen Register, so kann dieses zu einer Scan-Zelle erweitert werden.

Zum Zugriff auf die Scan-Register werden diese seriell zu einem Schieberegister, Scan-Pfad genannt, verbunden, durch welchen die Registerinhalte seriell ein- und ausgelesen werden können (Abbildung 2.9) [Abr90, Kap.9].

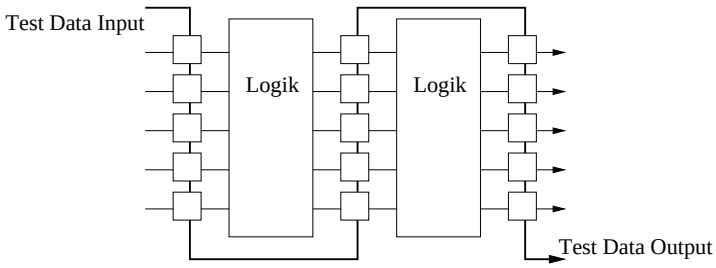


Abb. 2.9. Die Scan-Zellen sind in die Ein- und Ausgänge der Logikschaltungen eingebaut und zu einem Scan-Pfad aneinandergereiht, welcher über den *Test Data Input* beschrieben und über den *Test Data Output* gelesen wird.

Um die verschiedenen Modi der Scan-Zellen und das Ein- und Auslesen über die Scan-Pfade zu kontrollieren, benötigt man eine Steuerlogik, diese wird Hardcore (*deutsch* "harter Kern") genannt. Der Hardcore sollte vor dem Test der Schaltung, soweit wie möglich, ebenfalls getestet werden.

Im Gegensatz zu einem In-Circuit-Test wird der Test immer bei angelegter Betriebsspannung durchgeführt, da die ICs mit ihren integrierten Testschaltungen sonst nicht funktionieren würden.

Boundary-Scan-Test Standard

Neben einer individuell auf eine Schaltung zugeschnittenen Implementierung der Scan-Zellen und des Scan-Pfads, auch "custom" oder "ad hoc" Scan-Design bezeichnet, hat sich der Boundary-Scan-Test-Standard nach IEEE[7] Std. 1149.1 etabliert.[8]

Abbildung 2.10 zeigt zwei ICs mit integrierten Boundary-Scan-Registerzellen. Die Zellen sind in die Verbindung zwischen der internen Logik und den Ein- und Ausgangs-Pins des ICs eingefügt[9] und zu einem seriellen Scan-Pfad verbunden.

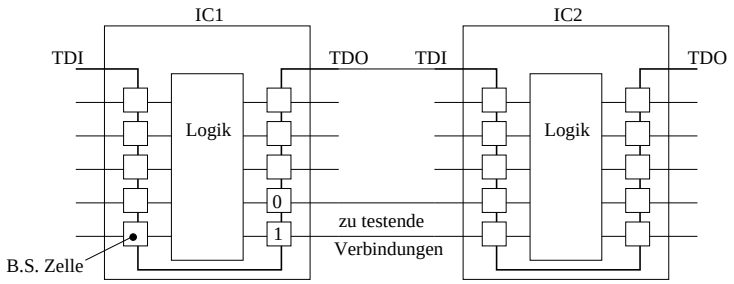


Abb. 2.10. Zwei mit Boundary-Scan-Zellen ausgestattete ICs. Sie sind mit 2 Leitern verbunden, welche getestet werden sollen. In 2 Ausgangs-Boundary-Scan-Zellen des 1. IC wurde eine 0 und eine 1 hineingeschrieben, die in den verbundenen Eingangzellen des 2. ICs gefunden werden, falls die Verbindung fehlerfrei funktioniert.

Der Test von Verbindungen zwischen zwei ICs mit Boundary-Zellen läuft folgendermaßen ab (vergl. Abbildung 2.10):

1. Serielles Schieben der Test-Stimuli vom Test-Daten-Eingang (TDI) des Sender-ICs entlang der Registerzellen des Scan-Pfads bis zu den Zellen, die an zu testenden Ausgangs-Pins angeschlossen sind (Shift).

2. Die Ausgänge der Zellen mit den neuen Registerinhalten aktualisieren (Update).

3. Einlesen der Logikpegel der Leitungen an den Eingangs-Pins in die Registerzellen des Empfängerbausteins (Capture).

4. Serielles Schieben der Registerinhalte zum Test-Daten-Ausgang (TDO) des Empfänger-ICs in die Testeinrichtung zur Analyse der Daten (Shift).

[7] IEEE: Institute of Electrical and Electronics Engineers

[8] Der Standard ist nach der Joint Test Action Group, welche ihn entwickelt hat, auch als JTAG-Standard bekannt.[Lat99a]

[9] daher der Name "Boundary", engl. Rand;

Eine solche Verbindung setzt sich in der Regel, wie in der Abbildung 2.11 zu sehen ist, aus in den unterschiedlichsten Techniken gefertigten Verbindungsstücken zusammen, welche alle Fehler verursachen können. Dies unterstreicht die Wichtigkeit eines solchen Tests der externen Verbindung (EXTEST). Entsprechend funktioniert der Test der internen Logik (INTEST). Teststimuli werden an den Eingängen der internen Logik angelegt und an den Ausgängen gemessen.

Neben dem EX- und INTEST sind weitere Boundary-Scan-Modi möglich. Dazu gehört der Modus zum normalen Betrieb der Kernlogik des Systems ohne Beeinflussung, bei dem die Eingänge der Zellen transparent auf die Ausgänge geschaltet werden (BYPASS). Im SAMPLE-Modus ist es während des Bypass-Betriebs der Zelle möglich, die Daten zu einem bestimmten Zeitpunkt in die Register zu übernehmen, ohne den Bypass-Datenfluß zu stören. Der optionale Modus RUNBIST steuert den Selbsttest des Boundary-Scan-Systems.[10]

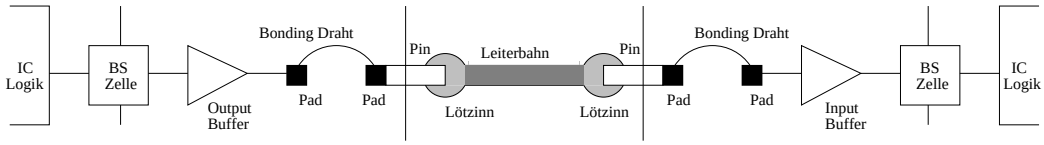


Abb. 2.11. Alle Elemente einer Verbindung zwischen den Boundary-Scan-Zellen zweier ICs

Die Testabläufe der verschiedenen Modi steuert eine Zustandsmaschine (TAP Controller), welche extern über den Test Access Port (*deutsch* Test-Zugriffs-Schnittstelle, TAP) kontrolliert wird. Der TAP besteht aus vier, bzw. bei einem optionalen Reset-Anschluß fünf Pins, zwei davon sind der TDI und TDO, die beiden anderen die Eingänge der Steuerdaten und des Test-Takts. Es werden also sehr wenig zusätzliche Pins benötigt, um die Testfunktionen in den IC zu integrieren.[11]

Durch die Standardisierung der Scan-Technik ist es möglich, bei der Entwicklung digitaler Schaltungen ICs mit integrierten Boundary-Scan-Möglichkeiten verschiedener Hersteller einzusetzen und auf standardisierte Testmethoden und Testausrüstung zurückzugreifen, wie z.B. die ATG oder ATPG (siehe oben) oder auch eigene Programmiersprachen zum Design oder zur

[10] Darüber hinaus sind weitere Modi möglich, die der interessierte Leser z.B. in [Aue96], [Ble93] oder [Par98] nachlesen kann.

[11] Aktuelle PLDs (Programmable Logic Devices) können zusätzlich über die TAP-Schnittstelle programmiert werden. Dadurch sind sie auch schnell umprogrammierbar, z.B. läßt sich die "normale" Logik kurzzeitig durch eine speziell für einen Test entwickelte Logik ersetzen.[Lat99b]

Steuerung eines Tests, wodurch die Testentwicklungszeit verkürzt werden kann [Ble93].

Gleichzeitig ist auch die Zeit zur Durchführung eines Boundary-Scan-Tests kürzer. Wenn beispielsweise die Diagnose bei einem konventionellen In-Circuit-Test noch 40 Minuten in Anspruch nimmt, dauert sie bei einem Boundary-Scan-Test nur noch zwei Minuten [Aue96].

BIST-Technik

Bisher haben wir Techniken kennengelernt, bei welchen externe Testgeräte zum Einsatz kommen. Der Build-in Self-Test (*deutsch* "eingebauter" Selbsttest, BIST) ist eine Design-Technik, bei der in die Schaltung Einheiten integriert werden, so daß sie sich selbst testen kann.

BIST-Techniken werden in zwei Kategorien unterteilt, den Online-BIST und den Offline-BIST. Beim Online-BIST wird während des normalen Betriebs der Schaltung getestet. Hier unterscheidet man wiederum zwischen dem "gleichzeitigen" (*engl.* concurrent) Online-BIST, bei dem der Test simultan mit den normalen funktionellen Operationen der Schaltung stattfindet, und dem "nicht-gleichzeitigen" Online-BIST, bei dem während Unterbrechungszeiten der Schaltung getestet wird, z.B. wenn sich das System in einem Idle-Zustand befindet. Der "nicht-gleichzeitige" Testprozeß kann jederzeit unterbrochen werden, so daß das System seine normale Operation wieder aufnehmen kann. Der Offline-Test wird dagegen nur ausgeführt, wenn das System nicht seine normale Funktion ausübt.

Abbildung 2.12 zeigt die Elemente des BIST. Der Teil der Schaltung, der benötigt wird, um den BIST auszuführen, wird Hardcore genannt. Der Hardcore ist gewöhnlich sehr schwer explizit zu testen, und wenn er fehlerhaft ist, versagt auch der Test der restlichen Schaltung. Daß der Hardcore fehlerhaft ist, entdeckt man daher oft sehr schnell, eine genauere Diagnose oder Fehlerlokalisation kann aber nicht durchgeführt werden. Daher wird der Hardcore gewöhnlich von einer externen Testeinrichtung getestet, oder ist so aufgebaut, daß er sich durch Verwendung verschiedener Formen der Redundanz selbst testen kann. Normalerweise strebt der Designer an, die Komplexität des Hardcores möglichst klein zu halten [Abr90, Kap.11].

In der Regel werden beim BIST exhaustive, pseudo-exhaustive oder pseudo-zufällige Testmuster verwendet, da diese leicht intern generiert werden können.

Vorteile des BISTs sind die Reduzierung der Notwendigkeit teurer externer Testeinrichtungen und die Möglichkeit, unabhängig von dem Standort der Testeinrichtungen zu sein. Der BIST kann vor Ort ausgeführt werden, sowohl während als auch nach der Produktion beim endgültigen Einsatz der

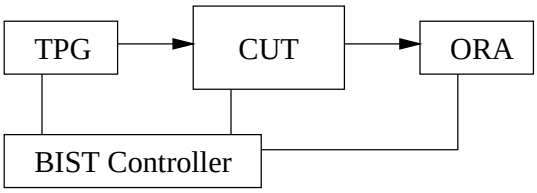


Abb. 2.12. Die zentralen Elemente des BISTs sind der Testmustergenerator (*engl.* Test Pattern Generator, TPG), der Analysator zur Auswertung der Reaktionen auf die Testmuster (*engl.* Output Response Analysator, ORA), die zu testende Schaltung (*engl.* Circuit under Test, CUT), und die Testablaufsteuerung (BIST-Controller) [Abr90, Kap.11].

Schaltung [Abr90, Kap.11]. Nachteile sind der zusätzliche Schaltungsaufwand und damit der zusätzliche Leistungsverbrauch [Aue96].

2.2.4 Stand der Technik: Tests auf dynamische Fehler

Bisher wurden Techniken gezeigt, welche eine digitale Schaltung auf statische Fehler testen. Doch während des Betriebs einer Schaltung können auch von dynamischen Effekten verursachte Fehler auftreten. Dazu zählen z.B. Reflexionen am Ende einer Signalleitung, wie sie bei fehlenden oder falschen Abschlußwiderständen auftreten (vergl. Abbildung 2.1), sowie Design- oder Fertigungsfehler bei Leiterbahnen, die zu einem verstärkten Übersprechen zwischen den Signalleitungen führen können. Dynamische Fehler können ferner durch Spannungsschwankungen verursacht werden, welche durch das gleichzeitige Schalten vieler Transistoren entstehen (*engl.* Ground Bounce) (vergl. Anhang B.3) [Mün00]. Diese dynamischen Effekte können zu größeren Signalverzögerungen und damit zu Delay-Fehlern (Delay: *deutsch* Verzögerung) führen. Darüber hinaus ist es möglich, daß dynamische Effekte im Echtzeitbetrieb der Schaltung Signale nachfolgender Taktzyklen stören (siehe Abbildung 2.13).

Bei einem Test auf statische Fehler ist die Datenrate oft viel kleiner als im Echtzeitbetrieb der zu testenden Schaltung. Die konstanten Testdaten werden oft lange genug an die Eingänge der Schaltung angelegt, so daß sich dynamische Effekte vor der Messung "ausschwingen" und es zu einer Stabilisierung der Ausgangssignale kommt. Insbesondere beim Scan-Verfahren dauert es in der Regel mehrere Taktzyklen, bis die Eingangsdaten eines Tests und die gemessenen Resultate durch das Schieberegister, welches der Scan-Pfad bildet, geschrieben werden, so daß dynamische Einflüsse aus früheren Taktzyklen, die beim Echtzeitbetrieb zu Störungen führen würden, wahrscheinlich verschwunden sind und nicht gefunden werden. Ein "statischer"

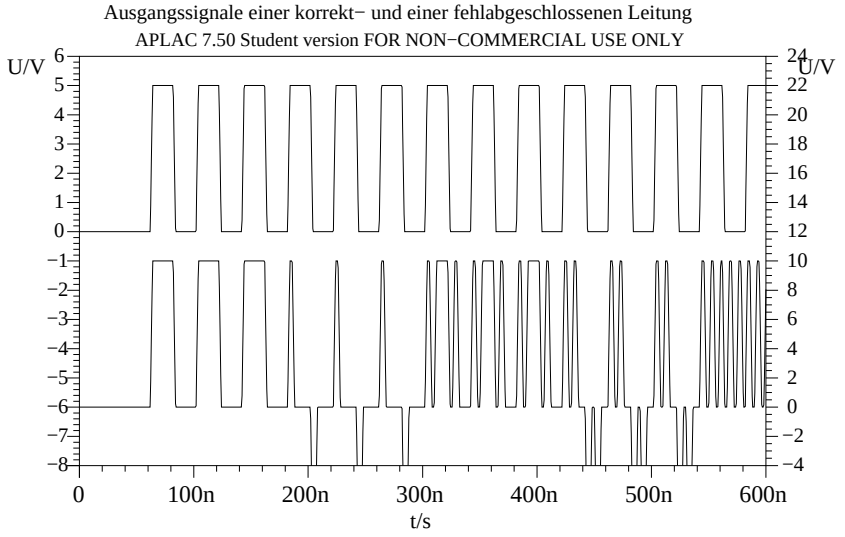


Abb. 2.13. In dieser Simulation wurde ein oszillierendes TTL-Signal an eine Leitung angelegt und an ihrem Ausgang "gemessen" (vergl. Anhang B.1). In der oberen Hälfte des Diagramms sieht man die Messung an der am Ausgang korrekt abgeschlossenen Leitung. Unten sieht man das gleiche Signal auf der gleichen Leitung ohne Abschlußwiderstand. Wie man deutlich erkennen kann, wird das Signal ab dem vierten Signalpuls durch Reflexionen früherer Pulse am fehlabgeschlossenen Leitungsende gestört.

Test eignet sich also nicht ohne weiteres, dynamische Fehler zu finden, die im Echtzeitbetrieb der Schaltung auftreten.

Effektive Tests auf dynamische Fehler werden benötigt, insbesondere da die Betriebsfrequenz neu entwickelter digitaler Schaltungen immer weiter steigt, so daß kleine Variationen in den Signalverzögerungen bei niedrigen Frequenzen heutiger Schaltungen die Operationen zukünftiger schnellerer Schaltungen verhindern [Ait99][Che99]. So sind z.B. für die geplante Produktion von ICs in der Nanometer-Technologie effektive Echtzeit-Tests ein absolut kritischer Faktor. Das Problem wird in der Forschung studiert, Lösungen sind nicht unmittelbar in Sicht:

> "The ability to generate effective at-speed tests [...] is absolutely critical to successful production of integrated circuits in nanometer technologies. [...] Researchers are studying this problem, but no solutions are immediately forthcoming [Ait99]."

An das Messen analoger Signale werden besondere Anforderungen gestellt. Prüfköpfe verändern durch ihre eigene Induktivität und Kapazität die der

Schaltung und verändern somit das dynamische Verhalten [Par98]. Die Tester und Meßgeräte müssen schnell genug sein, um im Echtzeitbetrieb der Schaltung Teststimuli anlegen und deren Resultate messen zu können. Da digitale Schaltungen meistens mit ihrer maximal möglichen Frequenz getaktet werden, werden hier sehr hohe Anforderungen an die Testinstrumente gestellt. Hinzu kommt noch, daß das Abtasttheorem beachtet werden muß, welches besagt, daß ein periodisches Signal mindestens mit der doppelten Rate der Signalfrequenz abgetastet werden muß, um es korrekt rekonstruieren zu können [Hin92]. Die Tester müssen demnach mindestens mit der doppelten Taktrate der Schaltung messen können.

Zur Detektion von Delay-Fehlern ist die In-Circuit-Technik aus diesen Gründen nicht geeignet [Nad99]. Die nachfolgenden Vorschläge beschreiben Erweiterungen des Scan-Tests, so daß auch Delay-Fehler detektiert oder analoge Elemente in der Schaltung gemessen werden können.

Delay-Test

Ein Delay-Fehler (siehe Kapitel 2.1.3) in einer Schaltung mit kombinatorischer Logik kann durch eine Sequenz zweier Testmuster detektiert werden. Der Vorgang soll anhand des Modells in Abbildung 2.14 verdeutlicht werden.

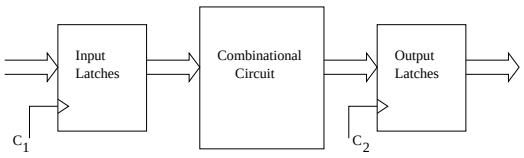


Abb. 2.14. Hardware Modell für den Test auf Delay-Fehler [Lal97]

Zuerst wird das sogenannte Initialisierungs-Muster in die Eingangs-Latches geladen. Nachdem die Schaltung sich stabilisiert hat, wird das zweite Muster, das Übergangs- oder Ausbreitungsmuster, mit dem Taktsignal C_1 in die Eingangs-Latches eingelesen. Nach einer Periode, die gleich oder größer ist als die Zeit, welche die Ausgangssignale der Logik benötigen, um sich zu stabilisieren, werden die Ausgangsmuster mit dem Takt C_2 in die Ausgangs-Latches geladen. Ein Delay-Fehler wurde gefunden, wenn das Ausgangsmuster nicht mit dem erwarteten Muster übereinstimmt [Lal97].

Durch Wiederholung des Tests und Variation der Periode zwischen den Taktimpulsen C_1 und C_2 könnte das Ausgangssignal über das entsprechende Zeitintervall hinweg beobachtet werden. Andere denkbare Fehler durch Störungen, die nur im Echtzeitbetrieb auftreten, z.B. durch Übersprechen von benachbarten Signalleitungen, können durch diesen Test nicht gefunden werden.

Der analoge Boundary-Scan Standard

Mit dem analogen Boundary-Scan Standard IEEE Std. 1149.4 wurde ein Standard definiert, mit dem es möglich ist, neben digitalen Tests auch analoge Messungen durchzuführen. Der analoge Standard ist eine kompatible Erweiterung des digitalen Boundary-Scan Standards.

Das in der Praxis extrem schwierige Problem, bei hohen Frequenzen zu messen, was für Echtzeit-Funktionstests notwendig ist, wurde bei dem Standard bewußt vermieden. Der Standard verzichtet auf die Möglichkeit, Echtzeit-Funktionstests durchführen zu können, und beschränkt sich auf Messungen vom Gleichstrom an aufwärts bis ca. 1 MHz,[12] was ausreicht, wenn nur Produktionsfehler in Betracht gezogen werden sollen [Par98]. Zu den Produktionsfehlern zählen z.B. fehlende oder falsch bestückte analoge Komponenten wie Abschlußwiderstände.

Die digitale Boundary-Scan-Zelle wird durch eine analoge Boundary-Scan-Zelle, genannt Analog-Boundary-Modul (ABM), ersetzt, welche sowohl die digitalen Testfunktionen nach IEEE Std. 1149.1 unterstützt, als auch zusätzliche Ressourcen für analoge Messungen enthält. Ein mit ABM-Zellen ausgestatteter IC hat neben der digitalen TAP-Schnittstelle zwei weitere Pins, die analogen Test-Pins AT (siehe Abbildung 2.15). Sämtliche ABMs sind parallel mit den beiden AT-Pins verbunden. Das ABM kann seine Pins mit einer AT-Leitung verbinden. Auf diese Art und Weise können an die AT-Pins angeschlossene analoge Testinstrumente wie Signalgeneratoren und Meßinstrumente zur Durchführung analoger Tests und Messungen direkt mit IC-Pins verbunden werden. Um die Eigeninduktivität und Kapazität der analogen Meßleitungen zu ermitteln, können die beiden Leitungen in der jeweiligen ABM kurzgeschlossen werden.

Der analoge Boundary-Scan-Standard eignet sich, eine Schaltung auf defekte analoge Komponenten wie defekte Abschlußwiderstände zu testen. Messungen bei hohen Frequenzen sind nicht möglich. Wie auch beim Delay-Test können Fehler, die nur im Echtzeitbetrieb auftreten, durch diesen Test nicht gefunden werden.

2.2.5 Beispiel aus einer industriellen Entwicklung und Produktion

In diesem Abschnitt sollen als Beispiel die Tests bei der Entwicklung und Produktion eines kommerziellen Produkts, eines PC-Mainboards, beschrieben werden. Die Informationen stammen von der Firma Elito-Epox.

[12] Mit In-Circuit-Testern wird bei Frequenzen kleiner als 10 kHz gemessen [Par98].

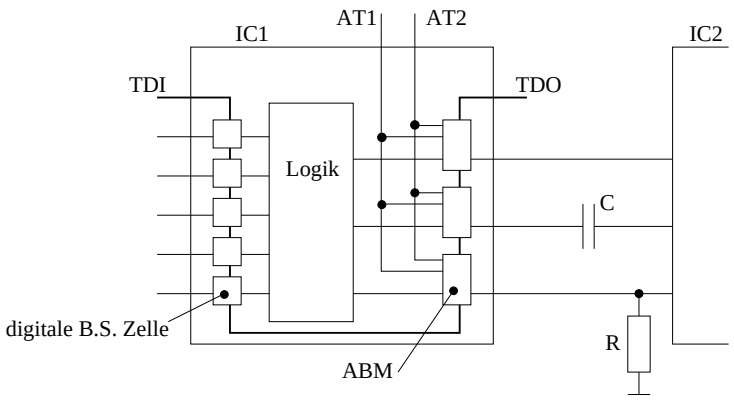


Abb. 2.15. Im IC1 sind drei analoge Boundary-Module (ABM) zu sehen, welche alle mit den beiden Pins AT1 und AT2 des analogen Test-Ports verbunden sind, an welche analoge Signalgeneratoren oder analoge Meßinstrumente angeschlossen werden können. Das ABM bietet die Möglichkeit, eine der AT-Leitungen mit dem Ausgangs-Pin des ICs zu verbinden und dadurch angeschlossene analoge Komponenten, hier ein Kondensator C und ein Widerstand R, zu testen.

Die Platine des Mainboards ist ca. 25×37 cm^2 groß (ATX-Formfaktor) und besteht aus vier Leiterbahn-Lagen (Die TFU besteht aus 14 Lagen). Sie ist doppelseitig mit SMD-Bausteinen und einseitig mit Elektrolyt-Kondensatoren und Sockeln für den Prozessor, für die RAM-Module und die Steckkarten sowie mit Steckkontakten für den IDE-Bus und das Floppy-Laufwerk bestückt.

Prototypentwicklung eines PC-Mainboards

1. *Simulation:* Die Schaltung wird mit einem IBIS-Verhaltensmodell[13] simuliert. (Dauer mind. 1 Woche)

2. *Debugging:* Am Prototyp untersucht man das Timing, das Signalverhalten und Störungen in der Stromversorgung mit Meßinstrumenten wie Oszilloskopen und beseitigt fehlerhaftes Verhalten. Meistens benötigt man zwei bis drei Revisionen des Prototyps vor der Serienproduktion. (Dauer ca. 3-5 Tage pro Revision)

3. *Dauertest:* Die Hardware wird mit kommerzieller Testsoftware im Dauerbetrieb geprüft und Fehler korrigiert. Bevor der Prototyp an ein Test-Laboratorium weitergegeben wird, führt man einen Kompatibilitäts-Vortest mit verschiedenen Steckkarten aus.

4. *Test der Kompatibilität:* Ein Test-Laboratorium führt einen ausführli-

chen Test der Kompatibilität mit diversen Betriebssystemen, darunter Linux und Microsoft Windows 98/NT/2000, der vollen Bandbreite von Steckkarten und anderer Peripherie durch. (Dauer für Punkt 3-4 ca. 2 Wochen)

5. *Probebetrieb der Serienproduktion:* Bevor man die Massenproduktion startet, testet man sie mit einer kleinen Serie. (Dauer ca. 10 Tage)

Tests nach der Serienproduktion des PC-Mainboards

1. *Optischer Test:* Ein Fotosensor vergleicht bei der Produktion jedes Mainboard optisch mit einer Vorlage.

2. *In-Circuit-Test:* Bei einem In-Circuit-Test werden über mechanische Kontakte Testsignale an die Hardware angelegt und deren Resultate geprüft.
   Im Falle eines Mainboards ist ein In-Circuit-Test im Gegensatz zu einem Boundary-Scan-Test angebracht, da die Hauptaufgabe des Mainboards in der Verbindung der über Sockel und Steckkontakte verbundenen Peripherie besteht. Insofern enden viele Leiterbahnen auf der Platine an diesen Sockeln und Steckkontakten, welche keine Möglichkeiten für einen Boundary-Scan-Test vorweisen.
   dort und nicht an den Ein- und Ausgängen von ICs.

3. *Funktionstest:* In einem speziellen Aufbau wird die Funktion des Mainboards geprüft, als Software kommen ein spezielles BIOS-Betriebssystem[14] und kommerzielle Testsoftware für PCs zum Einsatz. (Dauer für Punkt 1-3 ca. 2 min)

2.3 Simulation digitaler Schaltungen

Die Simulation einer digitalen Schaltung wird anhand eines Modells der Schaltung durchgeführt. In diesem Abschnitt werden die theoretischen Grundlagen einer Simulation kurz dargelegt.

[13]IBIS (Input Output Buffer Information Specification) ist eine Spezifikation für Verhaltensmodelle, um das Signalverhalten der Ein- und Ausgänge einer digitalen Schaltung zu beschreiben. IBIS-Modelle der eigenen Produkte werden gerne von Halbleiter-Herstellern publiziert, da sie keine internen Prozeß- und Architekturinformationen preisgeben. Damit ermöglichen sie dem Kunden, mit diesen Produkten aufgebaute digitale Schaltungen zu simulieren.

[14]BIOS: Basic Input/Output System

2.3.1 Das Modell einer digitalen Schaltung

Das Modell spielt eine zentrale Rolle im Design, der Produktion und beim Test eines digitalen Systems. Die Repräsentation des Systems bestimmt die Art und Weise, wie man es zur Überprüfung seiner Korrektheit simuliert, wie man es unter Anwesenheit von Fehlern simuliert oder wie man Tests generiert.

Durch den Vorgang der Modellbildung werden Systeme in Modelle abgebildet. Die Modelle sind selbst wiederum Systeme, die aber die Elemente und Relationen des Ursprungssystems in veränderter Weise darstellen. Beim Übergang vom Ursprungssystem oder Original zum Modell findet eine Abstraktion und Idealisierung statt. Ein Modell stellt das Original also vereinfacht dar. Dadurch wird die Untersuchung komplexer Systeme erst handhabbar [Pag91].

Modelle lassen sich nach der Art der Zustandsübergänge klassifizieren. Beim statischen Modell treten keine Zustandsänderungen auf, beim dynamischen Modell dagegen sind sie zeitabhängig. Je nachdem, ob sich die Zustandsvariablen mit der Zeit kontinuierlich oder sprunghaft, also zu auf der Zeitachse diskret verteilten Zeitpunkten, ändern, spricht man von einem kontinuierlichen oder zeitdiskreten Modell. Deterministisch heißt ein Modell, wenn seine Reaktion auf eine bestimmte Eingabe, ausgehend von einem bestimmten Zustand, eindeutig festgelegt ist. Wenn dies nicht der Fall ist, d.h. wenn sich die Reaktionen des Modells nur durch Wahrscheinlichkeitsverteilungen beschreiben lassen, nennt man es stochastisch [Pag91].

Unabhängig vom Grad der Abstraktion kann ein digitales Modell oder eine Schaltung als eine Art Black-Box angesehen werden, deren Eingangsinformationen auf die Ausgänge abgebildet werden. Ein Funktionsmodell beschreibt diese Abbildung auf die Ausgangswerte als Funktion der Eingangswerte. Ein Verhaltensmodell berücksichtigt neben der Funktion zusätzlich noch die zeitliche Beziehung. Ein strukturelles Modell beschreibt diese Black-Box als eine Ansammlung kleinerer Boxen, den Komponenten oder Elementen, deren Ein- und Ausgänge miteinander verbunden sind. Ein strukturelles Modell ist oft hierarchisch aufgebaut, wobei die Basiselemente Funktions- oder Verhaltensmodelle sein können[Abr90, Kap.2].

2.3.2 Die Simulation

Von einer Simulation spricht man im allgemeinen, wenn bei einer Systemanalyse ein Modell an die Stelle des Originalsystems tritt und Experimente am Modell durchgeführt werden. Ist eine hinreichend korrekte Abbildung zwischen Original und Modell gesichert, lassen sich die Abläufe des rea-

len, dynamischen Systems im Modell nachvollziehen und Kenntnisse über das Modellverhalten sammeln, die in gewissen Grenzen Rückschlüsse auf das Verhalten des Originals erlauben [Pag91].

Die Logiksimulation ist ein Werkzeug, um das Design der digitalen Schaltung zu prüfen, um ihre Brauchbarkeit und ihre Grenzen zu untersuchen, um zu prüfen, ob sie das spezifizierte Verhalten zeigt, sowohl in ihrer Funktion als auch im Timing. Die Überprüfung des Designs führt man durch, indem man die Resultate der Simulation mit den nach der Spezifikation erwarteten Ergebnissen vergleicht. Nach [Abr90, Kap.3] gibt es keinen formalen Weg, solch einen Vergleichstest zu generieren. Die Generierung der Stimuli ist im generellen ein Prozeß, der stark von der Intuition des Designers und von seinem Verständnis des Systems abhängt. Ein System, das den Test besteht, zeigt, daß es sich nur unter den angewandten Testmustern korrekt verhält, man kann also nur eine partielle Korrektheit des Designs beweisen.

Die Simulation ersetzt somit einen Prototyp der Hardware durch das Software-Modell, welches man einfacher analysieren und modifizieren kann. So ermöglicht es die Simulation beispielsweise, auch Signale zu testen, auf die man in der Hardware nicht zugreifen kann [Abr90, Kap.3].

2.4 Zusammenfassung

Fehler in digitalen Schaltungen: Fehlermodelle charakterisieren die Effekte physikalischer Fehler durch logische Fehler und ermöglichen damit eine mathematische Behandlung.

Mit dem Stuck-at-Modell kann man eine große Bandbreite statischer physikalischer Fehler beschreiben. Durch seine relative Einfachheit gegenüber komplexen Modellen und damit der Möglichkeit einer relativ einfachen Beherrschbarkeit durch computergestützte Anwendungen wird es z.B. zur Fehlersimulation oder zur automatischen Testgenerierung eingesetzt.

Im Gegensatz zu statischen Fehlern verursachen dynamische Effekte ein Verhalten, das nicht oder nur schlecht vorhergesagt werden kann. Zu diesen dynamischen Effekten zählen z.B. die Reflexion von Signalen am Leiterende, das Übersprechen zwischen benachbarten Signalleitungen oder Spannungsschwankungen durch das gleichzeitige Schalten vieler Transistoren. Unvorhersagbares Verhalten bedeutet, daß deterministische Fehlermodelle nicht ausreichen. Sie sind Gegenstand der aktuellen Forschung.

Test digitaler Schaltungen: Der Test digitaler Schaltungen dient zur Fehlerdiagnose. Darunter versteht man die Detektion und Lokalisierung der Fehler.

Zur Fehlerdiagnose werden in der Regel eine Reihe von Testsignalen an die Eingänge der Schaltung gelegt und deren resultierende Signale gemessen und überprüft. Zur Überprüfung der Ausgangsmuster vergleicht man diese mit den korrekten Ausgangsmustern, die vor oder während des Tests erzeugt werden.

Durch seine Entwicklung und Anwendung verursacht ein Test Kosten, welche durch Techniken zur Reduzierung der Komplexität und Erhöhung der Testbarkeit der Schaltung, dem Design zur Verbesserung der Testbarkeit (DFT), minimiert werden können.

Die Scan- und die Build-in-Self-Test-Techniken (BIST) sind beide DFT-Techniken, in welchen beim Design der Schaltung zusätzliche Testmöglichkeiten integriert werden. Dadurch kann man trotz des erhöhten Entwicklungsaufwands Kosten durch eine verringerte Komplexität bei der Generierung und Anwendung von Tests sparen. Durch die Standardisierung des Scan-Tests stehen eine Vielzahl passender Werkzeuge zur Verfügung, wodurch die Test- und Entwicklungszeiten verkürzt werden können. Der BIST ermöglicht es einer Schaltung, sich selbst zu testen, was besonders vorteilhaft ist, wenn am Einsatzort der Schaltung keine Testwerkzeuge zur Verfügung stehen.

Je später ein Fehler im Produktionsprozeß einer Schaltung auftritt, um so höhere Kosten verursacht seine Beseitigung. Um diese Kosten möglichst gering zu halten, muß in den frühestmöglichen Produktionabschnitten getestet werden.

Ein kritischer Faktor, insbesondere für zukünftige Technologien zur Fertigung höher getakteter digitaler Schaltungen, sind die von dynamischen Effekten verursachten Fehler. Die Entwicklung von Teststrategien zur Diagnose dieser Fehler sind ein Schwerpunkt der aktuellen Forschung.

Simulation digitaler Schaltungen: Eine digitale Schaltung ist ein System, das sich als Modell abbilden läßt. Mit einer Simulation anhand des Modells lassen sich Experimente durchführen und damit Rückschlüsse auf die digitale Schaltung ziehen. Mit Hilfe einer Simulation kann man das Design der Schaltung prüfen, eine Fehlersimulation durchführen oder Tests generieren.

Kapitel 3

Aufbau der TFU

Kapitel 1.2 hatte den Aufbau und die Funktion des FLTs zum Thema. Dieses nun behandelt den Aufbau der TFU (siehe Abbildungen 3.1 und 3.2).[1]

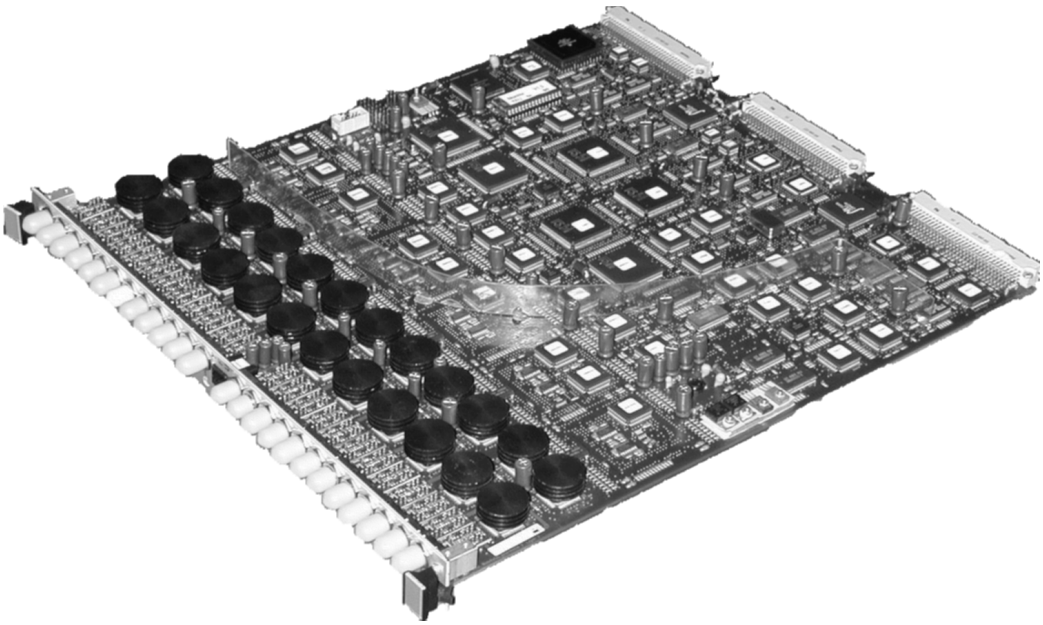


Abb. 3.1. Die Platine einer TFU ist 36×40 cm^2 groß. Sie besteht aus 14 Leiterbahn-Lagen und ist doppelseitig mit SMD-Bausteinen bestückt. Insgesamt gibt es auf einer TFU ca. 20.000 Lötstellen. Vorne links erkennt man auf dem Foto die Frontplatte mit den 24 Anschlüssen für die Glasfaserleitungen vom Detektor und die runden Kühlkörper der Empfänger-ICs. Hinten rechts befinden sich die Steckkontakte zum Anschluß an die Backplane.

[1] Weitere Fotos der TFU findet man im Internet unter
http://www-li5.ti.uni-mannheim.de/mass_par/herab.shtml

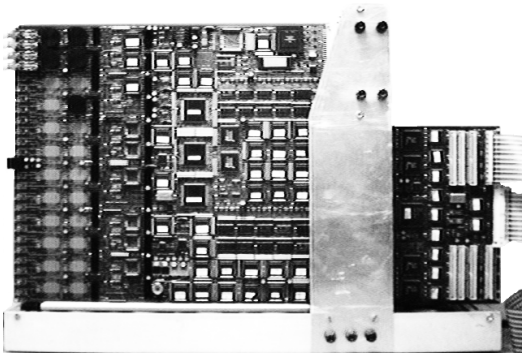


Abb. 3.2. Die TFU auf dem Foto befindet sich in einem kleinen Crate. Dort ist es über Steckkontakte an die Backplane angeschlossen, die es mit dem Netzteil, dem VME-Bus und dem Message-Board verbindet, dem Sender und Empfänger für die Messages, welches (Version 1) man auf dem Foto auf der rechten Seite erkennen kann.

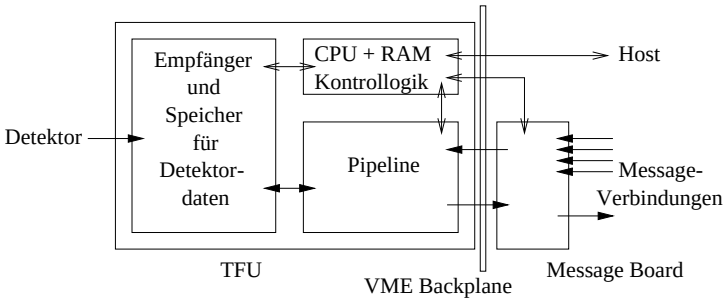


Abb. 3.3. Schematischer Aufbau der TFU (vergl. Abbildung 3.2) [Wol98]

Zuerst wird die Aufgabe einer TFU im Verbund des FLTs beschrieben, dann die Art und Weise, wie der Algorithmus in einem Pipeline-Prozessor implementiert wurde. Der darauf folgende Abschnitt beschreibt die Kommunikations-Schnittstelle zu den anderen FLT-Boards, das sog. Message-Board. Zur Inbetriebnahme und zum Betrieb der TFU-Hardware benötigt man neben der Steuer- und Kontrollogik auch Software zum Ansprechen der Hardware von einem Terminal oder UNIX-Host aus, diese wird am Ende des Kapitels vorgestellt.

3.1 Die Aufgabe einer TFU

Wie schon in Kapitel 1.2 beschrieben wurde, besteht die Aufgabe des dreistufigen Triggers in der Reduktion der Datenrate des Hera-B-Detektors. Die erste Stufe des Triggers, des First-Level-Triggers, sucht hierzu in den Detek-

tordaten nach Spuren von Teilchen, rekonstruiert den Spurverlauf und leitet daraus die Spurparameter ab. Mit diesen kann er die Spuren der Teilchen identifizieren, die von den nachfolgenden Stufen genauer untersucht werden sollen.

Abbildung 3.4 zeigt die Rekonstruktion einer Teilchenspur durch die TFU-Prozessoren. Es ist zu sehen, daß eine einzelne TFU einem bestimmten Bereich einer Detektorebene zugeordnet ist. Liegt die Region-of-Interest (Suchbereich) einer Teilchenspur in diesem Bereich, so erhält die zugehörige TFU den Auftrag, dort nach der Spur zu suchen.

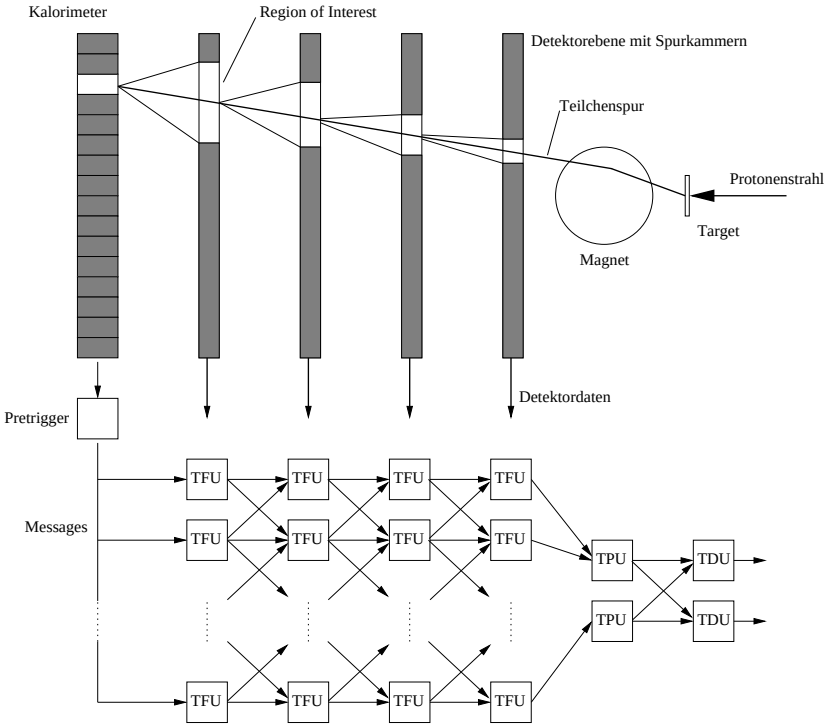


Abb. 3.4. Die Spur eines Teilchens, das von rechts geflogen kommt, wird vom Kalorimeter aus bis zum Target rekonstruiert.

Den Auftrag zur Suche nach einer Teilchenspur erhält die TFU über eine 80 Bit große Message, welche vom sog. Message-Board empfangen wird und welche die zur Identifikation und Rekonstruktion einer einzelnen Spur benötigten Daten enthält. Findet der TFU-Prozessor die Spur in den Detektordaten, verfeinert er die Koordinaten des Spurverlaufs in der Message und sendet diese an die folgenden TFUs bzw. Track-Parameter-Units.

Kann eine TFU die Spur in ihren Detektordaten nicht eindeutig identifizieren, generiert sie für jeden potentiellen Spurverlauf eine Message. Die TFUs der nachfolgenden Detektorebenen verfolgen alle diese Spurverläufe

weiter. Läßt sich eine Spur nicht finden, so löscht die TFU die zugehörige Message.

3.2 Der Speicher für die Detektordaten

Eine TFU analysiert die Daten des jeweiligen Detektorbereichs, dem sie zugeordnet ist. Sie erhält die Daten über 24 Glasfaserleitungen mit einer Länge von je 60 m. Die Datenrate auf einer Leitung beträgt ca. 900 MBit/s und insgesamt auf allen Leitungen ca. 22 GBit/s. Die Daten werden von der TFU im sog. Wire-Memory zwischengespeichert.

Das Wire-Memory ist in sechs identischen ASIC-Chips[2] untergebracht, die neben dem Speicher die Logik zum gleichzeitigen Schreiben der Detektordaten und Lesen durch den Pipeline-Prozessor enthalten.[3]

Das Target wird beim HERA-B-Experiment alle 96 ns mit Protonen beschossen. Um die Detektordaten aus den einzelnen Kollisionen unterscheiden zu können, werden sie mit der sog. BX-Nummer,[4] der fortlaufenden Nummer des auf das Target geschossenen Protonenbündels, markiert.

Das Wire-Memory speichert die Detektordaten der Wechselwirkungen aus den letzten 128 Kollisionen.[5] Der Pipeline-Prozessor liest daraus die zur Spurrekonstruktion benötigten, durch ihre BX-Nummer und die Adresse und Länge der Region-of-Interest selektierten Daten aus.

3.3 Der Pipeline-Prozessor

Den TFU-Algorithmus zur Spurensuche hat man in der Hardware mit einer synchronen Pipeline umgesetzt.[6] Über diesen Pipeline-Prozessor werden nacheinander die vom Message-Board empfangenen Messages verarbeitet. In Abbildung 3.5 ist der Aufbau der Pipeline skizziert.[7] Die Pipeline modifiziert

[2] ASIC: (Application Specific Integratet Circuit) ein IC, der speziell für bestimmte Anwendung entwickelt wurde.

[3] Es handelt sich um ein sog. Dual-Port-Memory.

[4] Bunch-Crossing-Nummer, *deutsch* "Bündel-Kreuzungs-Nummer"

[5] Es hat eine Speicherkapazität von $3\,\textit{Detektorlagen} \cdot \textit{Daten eines Ereignisses} \cdot 128 = 3 \cdot 384\,\text{Bit} \cdot 128 \approx 150$ kBit.

[6] Eine Pipeline (*deutsch* Verarbeitungskette) besteht aus einer Kette von einzelnen Pipelinestufen. Der Ausgang einer Pipelinestufe ist dabei jeweils mit dem Eingang der nächsten verbunden. Jeden Taktzyklus verarbeitet eine Stufe ihre Eingangsmuster und legt die Ergebnisse an ihren Ausgang. Das Muster wird also mit jedem neuen Taktimpuls von der jeweils nächsten Stufe verarbeitet.

[7] Das Fehlen der Pipelinestufen 9, 11 und 14 ist darauf zurückzuführen, daß bei der Endproduktion der TFUs schnellere PLD-Bausteine (s.u.) auf dem Markt erhältlich waren, und

eine Message beim Durchlaufen von Stufe zu Stufe, und neue Parameter, welche die Funktionsergebnisse von anderen Parametern oder von Detektordaten sind, werden ergänzt. Andere Parameter werden gelöscht oder modifiziert.

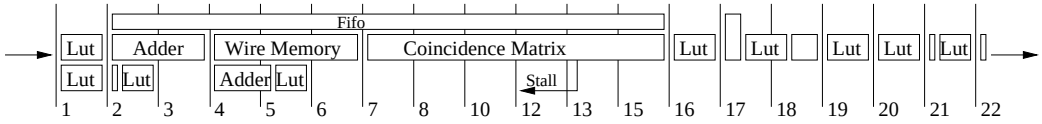


Abb. 3.5. Aufbau des TFU-Pipeline-Prozessors. Die Messages kommen von links vom Empfänger und laufen durch die Pipeline nach rechts zum Sender des Message-Boards.

In den ersten Pipelinestufen werden die Koordinaten der Region-of-Interest, also der Bereich der zugeordneten Detektorebene, durch welchen die gesuchte Spur laufen muß (siehe Kapitel 1.2), errechnet. In den nächsten Stufen werden aus dem Wire-Memory die Detektordaten der Region-of-Interest ausgelesen.

Eine Detektorebene besteht aus drei gegeneinander leicht gedrehten Spurkammerlagen. Die Koinzidenz-Matrix vergleicht die Daten aus diesen drei Lagen miteinander und sucht nach Koinzidenzen, die auf eine Teilchenspur hinweisen. Falls keine Spur zu finden ist, wird die zugehörige Message verworfen. Falls sie die Spur nicht eindeutig identifizieren kann, wird für jeden möglichen Spurverlauf eine neue Message generiert.

Die letzten Pipelinestufen aktualisieren die Koordinaten des Spurverlaufs durch die neuen Informationen aus der Suche nach Koinzidenzen und errechnen, welche TFUs für die Region-of-Interest der folgenden Detektorebene zuständig sind.

Am Ende der Pipeline werden die Messages an den Sender des Message-Boards weitergegeben und von dort aus zu den Message-Empfängern der nächsten FLT-Boards übertragen.

Über die Steuerlogik läßt sich die Pipeline anhalten, starten oder auch nur für eine bestimmte Anzahl von Takten betreiben. Die Logik kann zu Testzwecken in der ersten Pipelinestufe eine Message mit Pseudo-Zufallszahlen generieren und in der letzten Pipelinestufe eine Signatur dieser Message bilden, auf die man zugreifen kann.

Eine Skizze der gesamten TFU-Pipeline ist bei [Glä98b] zu finden. Eine detaillierte Beschreibung ist nicht notwendig für das Verständnis der Funktion der Pipeline. Im folgenden wird daher das Prinzip der Komponenten, aus

so durch kürzere Signallaufzeiten in der Logik der Koinzidenz-Matrix diese Pipelinestufen gegenüber dem Prototyp eingespart werden konnten.

welchen die einzelnen Funktionsgruppen der Pipeline aufgebaut sind, vorgestellt. Daran werden in den folgenden Kapiteln die Simulation und der Test der Pipeline erklärt.

3.3.1 Logikfunktionen innerhalb einer Pipelinestufe

Bestimmte Funktionen der Pipeline sind, wie auch solche der Message-Register der einzelnen Pipelinestufen, aus programmierbaren Logikbausteinen (PLDs)[8] aufgebaut. Abbildung 3.6 zeigt das Prinzip zur Integration der Logik in die Pipeline exemplarisch an einer einzelnen Pipelinestufe.

Ein Taktzyklus läuft hier folgendermaßen ab. Zu Beginn befinden sich die Werte der einzelnen Message-Parameter im ersten Pipelineregister. Die logischen Werte der Registerinhalte liegen unmittelbar an den Ausgängen des Registers. Die Ausgangssignale werden von der angeschlossenen kombinatorische Logik modifiziert und vom folgenden Pipelineregister mit der nächsten aktiven Taktflanke übernommen.

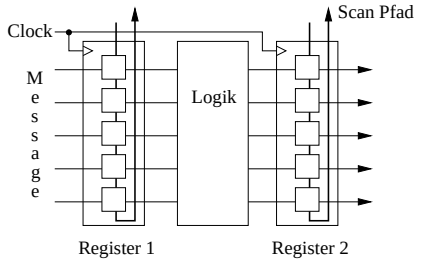


Abb. 3.6. Kombinatorische Logik zwischen den Registern zweier Pipelinestufen

Die Register einer Pipelinestufe sind zu jeweils einem Scan-Pfad zusammengeschaltet, über den sie beschrieben und ausgelesen werden können. Man konnte für die PLD-Bausteine, mit welchen die Logik der Pipeline realisiert wurde, keine Bausteine mit integriertem Boundary-Scan nach dem IEEE-Standard 1149.1 verwenden, da diese mit der benötigten maximal möglichen Taktfrequenz nicht verfügbar sind. Da die Scan-Pfade individuell auf das Design zugeschnitten wurden und sich nicht an den Boundary-Scan-Standard halten, handelt es sich hier um ein sog. "ad hoc" Scan-Design.

Die Register werden immer getaktet, ihre verschiedenen Betriebsmodi werden über Kontrollsignale gesteuert. Während des Betriebs der Pipeline lesen sie mit jedem Takt den neuen Signalwert am Eingang ein. Wird die Pipeline angehalten, so behalten die Register ihren momentanen Wert.

[8]PLD: Programmable Logic Device, engl. programmierbarer Logik Baustein. Die PLD-Hardware enthält elementare Logik-Funktionen, die sich nach bestimmten Regeln zu komplexen Logikfunktionen zusammenschalten lassen.

Beim Schreiben oder Lesen von Daten über einen Scan-Pfad übernehmen sie mit jeder Taktflanke den Wert der benachbarten Registerzelle. Die Ausgänge der Register sind immer aktiv, so daß auch beim Schreiben oder Lesen eines Scan-Pfads die Daten immer an den Eingängen der kombinatorischen Logik anliegen.

Mit kombinatorischer Logik wurden Logikfunktionen wie Multiplexer oder Addierer im Pipeline-Prozessor implementiert.

3.3.2 Die Koinzidenz-Matrix

Die Logik der Koinzidenz-Matrix ist so komplex, daß mehrere aufeinanderfolgende Pipelinestufen zur ihrer Realisation benötigt werden.

Ihre Logik gliedert sich in zwei Teile. Im ersten Teil wird die Suche nach Koinzidenzen in den Detektor-Daten aus dem Wire-Memory durchgeführt. Je nach Typ des Detektors benötigt man eine andere Strategie zur Koinzidenzbildung, die man über die Steuerlogik auswählen kann.

Der zweite Teil der Logik setzt die gefundenen Koinzidenzen in Message-Parameter um. Falls keine Koinzidenzen gefunden werden, wird die Spur verworfen und die zugehörige Message gelöscht. Werden bei der Suche mehrere Koinzidenzen gefunden, die auf mehrere potentielle Spuren hinweisen, so ist die Logik dafür zuständig, zu jeder Spur eine Message zu erzeugen. Die Pipeline vor dem zweiten Teil der Matrix wird dabei so lange angehalten, bis alle neu erzeugten Messages in die der Matrix folgende Pipeline geschrieben wurden. Dies geschieht über das von der Matrix generierte sog. Stall-Signal (siehe Abbildung 3.7). Ist das Stall-Signal aktiv, behalten die Register der vorderen Pipelinestufen beim nächsten Takt-Impuls die momentanen Werte der Message-Parameter.[9]

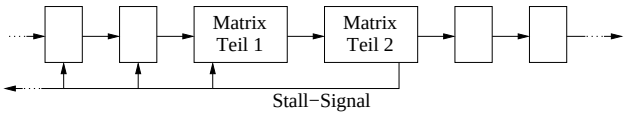


Abb. 3.7. Über das Stall-Signal wird die Pipeline vor dem zweiten Teil der Koinzidenz-Matrix angehalten.

[9]Die Logik erzeugt folglich ein sequentielles Verhalten, da beim Stall-Vorgang in Abhängigkeit von den Parametern der ersten Message die nachfolgenden Messages generiert werden.

3.3.3 Die Scan-Pfade

Die Register aller Pipelinestufen sind bei der TFU zu Scan-Pfaden zusammengeschaltet. Eine Ausnahme ist die Koinzidenz-Matrix, da es dort Pipelinestufen gibt, deren Register nur aus internen PLD-Knoten bestehen. Auf diese muß man zum Test nicht zugreifen können. Falls der PLD defekt ist, kann man das auch über den Scan-Pfad des folgenden Registers im selben PLD-Baustein feststellen.

Es existieren insgesamt 15 Scan-Pfade auf der TFU, die aus einer Kette von 5 bis 94 Registern gebildet werden. Die Register eines einzelnen Pfads liegen dabei in bis zu 9 verschiedenen PLD-Bausteinen [Glä98b].

3.3.4 Die Look-up-Tables

Eine Look-up-Table (LUT) ist eine Tabelle mit den Werten einer Funktion $f(x)$. Bei den FLT-Boards sind diese Tabellen aus RAM-Speicherbausteinen aufgebaut. Unter der Speicheradresse x legt man hierbei den Funktionswert $f(x)$ ab.

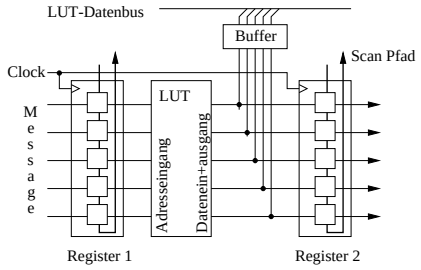


Abb. 3.8. Eine Pipelinestufe mit einer Look-up-Table (LUT)

Der TFU-Prozessor muß trigonometrische Funktionen berechnen. Wird die Berechnung mit einem Mikroprozessor durchgeführt, so dauert sie mehrere Takte. Durch kombinatorische Logik sind die trigonometrische Funktionen in der Regel nicht mit vertretbarem Aufwand implementierbar. Eine LUT hat den Vorteil, daß das Ergebnis einer Funktion nicht berechnet werden muß, sondern sofort nach der RAM-Zugriffszeit, im Falle der TFU nach einem Taktzyklus, bereitsteht. Darüber hinaus ist eine LUT gegenüber festverdrahteter Logik sehr flexibel. Änderungen der Funktion sind leicht durchzuführen.[10] Die TFUs sind identisch aufgebaut, die Zuordnung zu den verschie-

[10] Vorrausgesetzt, die Bit-Breite der Ein- und Ausgängsvariablen wird nicht vergrößert, da hierzu in der Hardware weitere Ein- und Ausgangs-Pins verdrahtet werden müssen oder größere RAM-Bausteine notwendig sind.

denen Detektorbereichen wird u.a. durch unterschiedliche LUT-Funktionen erreicht.[11]

Auf einem TFU-Board befinden sich 37 LUTs, diese besitzen eine Speicherkapazität von insgesamt ca. 2 Megabyte. Da die RAM-Bausteine der LUTs ihren Inhalt bei jedem Ausschalten der Hardware verlieren, müssen sie nach dem Einschalten neu geladen werden.

Ein LUT-RAM beschreibt man, indem man über den Scan-Pfad des Registers am Adreßeingang eine Adresse auswählt und die Daten über den LUT-Datenbus an die Datenschnittstelle des RAMs legt. Gesteuert wird der Vorgang von der Kontrollogik, welche die Register für die Adresse und die Daten und die Register zur Auswahl der gewünschten LUT bereitstellt. Auf die Register greift man über Software und den auf der TFU befindlichen Mikrocomputer zu. Das Lesen einer LUT über den LUT-Datenbus funktioniert entsprechend.

3.4 Das Message-Board

Das Message-Board stellt eine Schnittstelle zur Datenübertragung der 80 Bit großen Messages zwischen den einzelnen FLT-Boards zur Verfügung. Wie in Abbildung 3.3 zu sehen ist, gehört zu jeder TFU ein Message-Board, mit dem es über Steckkontakte auf der Backplane verbunden ist.

Die TFU kann die Messages mit einer Taktrate von 50 MHz verarbeiten. Auf einem Message-Board (siehe Abbildung 3.9) befinden sich je nach Version 4 oder 6 Empfänger, die jeweils alle 40 ns eine Message empfangen können. Die 40 ns entsprechen einer Taktrate von 25 MHz. Da sich dieser Takt von dem des TFU-Pipeline-Prozessors mit 50 MHz unterscheidet und asynchron ist, befinden sich zwischen den Empfängern und dem Pipeline-Prozessor zur Entkoppelung der verschiedenen Takte FIFOs.[12] Diese FIFOs sind auch zur Zwischenspeicherung von Messages notwendig, da je nach Auslastung des FLTs nicht mit jedem Takt eine Message ankommen muß, bzw. mehr Messages gleichzeitig empfangen werden können, als der Pipeline-Prozessor verarbeiten kann. Der Sender ist ebenfalls aufgrund seiner Taktrate von 25 MHz durch ein FIFO vom Pipeline-Prozessor getrennt [Glä98a][Glä99a].

Die Messages in den Empfängerregistern vor den FIFOs können über Scan-Pfade gelesen oder überschrieben werden, genauso kann man auf das Senderregister nach dem Sender-FIFO zugreifen. Von der Steuerlogik aus ist

[11] Der Nachteil einer LUT ist, daß die Größe des Speichers mit 2^n ansteigt, wobei n die Anzahl der Eingangs-Bits ist.

[12] Der First-In-First-Out-Speicher (FIFO) ist, wie der Name schon sagt, ein Speicher, aus dem man die zuerst hineingeschriebenen Daten auch zuerst ausliest.

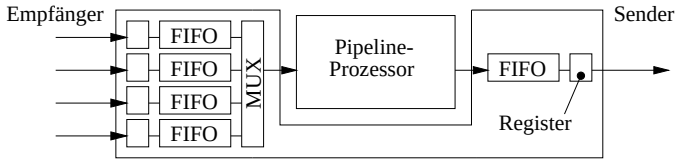


Abb. 3.9. Aufbau des Message-Boards (MUX: Multiplexer zur Auswahl des FIFO-Ausgangs).

es möglich, die einzelnen Empfänger und den Sender ein- und auszuschalten, einzelne Messages zu versenden und den Füllgrad der FIFOs zu kontrollieren.

3.5 Die Steuer- und Kontrollogik

Die Steuerlogik enthält einen eigenen Mikrocomputer mit einem MC 68020-Mikroprozessor der Firma Motorola und 4 MByte RAM-Speicher. Über Register, die über bestimmte Speicheradressen ansprechbar sind, kann man durch auf dem Mikrocomputer laufende Software die Parameter der Pipeline, des Wire-Memorys und des Message-Boards mit seinen Empfängern setzen und kontrollieren. Entsprechend lassen sich Daten in die Scan-Pfade hineinschreiben und wieder herauslesen, und man kann auf den LUT-Speicher zugreifen.

Sensoren erfassen die Temperatur und die Spannung der TFU an verschiedenen Stellen auf der Platine, sie können vom Mikrocomputer ausgelesen werden. Ebenso kann man einen Message-Zähler auslesen und damit die Message-Rate errechnen.

Leuchtdioden auf der Frontplatte des TFU-Boards zeigen den Status der TFU an. Sie signalisieren, ob der Pipeline-Prozessor in Betrieb ist, Messages empfangen oder gesendet werden, Detektordaten empfangen werden und dabei ein Fehler auftrat, auf dem VME-Bus Daten transportiert werden, ein Fehler auf dem Bus des Mikrocomputers auftrat oder der Mikroprozessor wegen eines Fehlers angehalten hat.

3.5.1 Betrieb des Mikrocomputers

Der Mikrocomputer ist mit einer VME-Bus-Schnittstelle und einer seriellen Schnittstelle ausgestattet. Über diese kann er mit einem Host-Rechner oder einem Terminal kommunizieren. Man unterscheidet zwei verschiedenen Möglichkeiten, dort Programme zu starten und auszuführen:

Beim Einschalten des Boards startet automatisch eine Routine zur Kommunikation über die serielle Terminalschnittstelle. Dadurch ist es möglich, Assembler-Programme wir z.B. Testroutinen zu laden und zu starten [Wol98].

Für komplexere Programme ermöglicht es die Laufzeitumgebung TRUN von Hageböke und Wolf, auf dem Mikrocomputer C-Programme auszuführen, die man vorher in gewohnter Weise in einer UNIX-Umgebung entwickelt und mit einem Cross-Compiler übersetzt hat [Hag97b][Wol98].

Ein TRUN-Prozeß wird auf dem VME-Bus-Host ausgeführt, einer Cetia mit einer Power-PC-CPU und dem UNIX-Betriebssystem LynxOS. Er ist für den Start und die Kontrolle eines auf dem Mikrocomputer ausgeführten Programms zuständig.

Die Ein- und Ausgaben des Programms erfolgen über den VME-Bus. Auf der Programmseite ist für die Abwicklung dieser Kommunikation eine speziell angepaßte C-Standardbibliothek zuständig. Die Gegenseite bildet der TRUN-Prozeß auf dem VME-Bus-Host, der die Ein- und Ausgaben an den Benutzer weitergibt. Aus Sicht des Benutzers, und auch aus des Sicht des Betriebssystems, scheint es, als würde das Board-Programm als Prozeß auf dem UNIX-Host ablaufen (siehe Abbildung 3.10).

Für ein einzelnes Board-Programm ist jeweils ein einzelner TRUN-Prozeß zuständig, er wird eindeutig mit einem UNIX-Prozeß identifiziert (siehe Abbildung 3.10) [Wol98].

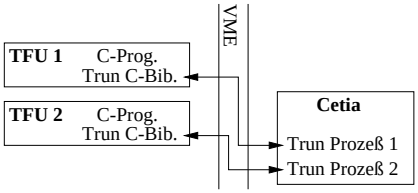


Abb. 3.10. 2 TRUN-Prozesse, die für jeweils 2 Programme auf 2 FLT-Boards zuständig sind und mit ihnen über den VME-Bus kommunizieren

3.6 Die Struktur der Hard- und Software

Abbildung 3.11 zeigt die Struktur der beim FLT eingesetzten Hard- und Software. Die rechteckigen Kästchen stellen die drei Plattformen mit ihren verschiedenen Betriebssystemen dar, darüber ist in den abgerundeten Kästchen die Software eingezeichnet, die auf den jeweiligen Plattformen läuft.

Auf UNIX-Systemen führt man die Simulation des FLTs durch. Dazu verwendet man ein Framework wie das im nächsten Kapitel beschriebene FLTSIM. Die Simulation nutzt die Modelle der TFU und der anderen FLT-Boards, ebenso wie deren LUT-Funktionen. Bei den LUT-Funktionen unterscheidet man Testfunktionen, die speziell für die Tests optimiert wurden, und die im FLT-Betrieb genutzten Funktionen.

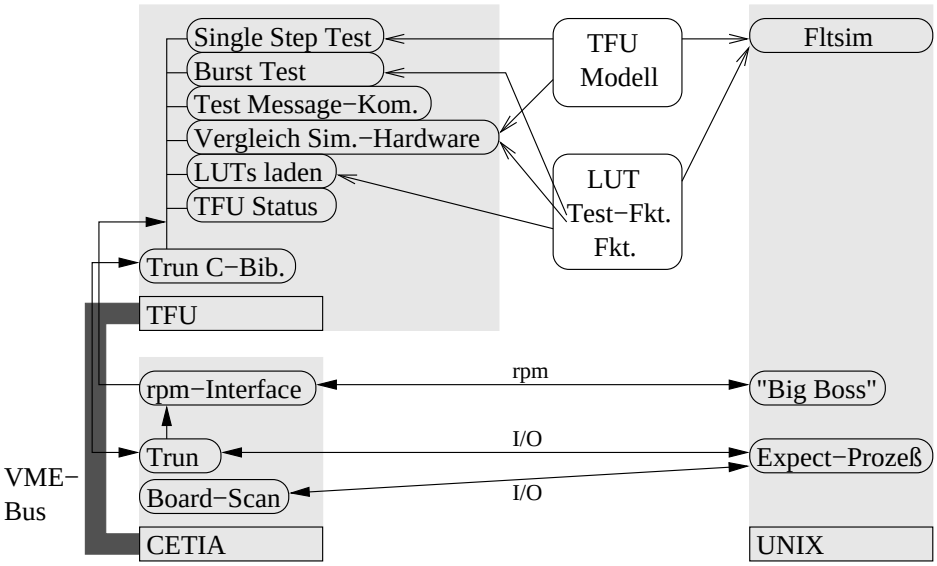


Abb. 3.11. Überblick über die Software und Hardware

Der in der Expect-Scriptsprache geschriebene Prozeß zur Kontrolle mehrerer TRUN-Prozesse (siehe nächsten Abschnitt) wird ebenfalls auf UNIX-Systemen ausgeführt. Beim DESY in Hamburg übernimmt der sog. Big-Boss-Prozeß die Aufgabe zur Kontrolle der FLT-Prozessoren.

Die TRUN-Prozesse werden auf dem VME-Bus-Kontrollrechner, einer Cetia, ausgeführt. TRUN kommuniziert über den VME-Bus mit der Software auf den FLT-Boards, in der Abbildung einer TFU, welche dazu speziell angepaßte C-Bibliotheksroutinen nutzen.

Der Expect-Prozeß kommuniziert über TRUN mit der Software auf dem FLT-Board. Big-Boss dagegen sendet seine Daten über RPM an das RPM-Interface, das sie in den per VME-Bus zugänglichen Speicher des FLT-Board-Mikrocomputers schreibt, wo sie die Board-Software lesen muß. Der umgekehrte Datenaustausch von der Board-Software zum RPM-Interface geschieht über TRUN.

Board-Scan ist ein eigenständiges Cetia-Programm, das den Adreßbereich des VME-Busses nach angeschlossenen FLT-Boards durchsucht und sie anhand ihrer Seriennummer im EEPROM[13] identifiziert.

Auf der TFU ausgeführte Testsoftware wird in den folgenden Kapiteln ausführlich beschrieben. Dort startet man auch die Routinen zum Laden der LUTs und zum Auslesen des aktuellen TFU-Status, der Temperatur, den

[13]EEPROM: Electrical Erasable and Programmable ROM, diese lösch- und wiederbeschreibbaren Speicherbausteine behalten auch nach dem Abschalten des Stroms ihren Inhalt.

angelegten Spannungen, der Message-Rate u.s.w.

3.6.1 Die Inter-Prozeß-Kommunikation mit TRUN

Die TRUN-Betriebssoftware enthielt zur Zeit der Entwicklung des Modells und der Testsoftware noch keine Möglichkeit zur Kommunikation mit UNIX-Prozessen. Eine Möglichkeit zur Synchronisation der einzelnen TRUN-Prozesse durch einen übergeordneten Kontrollprozeß benötigt man jedoch spätestens beim Test der Message-Kommunikation zwischen den TFUs (siehe Kapitel 5.6). Im folgenden wird beschrieben, welche Lösung hierzu am Einsatzort der TFUs beim DESY in Hamburg vorgesehen ist, und warum hier in Mannheim eine alternative Lösung gewählt wurde.

Beim HERA-B-Experiment soll ein einziges, integriertes Nachrichtensystem eingesetzt werden für alle verschieden Arten der Netzwerk-Kommunikation, die notwendig sind. Dazu gehören der Zugriff auf Datenbanken, die Kontrolle des Systems und die Kommunikation mit der Benutzerschnittstelle. Die Nachrichten müssen zwischen den verschiedensten Computerplattformen ausgetauscht werden, darunter UNIX-Systemen oder Plattformen ohne Betriebssystem. Die Länge der einzelnen Nachrichten unterscheidet sich um vier Größenordnungen, die Latenzzeit zeitkritischer Nachrichten um fünf Größenordnungen. Das Kommunikationssystem muß genügend Ressourcen zur Verfügung stellen, damit es auch beim Betrieb des Detektors und der Trigger nicht zusammenbricht. All diese Forderungen soll ein auf Reverse-Path-Multicasting (RPM)[14] aufgebautes System erfüllen [Leg97].

Die RPM-Implementierung des DESY befand sich zum Zeitpunkt, als die Kommunikation hier in Mannheim benötigt wurde, noch in der Entwicklung und war weder uneingeschränkt einsatzfähig noch dokumentiert.

Man benötigte eine alternative Möglichkeit, die Kommunikation eines Kontrollprozesses mit mehreren TRUN-Prozessen zu etablieren, um sofort benötigte Entwicklungen vorantreiben zu können. Diese konnte man zu einem späteren Zeitpunkt auf RPM umstellen.

Wolf hat die Erweiterung von TRUN um eine Netzwerk-Kommunikation über UNIX-Sockets vorgeschlagen [Wol98]. Bevor man jedoch Entwicklungsaufwand in eine Erweiterung von TRUN investiert, die inkompatibel zum RPM-System ist, wäre zu prüfen, ob man nicht auf ein alternatives Softwarepaket zurückgreifen kann, welche die Anforderungen ebenso erfüllt.

Die Script-Sprache Expect erfüllt die Anforderungen. Sie ist ein Werkzeug zur Automation interaktiver Prozesse und eine einfach zu programmierende

[14] Ein nach dem RPM arbeitender Netzwerk-Router reduziert den Datenverkehr, indem er Netzwerkpakete nur noch an ihm bekannte Empfänger weiterleitet [Eff99].

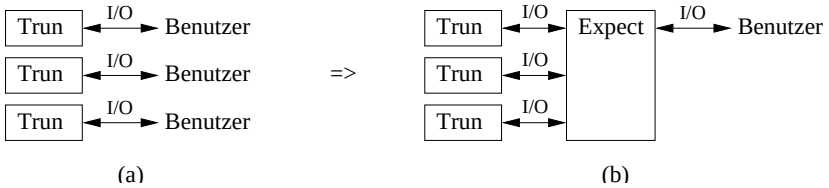


Abb. 3.12. Die Kommunikation eines Expect-Scripts mit mehreren TRUN-Prozessen untereinander und dem Benutzer (b) ersetzt die Kommunikation der einzelnen TRUN-Prozesse mit dem Benutzer (a).

und stabile Lösung zur Etablierung einer Netzwerkkommunikation. Expect ist eine Erweiterung der Scriptsprache Tcl, und läßt sich wie Tcl mit dem Tk-Paket zur Erstellung grafischer Oberflächen ergänzen, genannt Expectk.[15] Mit Expect kann man ein Script erstellen, das andere Prozesse, in diesem Fall TRUN-Prozesse, startet und kontrolliert. Mit ihnen kommuniziert ein Expect-Script über eine bidirektionelle Pipe. Ein Expect-Script übernimmt die Aufgaben eines Benutzers, der über Ein- und Ausgaben mit einem TRUN-Prozeß kommuniziert (siehe Abbildung 3.12) [Lib96][Wel96].

Die Kommunikation über Standard-I/O bietet den Vorteil, daß sich sämtliche beteiligten Prozesse getrennt voneinander entwickeln lassen. Die TRUN-Prozesse kann man unabhängig von dem Expect-Prozeß testen und von Fehlern befreien, indem man als Benutzer die Ein- und Ausgaben simuliert. Auf diese Art und Weise ist es möglich, Fehlerursachen schnell einzukreisen. Expect ist sehr stabil und ausgereift. Es läßt sich schnell erlernen und paßt zu dem Tcl/Tk-Paket, das z.B. auch beim DESY zur Entwicklung der grafischen Benutzeroberfläche eingesetzt wird.

3.7 Zusammenfassung

Die Platine einer TFU ist 36×40 cm^2 groß, besteht aus 14 Leiterbahn-Lagen und ist doppelseitig mit SMD-Bausteinen bestückt. Insgesamt existieren auf einer TFU ca. 20.000 Lötstellen.

Eine TFU ist für die Identifikation und Rekonstruktion einer Teilchenspur in den Detektordaten zuständig. Sie erhält den Auftrag zur Suche einer bestimmten Spur über eine elektronische Nachricht, eine sog. Message. Findet die TFU eine oder mehrere Spuren, so werden je Spur eine Message an die nachfolgenden Trigger-Boards versandt.

Den Algorithmus zur Spurensuche führt der TFU-Pipeline-Prozessor aus, dessen Funktion in Hardware mit kombinatorischer Logik und LUTs realisiert

[15] Beispiele für Expectk-Oberflächen findet man im Anhang A.3.

wurde. Er sucht nach Koinzidenzen in den Detektordaten, die über Glasfaserleitungen empfangen und im Wire-Memory zwischengespeichert werden. Auf die Registerinhalte der Pipelinestufen kann über Scan-Pfade zugegriffen werden.

Die Steuer- und Kontrollogik der TFU ist mit einem integrierten Mikrocomputer ausgestattet, über welchen man auf die Scan-Pfade des Pipeline-Prozessors und Message-Boards zugreifen kann. Die Kommunikation mit dem Mikrocomputer geschieht über eine serielle Terminalschnittstelle oder über den VME-Bus und die TRUN-Laufzeitumgebung von Hageböke und Wolf, welche die Ein- und Ausgaben auf einen UNIX-Prozeß des VME-Host-Rechners exportiert. Zur Kommunikation mit TRUN-Prozessen ist am DESY das RPM-System vorgesehen, in Mannheim wurden im Kontext dieser Arbeit für diese Aufgabe Expect-Scripte verwendet.

Kapitel 4

Simulation der TFU

Die Simulation der TFU ist ein Teil der Simulation des gesamten FLTs und der Pretrigger, daher müssen ihre Ziele und die Anforderungen an sie auch in diesem Kontext untersucht werden. Das Framework FLTSIM wurde von Wolf zur Realisation einer Simulation, die diese Anforderungen erfüllt, entworfen. Bei der Entwicklung des TFU-Modells im Kontext dieser Arbeit haben sich verschiedene Nachteile von FLTSIM herausgestellt. Aus diesem Grund wurde mit dieser Dissertation ein von FLTSIM unabhängiges Modell zur Simulation der TFU entwickelt. Die Definition der Anforderungen an das neue Modell, seine Realisation, seine Simulation und der Vergleich der Simulationsergebnisse mit der Hardware sind die Themen der letzten Abschnitte dieses Kapitels.

4.1 Anforderungen an die Simulation

Wolf hat in [Wol98] die Ziele und Anforderungen einer Simulation des gesamten FLT untersucht. Die Simulation des FLTs baut auf die Simulation seiner einzelnen Komponenten, der TFU, Track-Parameter-Unit, Trigger-Decision-Unit und der Pretrigger, auf. An die Simulation der TFU werden daher Anforderungen gestellt, die sich zum Teil aus den Zielen der FLT-Simulation ergeben.

Nach Wolf hat die FLT-Simulation folgende Ziele:

- *Simulation des FLTs:* Um die Meßergebnisse des Experiments interpretieren zu können, ist die Kenntnis der Effizienz des Detektors und des Triggersystems unbedingt notwendig. Da der Detektor und Trigger nicht mit einem Referenzsignal geeicht werden kann, muß man die Effizienz durch die Simulation bestimmen.
  Mit der Simulation soll auch das Verhalten des Gesamtsystems studiert

werden. Dazu gehört z.B. die Latenzzeit des Triggers, das ist die Zeit, die er benötigt, um eine Entscheidung zu treffen. Ist die Latenzzeit zu hoch, so kann die zugehörige zwischengespeicherte Spur nicht weiter verfolgt werden, da die zugehörigen Detektordaten verloren gehen. Sie werden im Zwischenspeicher mit neuen Detektordaten überschrieben. Das kann z.B. geschehen, falls sich in den Eingangs-FIFOs des Messageboards einer bestimmten TFU zu viele Messages sammeln, so daß sie nicht alle rechtzeitig verarbeitet werden können. Insbesondere muß die Simulation hierzu das asynchrone Verhalten der einzelnen FLT-Boards zueinander möglichst exakt wiedergeben können. Auch das Verhalten bei einem Teilbetrieb des Detektors oder andere Triggerbedingungen möchte man mit der Simulation studieren.
Mit der Simulation können Parameter des Triggers bestimmt werden, noch bevor die Hardware existiert. Dazu gehört beispielsweise Parameter der LUT-Funktionen oder die Verteilung der TFUs auf die Detektorlagen.

- *Inbetriebnahme des Triggers:* Die Simulation wird als Diagnoseinstrument zur Inbetriebnahme des Triggers benötigt, um das Verhalten des Systems unter bestimmten Betriebsbedingungen zu verstehen und Fehler zu entdecken. Änderungen in den Betriebsparametern des Triggers kann man simulieren, um den Betrieb des Triggers zu optimieren.

Hierzu ist noch folgendes zu ergänzen:

- *Überprüfung des Designs der Hardware:* Anhand einer Simulation muß man das Design des Triggers prüfen. Idealerweise sollte dies in einem möglichst frühen Entwicklungsstadium der Hardware geschehen, noch während der Designphase, da entsprechend der 1:10:100:1000-Regel (siehe Kapitel 2.2) die Kosten zur Korrektur von Fehlern dann noch am geringsten sind. Änderungen im Simulations-Modell sind sehr leicht zu realisieren, dagegen erfordern Änderungen von schon bestehender Hardware im ungünstigsten Fall ein komplettes Redesign mit anschließendem Neubau. Insbesondere bei der großen Stückzahl von 90 TFUs sind sie nur mit großem Zeitaufwand auszuführen. Das Simulationsmodell der TFU wurde während der Prototypentwicklung der Hardware fertiggestellt, eine Verhaltenssimulation der TFU konnte ab diesem Zeitpunkt durchgeführt werden.
Man benötigt für die Simulation ein Modell, das die Hardware möglichst exakt nachbildet, mit den selben Beschränkungen in der Genauigkeit, in der Anzahl der Bits der Message-Parameter, der selben LUT-Funktionen, etc. und einem auf den Takt genauen Timing. Die Überein-

stimmung des Modells mit der Hardware wurde durch einen Vergleich der Simulationsergebnisse mit der Hardware geprüft.

- *Test der Hardware:* Man unterscheidet je nach Art der Fehler verschiedene Tests. Im Kapitel 5.4.9 wird beschrieben, daß Teile des TFU-Modells beim Single-Step-Test zum Einsatz kommen.

Um einen doppelten Software-Quellcode und damit Fehler durch Differenzen zu vermeiden und um Entwicklungszeit zu sparen, beabsichtigt man, für die unterschiedlichen Ziele und Aufgaben der Simulation auf ein einziges Modell des Triggers zurückzugreifen.

Die Simulation soll den vorhandenen Fortran-Code ersetzen, mit dem zwar der Algorithmus des FLTs überprüft wurde, jedoch nicht das Zeitverhalten simuliert werden kann, das man benötigt, um die Latenzzeit und damit die Effizienz des Triggers zu bestimmen. Man hat sich gegen eine Erweiterung des Fortran-Codes um Möglichkeiten zur Simulation des Zeitverhaltens entschieden, da dieser im Laufe der Zeit durch verschiedene Modifikationen undurchschaubar geworden ist, was man bei einer Neuentwicklung durch eine übersichtliche Programmstruktur und eine konsequente Dokumentation zu vermeiden beabsichtigt.

4.2 Das FLTSIM-Framework

Die Simulation soll den gesamten FLT und die Pretrigger umfassen. Da mehrere Gruppen aus 4 bis 5 Instituten an der Entwicklung der Triggersysteme beteiligt sind, muß das zur Simulation verwendete Modell so konzipiert sein, daß es von diesen Gruppen entwickelt werden kann. Wolf hat dazu das auf Software-Engineering-Methoden basierende objektorientierte Framework FLTSIM entwickelt [Wol98].

FLTSIM besteht aus einem modularen C++-Code mit einer genauen Abgrenzung der verschiedenen Module und mit einer festen Definition der Schnittstellen. Die Module sind hierarchisch ineinander verschachtelt, von den einzelnen Trigger-Boards, über den Pipeline-Prozessor, bis zu den einzelnen Pipelinestufen. Aufgrund der Modularität können sich mehrere Gruppen gleichzeitig an der Entwicklung und Durchführung einzelner Teile der Simulation beteiligen.

4.2.1 Nachteile von FLTSIM

Ursprünglich war geplant, die TFU-Simulation im Rahmen dieser Dissertation mit den Möglichkeiten des FLTSIM-Frameworks zu entwickeln. Bei der

Entwicklung fand man folgende Nachteile von FLTSIM:

- Das Framework FLTSIM sieht vor, daß jedes FLT-Board im Modell durch eine C++-Klasse abgebildet wird. Darin eingebettet werden die Klassen des Pipelineprozessors und der einzelnen Pipelinestufen. Die Message-Parameter werden für jede Pipelinestufe in einer eigenen Klasse gespeichert, auf die durch Zeiger zugegriffen wird.

- Durch die Verwendung von Zeigern ist der Code bei seiner Ausführung sehr instabil. Um Zugriffe auf undefinierte Zeiger, welche die Simulation zum Absturz bringen, zu vermeiden, ist der Code zur Vermeidung dieser Zugriffe in vielen Pipelinestufen aufwendiger als als die eigentliche TFU-Logik.

- Durch die differenzierte Unterteilung der TFU-Logik in die Klassen für den Pipeline-Prozessor, die einzelnen Pipelinestufen und die Datenklassen wurde der Code unflexibel und unübersichtlich. Modifikationen waren nur mit großem Aufwand durchzuführen. Die Logik-Einheiten wie die Addierer und die Koinzidenz-Matrix erstrecken sich in der Hardware dagegen teilweise über mehrere Pipelinestufen. Diese mußte man unterteilen oder in einer einzigen Stufe konzentrieren. Dies erschwerte besonders die Realisation der Koinzidenz-Matrix mit ihrer Stalling-Funktion.

- Eine FLTSIM-Simulation führt man auf einem UNIX-Rechner aus. Die Hardware kontrolliert man mit einem TRUN-Programm. Da TRUN außer der Standard- Ein- und Ausgabe und der Schnittstelle zum Dateisystem keine Möglichkeit zur Kommunikation mit dem Simulationsprozeß besitzt, hat man beim Vergleich der Simulation mit der Hardware die Message-Parameter über das Dateisystem ausgetauscht. Dies setzte zusätzlichen Programmcode zur Erzeugung und Auswertung dieser Dateien voraus.

Mit FLTSIM wurden 4 TFUs in Serie aneinandergereiht und die Verarbeitung zweier Messages simuliert. Weitere Simulationen hat man nicht durchgeführt, da sich aufgrund der genannten Nachteile notwendige Korrekturen als kompliziert und zeitaufwendig erwiesen haben. Der Vergleich mit der Hardware über das Dateisystem hat sich als impraktikabel erwiesen, insbesondere bezüglich der Analyse der Fehler bei der Implementation der Stalling-Funktion.

4.3 Neuentwicklung der TFU-Simulation

Aufgrund der im letzten Abschnitte genannten Nachteile von FLTSIM wurde im Rahmen dieser Arbeit entschieden, das Modell zur Simulation des TFU-Pipeline-Prozessors neu zu entwickeln.

4.3.1 Anforderungen

Folgende Anforderungen werden an eine Neuentwicklung gestellt:

- *Verhaltensmodell:* Wie schon oben beschrieben wurde, muß das Modell im Timing auf den Taktzyklus genau mit der Hardware übereinstimmen und die Hardware mit allen ihren Beschränkungen z.B. in der Bit-Breite der Message-Parameter exakt nachbilden. Es handelt sich hier um ein Verhaltensmodell, da nicht nur die Funktion, sondern auch das Timing nachgebildet wird.
  Damit man die Übereinstimmung mit der Hardware durch Vergleich der Message-Parameter überprüfen kann, muß das Modell zumindest an den Pipelinestufen, auf die über Scan-Pfade zugegriffen werden kann, die Messages exakt berechnen.

- *Zugriff auf die Register der Hardware:* Um die Überprüfung des Modells durch den Vergleich mit der Hardware zu erleichtern, und um auch beim Test der Hardware auf bestimmte Teile des Modells zugreifen zu können, soll die Software auf dem TFU-Mikrocomputer lauffähig sein. In diesem Fall ist keine spezielle Kommunikation zwischen zwei Programmen auf zwei Rechnern notwendig, sondern man kann über die Scan-Pfade direkt auf die Hardware zugreifen. Da bei TRUN-Programmen ein C-Cross-Compiler zum Einsatz kommt, beschränkt sich der Code auf die Programmiersprache C.

- *Integration in ein übergeordnetes Framework:* Das Modell soll in ein übergeordnetes Framework integrierbar sein, um seine Ziele, die Simulation und die Unterstützung bei der Inbetriebnahme des gesamten FLTs, weiterhin zu ermöglichen. Das neue Modell ist als C-Funktion ausgelegt, die ohne Probleme in ein C++-Framework eingefügt werden kann.

- *Struktur der Daten:* Der Zugriff auf die Message-Parameter soll gegenüber FLTSIM vereinfacht werden. Dort ist der Zugriff durch die Abkapselung der Daten in Klassen, auf die man nur über Zeiger zugreifen kann, sehr komplex. Man benötigt den Zugriff beim Vergleich mit der

Hardware, beim Verfolgen einer Message in der Pipeline, zur Ausgabe von Zwischenergebnissen, und zum Speichern der Zwischenergebnisse beim Erstellen einer Historie.
Die Daten werden dazu in einer einzigen, globalen Struktur untergebracht. Bei der Simulation auf dem TFU-Mikrocomputer ist die Globalität kein Problem, da ja auch nur auf eine einzige Hardware zugegriffen wird. Die Simulation mehrerer TFUs soll weiterhin mit einem übergeordneten Framework durchgeführt werden, dabei kann man die Daten-Struktur in der TFU-Board-Klasse kapseln, wobei ihre Globalität verloren geht.

- *Stabilität:* Das Programm soll wesentlich stabiler laufen. Daher vermeidet man nach Möglichkeit Zeiger.

- *Komplexität:* Der Programm-Code soll von wesentlich geringerer Komplexität sein. Indem man die Unterteilung der einzelnen Pipelinestufen in eigene Klassen vermeidet, müssen die Funktionen, welche sich über mehrere Pipelinestufen erstrecken, nicht mehr unterteilt werden. Korrekturen und Änderungen in den Funktionen lassen sich damit wesentlich einfacher durchführen.
  Zur Reduktion der Komplexität tragen auch die Vermeidung von Zeigern und die Möglichkeit zum direkten Zugriff auf die Register der Hardware bei. Durch die geringere Komplexität des Programmcodes war es einfacher, Fehler zu finden und zu korrigieren, hierdurch konnte man im Vergleich zur Entwicklung des TFU-Modells mit FLTSIM sehr viel Entwicklungszeit sparen.

- *Dokumentation:* Weiterhin soll, wie auch schon bei FLTSIM, die automatische Erstellung einer Dokumentation mit DOC++ unterstützt werden. DOC++ durchsucht den Quellcode nach Funktionen, globalen Variablen und speziell markierten Kommentarzeilen erzeugt daraus HTML- oder TEX-Dokumente [Aco99].

4.3.2 Realisation der TFU-Simulation

Die Anforderungen erfüllt das in der Abbildung 4.1 skizzierte Programm. Die globale Struktur *PipelineData* enthält sämtliche Message-Parameter aller Pipelinestufen und alle Flags (Schalter) für die verschiedenen Modi des Pipeline-Prozessors. In den *operatePipeN()*-Funktionen werden aus den alten die neuen Message-Parameter berechnet. Der einmalige Aufruf der Funktion *tfuOperatePipeline()* entspricht einem Aktivieren der Pipeline für einen Takt

und bearbeitet sämtliche Pipelinestufen durch Aufruf der Funktionen *operatePipeN()*. Dies geschieht in umgekehrter Reihenfolge von der letzten bis zur ersten Pipelinestufe, so daß die einzelnen *operatePipeN()*-Funktionsaufrufe immer auf die Daten der vorigen Stufen zugreifen, bevor diese von den neu berechneten Message-Parametern überschrieben werden.

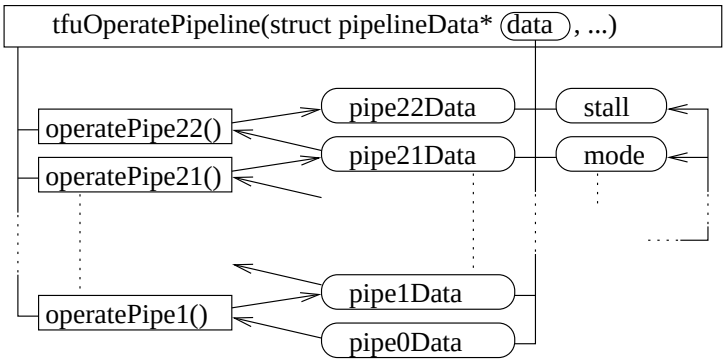


Abb. 4.1. Durch den Aufruf der Funktion *tfuOperatePipeline(...)* wird die Pipeline für einen Takt simuliert. Die Eingangsmessage- (*pipe0Data*) und Betriebsparameter übergibt man der Funktion über die Datenstruktur *struct pipelineData*, sie enthält nach dem Aufruf die Ergebnisse der Simulation in *pipe22Data*.

Das Modell ermöglicht eine zeitdiskrete, deterministische Simulation. Die diskreten Zeitschritte durch das wiederholte Aufrufen der *tfuOperatePipeline()*-Funktion entsprechen in der Hardware den Arbeitstakten. Deterministisch ist es nach [Pag91], da seine Reaktion auf eine bestimmte Eingabe, ausgehend von einem bestimmten Zustand, eindeutig festgelegt ist.[1]

Da nicht nur die logische Funktion, sondern auch das zeitliche Verhalten der Hardware in dem Modell nachgebildet wird, handelt es sich beim Modell des Pipeline-Prozessors um ein Verhaltensmodell.

Simulation der in Hardware integrierten kombinatorischen Logik

Die mit kombinatorischer Logik realisierten Funktionen hat man im Modell durch C-Funktionen, die mit einem Aufruf der *operatePipeN()*-Funktionen ausgeführt werden, nachgebildet. Dazu zählen Funktionen wie die logischen Operationen, die Addition oder die Identitätsabbildung von Message-Parametern von einer Pipelinestufe zur nächsten. Um sicherzustellen, daß die Argumente und Funktionswerte nicht den durch die Hardware beschränkten Wertebereich überschreiten, und dadurch falsche Ergebnisse liefern, kürzt man diese mittels einer Maske auf die gleiche Bit-Breite wie in der Hardware.

[1] Das trifft, wenn sie fehlerfrei arbeitet, auch für die Hardware zu.

Simulation der LUTs

Die LUT-Funktionen lassen sich in das Modell übernehmen. Man verwendet zum Laden der LUTs genau den selben Quellcode mit den C-Funktionen der LUTs wie im Modell. Dadurch muß eine Funktion bei Bedarf nur an einer einzigen Stelle im Code geändert werden, doppelter Code wird vermieden, die exakte Übereinstimmung der Hard- und Software-LUTs bleibt gewährleistet. Zu beachten ist auch hier der beschränkte Wertebereich der Argumente und Funktionswerte in der Hardware.

Simulation der Koinzidenz-Matrix

Die Logik der Koinzidenz-Matrix gliedert sich, wie schon in Kapitel 3.3.2 beschrieben wurde, in zwei Teile. Im ersten Teil wird die Suche nach Koinzidenzen in den Detektor-Daten aus dem Wire-Memory durchgeführt. Die Logik der PLD-Bausteine hat man in der Programmiersprache ABEL erstellt [Dat90]. Da im ersten Teil größtenteils nur logische Verknüpfungen verwendet werden, die in der Syntax von ABEL und C fast gleich sind, konnte sie nahezu unverändert in das C-Modell übernommen werden.

Der zweite Teil der Logik setzt die gefundenen Koinzidenzen in Message-Parameter um und erzeugt bei Bedarf neue Messages, wobei der vorangehende Teil der Pipeline über das Stall-Signal angehalten wird, bzw. er löscht Messages. Die Logik wurde im Modell mit einer Funktion nachgebildet, die zur Realisation des Stall-Vorgangs von den Funktionswerten ihres letzten Aufrufs abhängt, also genauso wie die Hardware ein sequentielles Verhalten erzeugt.

4.4 Vergleich der Simulation mit der Hardware

Im Kontext der vorliegenden Dissertation wurde geprüft, ob das Modell mit der Hardware übereinstimmt. Die Überprüfung wurde durchgeführt, indem man die Hardware und die Simulation mit den selben Daten versorgte, mit den selben Messages, Detektordaten, Betriebsmodi und LUT-Funktionen. Die Messages verglich man in jedem Takt in allen Pipelinestufen, auf die in der Hardware über Scan-Pfade zugegriffen werden konnte.

Die gefundenen Differenzen, hatten ihre Ursache in Fehlern des Modells, der Hardware und der Software, mit welcher der Vergleich durchgeführt wurde.

4.4.1 Abschätzung der Zeitdauer

An dieser Stelle soll abgeschätzt werden, wie lange es dauert, beim Vergleich alle möglichen Messages auf alle möglichen Zustände des Wire-Memorys, der LUTs und der verschiedenen Modi der Addierer und der Koinzidenz-Matrix anzuwenden, das System also mit exhaustiven Testmustern zu prüfen. Wir wollen uns bei der Abschätzung auf alle möglichen Messages beschränken. Eine Message ist 80 Bit groß, damit ergibt sich bei der Message-Verarbeitungrate der TFU von 50 MHz eine Testdauer von $2^{80}/50\text{MHz} \approx$ 800 Millionen Jahren. Die Software ist dabei noch um mehrere Größenordnungen langsamer als die Hardware. Man kann folglich keine exhaustive Testmuster verwenden und muß geeignete Methoden finden, dennoch einen möglichst effektiven Vergleich durchzuführen.

4.4.2 Der Algorithmus

Der Algorithmus in Abbildung 4.2 schildert das angewandte Vorgehen eines Vergleichs. Zuerst werden die LUTs der Hardware geladen. Man verwendet spezielle LUT-Funktionen, die alle Eingangsparameter auf alle Ausgangsparameter abbilden und den Zufallscharakter der Eingangswerte möglichst bewahren. Dann setzt man die Modi des Addierers und der Koinzidenz-Matrix und lädt das Wire-Memory mit Zufallszahlen.

Man erzeugt eine Message, deren Parameter bis auf bestimmte Parameter, die z.B. die Gültigkeit der Message anzeigen, aus Pseudo-Zufallszahlen bestehen. Nun aktiviert man den Pipeline-Prozessor für einen Takt und ruft die *tfuOperatePipeline()*-Funktion einmal auf. Anschließend vergleicht man die Message-Parameter aller Pipelinestufen, auf die über die Scan-Pfade zugegriffen werden kann. Dieser Vorgang wird mehrere 10.000 mal wiederholt, oder abgebrochen, falls ein Fehler gefunden wird. Nun ändert man die Wire-Daten und die Modi, und startet den Vorgang erneut. Insgesamt wird das mindestens solange wiederholt, bis man die Pipeline in allen Kombinationen ihrer verschiedenen Betriebsmodi verglichen hat, ohne Fehler zu finden.

Die Pseudo-Zufallszahlen erzeugt man mit der *rand()*-Funktion aus der C-Standardbibliothek. Gibt man der Pseudo-Random-Funktion einen Initialisierungswert vor, so läßt sich die gleiche Folge von Zufallszahlen erzeugen. Diese Eigenschaft ist wichtig, wenn man Fehler rekonstruieren möchte.

4.4.3 Ergebnisse des Vergleichs

Durch den Vergleich konnte man verschiedene Fehler im Modell korrigieren, so z.B. bei der Stall-Funktion. Es wurde ein Fehler in der durch Hardware

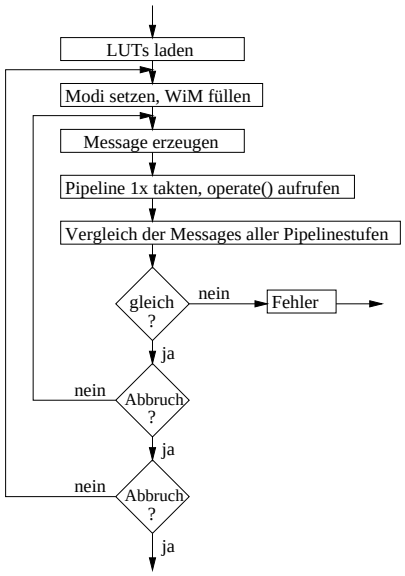


Abb. 4.2. Der Algorithmus zur gegenseitigen Überprüfung der Simulation mit der Hardware

realisierten kombinatorischen Logik gefunden, den Gläß durch eine Neuprogrammierung der PLDs beheben konnte.

Der Vergleich wurde mit mehreren 100.000 Messages durchgeführt, ohne weitere Fehler zu finden. Dies ist kein Beweis für die exakte Übereinstimmung, den man auf diese Art und Weise auch nicht erbringen kann, aber man kann mit großer Sicherheit davon ausgehen, daß das Modell und die Hardware übereinstimmen.

4.5 Zusammenfassung

An eine Simulation der TFU werden verschiedene Anforderungen gestellt: Sie soll als Teil der Simulation des gesamten FLTs nicht nur dessen Funktion, sondern auch sein Zeitverhalten berechnen können. Dies ist eine Voraussetzung für die Bestimmung der Latenzzeit und damit der Effizienz des Triggers. Des weiteren soll sie den Einfluß der Betriebsparameter auf das Triggersystem voraussagen können und als Diagnoseinstrument die Inbetriebnahme des Triggers unterstützen. Ferner soll sie zur Verifikation des Hardware-Designs in einem möglichst frühen Entwicklungsstadium sowie zum Test der Hardware eingesetzt werden. Um einen doppelten Software-Code und damit Fehler durch Differenzen zu vermeiden und um Entwicklungszeit zu sparen, besteht die Absicht, für die unterschiedlichen Ziele und Aufgaben der Simu-

lation auf ein einziges Modell des Triggers zurückzugreifen. Der vorhandene Fortran-Code, mit dem die physikalische Funktion des FLTs simuliert wurde, soll ersetzt werden, da er den Anforderungen nicht mehr genügt.

Zeitgleich mit der Entwicklung eines Hardware-Prototyps der TFU hat Wolf das objektorientierte Framework FLTSIM zur Simulation des FLTs und seiner Komponenten, incl. der TFU, entwickelt. Bei der Implementation des TFU-Pipeline-Prozessors im Kontext dieser Arbeit haben sich Nachteile von FLTSIM herausgestellt. Daher wurde zur Simulation des Pipeline-Prozessors in dieser Dissertation ein neues Modell entworfen, dessen Programm-Code im Vergleich zu FLTSIM von wesentlich geringerer Komplexität ist. Dies vereinfacht die Entwicklung, Pflege und Modifikation des Codes. Der Code ist in der Ausführung sehr stabil. Er läßt sich zur Simulation des gesamten Triggers in ein übergeordnetes Framework integrieren.

Die Simulationsergebnisse können direkt, ohne Umweg über eine Netzwerkverbindung oder das Dateisystem, mit der Hardware verglichen werden. Im Kontext dieser Dissertation hat man bei einem Vergleich der TFU-Simulation mit der Hardware sowohl in der Simulations-Software als auch im Design der Hardware Fehler gefunden. Diese wurden korrigiert. Gläß konnte den entdeckten Hardware-Fehler durch Umprogrammieren der PLD-Bausteine berichtigen.

4.6 Status und Ausblick

Das TFU-Simulationsmodell wurde vom Autor dieser Arbeit zuerst, wie es von Wolf geplant war, innerhalb des FLTSIM-Frameworks entwickelt. Dieses Modell wurde jedoch vom Autor verworfen, da sich seine Überprüfung durch einen Vergleich mit der TFU-Hardware aus den in diesem Kapitel genannten Gründen als problematisch erwiesen hat. Insbesondere Fehler in der Stall-Funktion ließen sich im Modell nur ungenügend diagnostizieren.

Im Rahmen dieser Arbeit wurden die Anforderungen an ein verbessertes Simulationsmodell neu definiert und dieses entsprechend realisiert. Man hat das Modell anschließend durch einen direkten Vergleich mit der Hardware überprüft. Fehler im Design der Hardware und im Modell konnten dadurch gefunden und korrigiert werden. Diese Tatsache unterstreicht die Bedeutung einer solchen Überprüfung. Ohne sie wären die Fehler wahrscheinlich erst zu einem späteren Zeitpunkt während oder nach der Produktion entdeckt worden, wodurch die Kosten und der Zeitaufwand zur Korrektur erheblich höher gewesen wären.

Die Simulation der TFU anhand dieses Modells hat sich in der Ausführung als sehr stabil erwiesen. FLTSIM wird aufgrund der in diesem Kapitel

genannten Kritikpunkte am DESY von Nörenberg überarbeitet [Nör99].

Bei Modifikationen des Quelltextes stellte es sich der Nutzen heraus, mit dem Werkzeug DOC++ jederzeit eine Dokumentation der Software erstellen zu können.

Da sich das in dieser Dissertation projektierte und eingesetzte Softwarekonzept zur Realisation der TFU-Simulation als geeignet und zuverlässig erwiesen hat, wurde es von Harter zur Entwicklung der Simulation der Track-Parameter-Unit übernommen.

Am DESY hat man die Modelle der TFU und der Track-Parameter-Unit zur Simulation des kompletten FLTs eingesetzt. Erste Ergebnisse hat Flammer in [Fla99] vorgestellt.

Kapitel 5

Test der TFU

In diesem Kapitel werden zuerst die Anforderungen an einen Test der TFU definiert. Danach werden die Pläne für die Tests nach der Produktion und dem Betrieb der TFU im FLT vorgestellt.

Die Themen der darauf folgenden Abschnitte sind die Beschreibungen und Diskussionen zweier Prozeduren zum Test des TFU-Pipeline-Prozessors. Es handelt sich dabei um den Single-Step-Test, bei welchem die TFU-Pipeline zur Diagnose statischer Fehler in Testklassen unterteilt wird, und welcher einen Schwerpunkt dieser Dissertation darstellt, und um den Burst-Test von Gläß, der die Suche nach dynamischen Fehlern unterstützt.

5.1 Anforderungen an die Tests

In den verschieden Phasen des Hardware-Lebenszyklus werden verschiedene Anforderungen an die Tests gestellt.

In der Entwicklungsphase benötigt man einen Test, der die Funktion der TFU anhand ihrer Spezifikation überprüft. Fehler durch ein fehlerhaftes Design müssen so früh wie möglich gefunden werden, da ihre Korrekturen in späteren Entwicklungs-, Produktions- und Betriebsphasen, insbesondere auch im Hinblick auf den Zeitaufwand, nach der 1:10:100:1000-Regel (siehe Kapitel 2.2) immer höhere Kosten verursachen.

Zum anderen benötigt man Software, die den Entwickler bei der Überprüfung des Signal-Timings unterstützt. Das Timing eines Signals ist korrekt, wenn es in dem Zeitfenster, in welchem es einen der beiden logischen Werte angenommen haben muß, stabil ist. Daß die Suche nach den aus ungünstigem Timing resultierenden dynamischen Fehlern ein kritischer Faktor in der Entwicklung und Produktion ist, wurde in Kapitel 2.2.4 geschildert.

Von der TFU wurden in einer Kleinserie 90 Boards produziert. Die Her-

stellung und Bestückung der Platine wurde von einer externen Firma durchgeführt und die Platinen mit einer Rate von drei TFUs pro Woche geliefert. Um die Qualität der Platinen zu garantieren, wurden diese Tests unterzogen.

Damit die geplante Produktionszeit von drei TFUs pro Woche eingehalten werden konnte, waren Tests mit effektiver Fehlerdiagnose zur Qualitätskontrolle notwendig. Die Tests mußten die Fehler so gut wie möglich lokalisieren und den defekten Komponenten zuordnen können, damit sie schnell repariert werden konnten. Da die zweiseitige Bestückung der Platinen in SMD-Technik neben den hohen Anschaffungskosten den Einsatz eines In-Circuit-Testers (siehe Kapitel 2.2.2) erschwerte, benötigt man alternative Testmethoden.

Auch beim Einsatz der TFU im FLT müssen diese immer wieder getestet werden. Ein Fehler, der nicht zu einem Totalausfall der TFU führt, kann unter Umständen nicht sofort entdeckt werden. Wenn eine solche defekte TFU dazu führt, daß bestimmte Spuren im Detektor nicht weiter verfolgt werden, der FLT also relevante Spuren verwirft, gehen diese für die Auswertung des Experiments verloren.

Die Tests der TFU im FLT sollten unabhängig von externen Testgeräten sein, wie z.B. einem Terminal an der seriellen Schnittstelle, um sie vom Kontrollzentrum des FLT aus durchführen zu können. Eine Fehlerlokalisation ist hier nicht unbedingt notwendig, da eine TFU im FLT nicht dort repariert, sondern nur ausgetauscht wird.

5.2 Tests bei der Entwicklung und Produktion

Den hier beschriebenen Testplan haben Gläß und Wurz aufgrund der Erfahrungen mit den Tests und den aufgetretenen Fehlern im Laufe der Entwicklung der TFU-Prototypen und der Produktion immer weiter verbessert. In folgender Reihenfolge wurden die TFUs bei der Produktion den einzelnen Tests unterzogen. Die Reihenfolge ist wichtig, da ein späterer Test in der Regel die korrekte Funktion der zuvor getesteten Komponenten voraussetzt.[1]

1. *Produktion der Platine:* Die Produktion der Platine wurde von einer externen Firma ausgeführt. Nach der Produktion wurden die Platinen anhand ihrer Spezifikationsdaten (Gerberdaten) mit einem Flying-Probers-Gerät elektrisch überprüft

2. *Optische Kontrolle:* Nach der Bestückung, welche von der gleichen Firma ausgeführt wurde, hat man die Platinen einer Sichtkontrolle un-

[1] Fotos der TFU und der Tests findet man im Internet unter
http://www-li5.ti.uni-mannheim.de/mass_par/herab.shtml

terzogen. Man suchte dabei vor allem nach defekten Lötstellen und Bestückungsfehlern.

3. *Überprüfen von Kurzschlüssen:* Vor dem Anlegen der Versorgungsspannung prüfte man die TFU auf Kurzschlüsse. Die TFU benötigt einen Strom von ca. 40 A, das Netzteil kann bis zu 500 A liefern. Bei einem Kurzschluß kann ein Strom dieser Größenordnung die Hardware sehr schnell zerstören. Aus diesem Grund hat man die TFU auch mit Schmelzsicherungen geschützt.

4. *Test des Mikrocomputers:* Man nahm den Mikrocomputer über seine serielle Schnittstelle mit einem Assemblerprogramm in Betrieb und testete sein RAM durch Schreiben und Lesen von Pseudo-Zufallszahlen. Anschließend testete man seine VME-Bus-Schnittstelle. (Dauer ca. 10 min)
   Startete der Mikrocomputer nicht, so mußte man den Fehler mit konventionellen Methoden suchen. Hierbei war es hilfreich, daß man über den VME-Bus auf den gesamten Adreßbereich des Mikrocomputers zugreifen und so die Fehlerursachen einschränken kann.

5. *Test der Scan-Pfade:* Die Scan-Pfade testete man durch Schreiben und Lesen von Testmustern. (Dauer ca. 10 min)

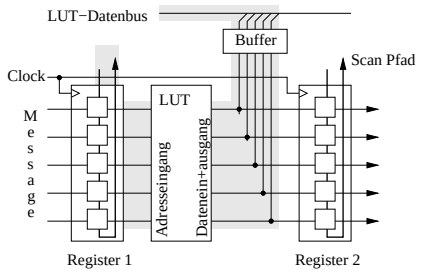


Abb. 5.1. Eine Pipelinestufe mit einer LUT. Grau unterlegt sind die Adreß- und Datenpfade, über welchen die LUT vom Mikrocomputer aus angesprochen wird.

6. *Test der LUTs:* Die RAM-Bausteine der LUTs wurden durch Schreiben und Lesen von Pseudo-Zufallszahlen getestet. Der Test lädt zuerst alle 37 LUTs und ließt sie anschließend aus, um sicherzustellen, daß sie auch korrekt selektiert werden. Dieser Vorgang dauert ca. 8 s und wurde ein- bis zweihundert mal wiederholt. Der Test findet Adressierungsfehler sowie Fehler im Datenpfad zum Mikrocomputer, nicht jedoch im Datenpfad zum Pipeline-Prozessor (vergl. Abbildung 5.1). (Dauer ca. 20 min).

7. *Test des Wire-Memorys:* Das RAM des Wire-Memorys wurde ebenfalls durch Schreiben und Lesen von Pseudo-Zufallszahlen getestet. (Dauer ca. 10 min)

8. *Burst-Test (Assembler-Version):* Mit diesem Test prüfte man die komplette Pipeline der TFU von den Registern der Empfänger auf dem Message-Board bis zum Sender-Register im Dauerbetrieb (siehe Abbildung 5.2). Der Dauerbetrieb hat den Vorteil, daß im Gegensatz zum Einzelschrittbetrieb auch dynamische Fehler gefunden werden können. Im Dauerbetrieb ist der Pipeline-Prozessor mehrere aufeinanderfolgende Takte aktiv. Beim Einzelschrittbetrieb wird er dagegen nur für einzelne Taktschritte aktiviert, dazwischen hält er für mehrere Takte an. In dieser inaktiven Zeit können sich die Signalpegel, die im Dauerbetrieb zu dynamischen Fehlern führen können, stabilisieren.
   Die Assembler-Version des Tests füllt zuerst die FIFOs der Message-Empfänger mit Pseudo-Zufallszahlen und aktiviert dann den Pipeline-Prozessor. Dieser verarbeitet die Messages im Dauerbetrieb und schreibt die Ergebnisse in das Sender-FIFO. Er testet also auch die Schnittstelle zwischen dem Message-Board und der TFU. Die Ergebnisse werden anschließend mit einem gespeicherten Datensatz der gleichen Testmuster aus einem Einzelschrittbetrieb verglichen. (Dauer ca. 1 Stunde)

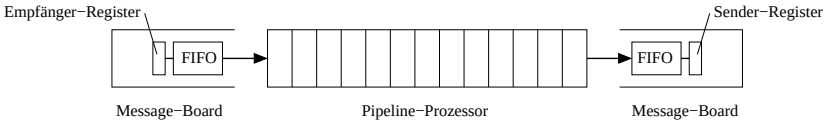


Abb. 5.2. Beim Burst-Test werden die Messages aus einem oder mehreren Empfänger-FIFOs vom Pipeline-Prozessor im Dauerbetrieb verarbeitet und in das Sender-FIFO geschrieben.

9. *Test der Empfänger für die Detektordaten:* Zum Test der 24 Empfänger für die Detektordaten verband man diese über optische Glasfaserleitungen mit den Sendern eines speziellen Test-Boards. Das Test-Board besitzt neben den 24 Sendern einen Mikrocomputer, der wie bei der TFU über den VME-Bus oder eine serielle Schnittstelle angesprochen wird. Mit ihm generierte man die Testmuster. Pro Leitung sandte der Test ca. 10^9 Datenpakete. Fand er keine Fehler, so war die Bitfehlerrate der getesteten Leitungen damit kleiner als 10^{-10}. (Dauer ca. $1\frac{1}{2}$Tage)

10. *Start von TRUN:* Die zuvor eingesetzten Tests bestehen aus Assembler-Routinen. Die folgende Testsoftware führte man unter der TRUN-Laufzeitumgebung aus. Startet TRUN nicht korrekt, so kann das, wenn

man die Fehlerquellen außerhalb des TFU-Boards ausschließt, z.B. an Problemen mit der VME-Bus-Schnittstelle liegen.

11. *Single-Step-Test:* Der Test unterteilt den Pipeline-Prozessor incl. des Message-Boards in einzelne Testklassen und überprüft diese im Einzelschrittbetrieb. Er findet schnell statische Fehler wie Stuck-at-Fehler, durch welche man einen großen Teil der Produktionsfehler klassifizieren kann, und lokalisiert sie auf die Testklasse, in der Regel eine Pipelinestufe, genau. (Eine detaillierte Beschreibung folgt) (Dauer ca. 10 min)

12. *Burst-Test (TRUN-Version):* Diese Version des Burst-Tests startet man unter TRUN. Eine detaillierte Beschreibung des Tests folgt. Man führt ihn als Dauertest über den Zeitraum von 2 Tagen aus, um auch selten auftretende temporäre Fehler mit einer höheren Wahrscheinlichkeit zu finden. (Dauer ca. 2 Tage)

13. *Test der Message-Kommunikation:* Dieser Test ist in den oben genannten Burst-Test integriert. Man schließt hierzu die Empfänger des Message-Boards an den Sender des selben Boards an und sendet über diese Verbindungen Testdaten.

14. *Test der Empfänger für die Detektordaten (Wiederholung):* Der Test der Empfänger für die Detektordaten wurde aufgrund der beobachteten Alterungserscheinungen der Empfänger-ICs unmittelbar vor der Auslieferung einer TFU an das DESY wiederholt. Diese wiesen zum Teil nach einer Betriebsdauer ein bis zwei Wochen Defekte auf. (Dauer ca. 2 Stunden)

5.3 Test der TFU beim Einsatz im FLT

Um die Qualität und korrekte Funktion der TFUs auch bei ihrem Einsatz im FLT zu gewähren, muß man diese wiederholt testen.

Man unterscheidet Tests, die nach dem Einschalten der Stromversorgung unmittelbar vor, während und nach dem FLT-Betrieb der TFU durchgeführt werden. Viele der bei der Produktion eingesetzten Tests können übernommen werden, die Assembler-Routinen kann man in ein C-Programm integrieren und so über die FLT-Kontrollsoftware, die auf das TRUN-System aufbaut, ausführen.

Tests vor dem FLT-Betrieb

1. *Test des Mikrocomputers:* Um eine korrekte Funktion der Software zu gewährleisten, sollte man das RAM des Mikrocomputers und die VME-Schnittstelle durch Schreiben und Lesen von Testdaten überprüfen.

2. *Single-Step-Test:* Dieser Test hat einen Test der Scan-Pfade implementiert, deren korrekte Funktion die folgenden Tests voraussetzen, deshalb sollte er an dieser Stelle in der Test-Reihenfolge stehen. (Eine detaillierte Beschreibung folgt)

3. *Test der LUTs:* Hier sollte man den LUT-RAM-Test aus der Produktion einsetzen. Diesen Test kann man, um Zeit zu sparen, auf wenige Durchgänge verkürzen. Bei einem Durchgang werden alle LUTs mit Zufallszahlen beschrieben und anschließend deren Inhalt überprüft. Da nach wenigen Durchgängen nicht unbedingt sichergestellt ist, daß alle Bits mit beiden Logikwerten getestet wurden, sollte man den Inhalt der LUT nach dem Laden mit ihren FLT-Funktionen überprüfen.

4. *Test des Wire-Memorys:* Das Wire-Memory testet man wie bei der Produktion durch Schreiben und Lesen von Pseudo-Zufallszahlen.

5. *Test der Message-Kommunikation:* Dieser Test überprüft zuerst, ob alle FLT-Boards richtig verbunden sind und miteinander kommunizieren können, und dann die Fehleranfälligkeit der Kommunikation. (Eine detaillierte Beschreibung folgt)

6. *Burst-Test:* Hier kann man den Burst-Test aus der Produktion einsetzen.

7. *Test zum Empfang der Detektordaten:* Die Elektronik am Detektor mit den Sendern für die Detektordaten besitzt eine Möglichkeit zum Senden von einfachen Testdaten, mit diesen können die Glasfaserverbindungen getestet werden.

Tests während des FLT-Betriebs

- *Überwachung der Temperatur und Spannung:* Während des FLT-Betriebs müssen die Temperatur und die Spannung überwacht werden. Die FLT-Boards sind dazu mit den entsprechenden Sensoren ausgestattet, welche mit dem Mikrocomputer ausgelesen werden können. Bei einem Überschreiten der Grenzwerte muß die FLT-Betriebs-Software reagieren und im Notfall das Netzteil des Crates mit dem überhitzten Board

abschalten. Längerer Betrieb bei zu hoher Temperatur führt zur Zerstörung einzelner Komponenten.

- *Überwachung der Message-Rate:* Weichen die Message-Raten eines oder mehrerer FLT-Boards von ihren erwarteten Werten ab, so kann ein Fehler vorliegen. In diesem Kontext sind viele Fehlerursachen denkbar, z.B. Fehler im Detektor, Hardware-Defekte, Defekte in der Message-Kommunikation, oder Fehler in der Software, z.B. durch falsch gesetzte Betriebsparameter oder mit falschen Funktionen geladene LUTs.

- *Selbsttest des Pipeline-Prozessors:* Es ist möglich, in der ersten Pipelinestufe eine aus Pseudo-Zufallszahlen bestehende Message zu generieren und deren Signatur in der letzten Pipelinestufe zu analysieren.

Tests nach dem FLT-Betrieb

Um Fehler zu finden, die während des Betriebs der TFU auftreten, wie z.B. bei der Erwärmung der Hardware nach dem Einschalten, sollten die Tests, welche vor dem FLT-Betrieb ausgeführt werden, nach bestimmten Zeitintervallen wiederholt werden, wenn es der Detektorbetrieb zuläßt, spätestens jedoch nach dem FLT-Betrieb.

Findet man Fehler, so weiß man, daß der Trigger seit dem letzten Test falsche Entscheidungen getroffen haben kann. Findet er zu viele Spuren, so kann der Second-Level-Trigger diese ausfiltern, falls dadurch seine Eingangsdatenrate nicht überschritten wird. Findet er zu wenig Spuren, so wird die Effizienz des Experiments herabgesetzt.

5.4 Der Single-Step-Test

Die Anforderungen nach einer schnellen und detaillierten Fehlerdiagnose von statischen Fehlern, deren Ursachen z.B. an Lötfehlern oder defekten Bauteilen und Leiterbahnen liegen können, sind mit dem Single-Step-Test am besten zu erfüllen. Er prüft den Pipeline-Prozessor, der einen großen Teil der TFU-Ressourcen einnimmt. Er wird Single-Step-Test (*deutsch* Einzel-Schritt-Test) genannt, da man den Pipeline-Prozessor nur für einzelne Takte aktiviert. In der inaktiven Zeit, die mehrere TFU-Taktzyklen dauert, können sich die Signalpegel, die im Dauerbetrieb zu dynamischen Fehlern führen können, stabilisieren.

5.4.1 Teststrategie

Es gibt verschiedene Alternativen, um den Pipeline-Prozessor zu testen:

1. Angenommen, man legt bei einem Test an der ersten Pipelinestufe Testmuster an und liest die Resultate nach der letzten Pipelinestufe aus (siehe Abbildung 5.3), so hat man bei dieser Methode verschiedene Nachteile:

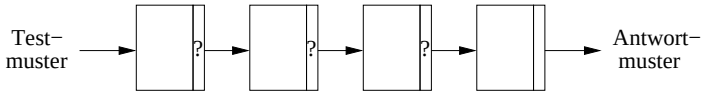


Abb. 5.3. Test einer kompletten Pipeline

   - Man muß bei der Testmustergenerierung das sequentielle Verhalten der Pipeline beachten. Damit in einer hinteren Pipelinestufe das gewünschte Muster angelegt werde kann, muß man den Einfluß der vorangehenden Stufen auf das Muster berechnen. In den hinteren Pipelinestufen sind bestimmte Testmuster nicht oder nur schwer zu realisieren. Als Beispiele könnte man die Muster *binär* 0000... und 1111... zur Detektion von Stuck-at-Fehlern anführen.
   - Fehlerhafte Ausgangsmuster sind schwer zu interpretieren, da die Stufen, welche der fehlerhaften Pipelinestufe folgen, das Muster in der Regel verändern.
   - Die LUT-Funktionen müssen so gewählt werden, daß sie die genannten nachteiligen Effekte minimieren.

2. Eine andere Möglichkeit besteht darin, in der ersten Pipelinestufe Testmuster anzulegen und die Testmuster beim Betrieb der Pipeline nach jeder Stufe zu prüfen (siehe Abbildung 5.4). Dadurch entfällt der Nachteil bei der Interpretation der Ausgangsmuster. Der Nachteil bei der Generierung der Testmuster für hintere Pipelinestufen, die Berechnung des Einflusses der vorangehenden Stufen auf das Muster, bleibt jedoch bestehen.[2]

3. In nahezu allen Pipelinestufen ist es möglich, über Scan-Pfade auf alle Message-Parameter zuzugreifen.[3] Da man über die Scan-Pfade ein Muster lesen und schreiben kann, ist es möglich, die gesamte Pipeline in einzelne Pipelinestufen zu unterteilen und diese unabhängig voneinander zu testen (siehe Abbildung 5.5).

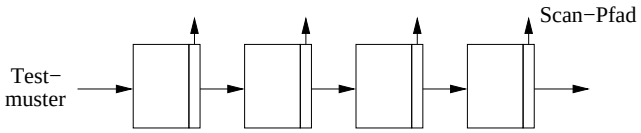


Abb. 5.4. Test einer Pipeline durch Auslesen der auf das Eingangsmuster resultierenden Muster über Scan-Pfade in jeder Pipelinestufe.

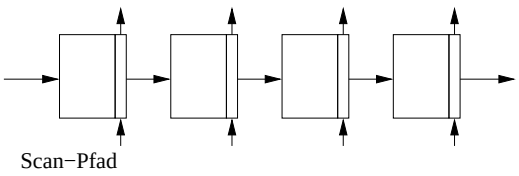


Abb. 5.5. Der Pipeline-Prozessor läßt sich entlang der einzelnen Pipelineregister, die zu Scan-Pfaden zusammengeschlossen sind, in Testklassen unterteilen. Dadurch lassen sich die einzelnen Pipelinestufen unabhängig voneinander testen.

> Wie in Kapitel 2.2.3, Design zur Verbesserung der Testbarkeit, beschrieben ist, wird durch diese Maßnahme die Kontrollierbarkeit und Observierbarkeit vergrößert. Die Generierung eines Tests ist von wesentlich geringerer Komplexität, da nicht die Einflüsse der vorhergehenden Pipelinestufen auf das gewünschte Eingangs-Testmuster berücksichtigt werden müssen, und auch die Ausgangsmuster direkt ausgelesen werden können, ohne daß es notwendig wäre, die Änderungen nachfolgender Stufen zurückzurechnen. Die Fehlerlokalisierung ist effizienter, da die Suche auf die Pipelinestufe beschränkt ist, in welcher der Fehler aufgetreten ist. Durch diese Maßnahme spart man Testentwicklungs- und Ausführungszeit und damit Testkosten.

Man hat sich aufgrund ihrer Vorteile für die dritte Alternative entschieden. Die Pipeline wurde in einzelne Stufen und diese wiederum in einzelne Funktionen unterteilt (siehe Abbildung 5.6). So werden z.B. die drei parallelen Addierer in den ersten Pipelinestufen unabhängig getestet.

[2]Das Vorgehen beim Vergleich der Simulation mit der Hardware (siehe Kapitel 4.4) entspricht dieser Möglichkeit.

[3]Das sind alle Pipelineregister, deren Ausgänge an Ausgangs-Pins von PLDs liegen. Auf die PLD-internen Register kann man nicht zugreifen. Da diesen im selben PLD immer ein Register folgt, auf das man zugreifen kann, ist der PLD über dieses testbar.

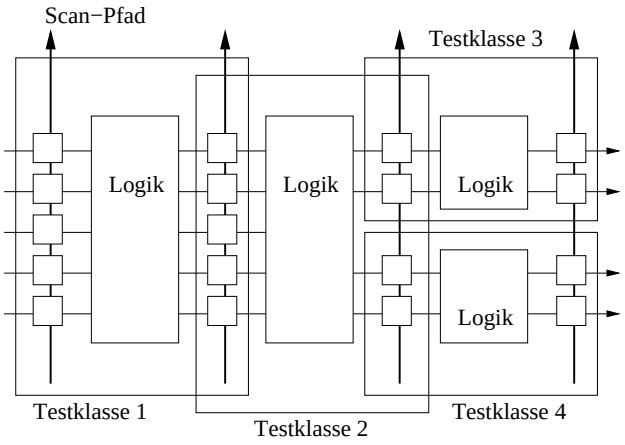


Abb. 5.6. Beispiel für die Unterteilung des Pipeline-Prozessors in mehrere Testklassen. Die letzte Pipelinestufe hat man hier in in zwei Testklassen unterteilt, um ihre beiden Logik-Bausteine gezielt testen zu können.

5.4.2 Durchführung des Tests

Bei den Scan-Pfaden der TFU handelt es sich um ein "ad-hoc"-Design, es entspricht nicht dem Boundary-Scan-Standard. Daher ist es schwierig, die auf den Standard aufbauenden Testeinrichtungen einzusetzen.

Beim herkömmlichen Vorgehen werden die Testmuster vor dem Test entweder automatisch durch eine ATE-Einrichtung oder manuell erzeugt und zur späteren Verwendung gespeichert. Aus diesen Testmustern errechnet man mit einer Fehlersimulation die erwarteten Antwortmuster der zu testenden Schaltung und speichert diese ebenfalls. Bei der Durchführung des Tests verwendet man die gespeicherten Testmuster zu Stimulation der Schaltung, vergleicht die Ergebnisse mit den gespeicherten Antwortmustern und schließt so auf mögliche Fehler zurück.

Ein Nachteil dieser Methode ist, daß eine Fehlersimulation nur im Kontext eines oder mehrerer Fehlermodelle durchgeführt wird. Nur für die in den Modellen enthaltenen Fehler werden die Ausgangsmuster errechnet. Andere Fehler erkennt man zwar durch ihr falsches Ausgangsmuster, ihre Ursache ist jedoch nicht bekannt und muß gesucht werden. Sie setzt außerdem die Kenntnis aller möglichen Fehler innerhalb des Modells voraus. Bei einer Schaltung von der Größe der TFU liegen diese mindestens in der Größenordnung der Anzahl der 20.000 Lötstellen.

Im Gegensatz dazu besitzt man bei der TFU die Möglichkeit, mit dem Mikrocomputer während der Durchführung des Tests die Eingangsmuster zu generieren und die Antwortmuster zu errechnen. Aus folgenden Gründen hat man sich für diese Vorgehensweise entschieden:

Die Komplexität dieser Methode ist wesentlich geringer. Zum einen muß man nicht alle möglichen Fehler schon vor einem Test kennen. Daß der Test die Fehler dennoch effektiv diagnostizieren kann, wird in den nachfolgenden Abschnitten zu sehen sein. Zum anderen benötigt man neben der Software zur Durchführung des Tests keine zusätzliche Software zum Berechnen und Speichern der Test- und Antwortmuster und zur Fehlersimulation. Fehler durch die Verwendung verschiedener Softwareversionen sind ausgeschlossen.

Es stellt sich die Frage, ob diese Vorgehensweise nicht eine längere Laufzeit des Tests zur Folge hat. Wie in Abschnitt 5.4.4 zu sehen ist, verwendet man zur Testmustererzeugung neben fest vorgegebenen Mustern einen Zufallszahlengenerator. Zur Berechnung der Ausgangsmuster kommen bei der TFU meistens einfache logische Funktionen zum Einsatz. Daß ihre Ausführungszeiten unkritisch sind, hat sich in der Praxis bestätigt.

Durch die Verwendung des Mikrocomputers auf der TFU ist man unabhängig von speziellen externen Testgeräten. Der Test konnte sowohl während der Entwicklung und Produktion in Mannheim als auch am Einsatzort im FLT am DESY in Hamburg durchgeführt werden.

5.4.3 Der Algorithmus des Single-Step-Tests

Der Algorithmus zum Test einer einzelnen Pipelinestufe oder Testklasse, so wie sie in Abbildung 5.7 zu sehen ist, funktioniert wie folgt (siehe Abbildung 5.8):

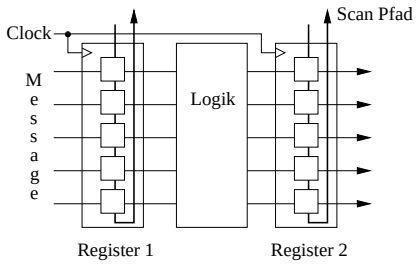


Abb. 5.7. Kombinatorische Logik zwischen den Registern zweier Pipelinestufen

Zuerst erzeugt man ein Testmuster und schreibt es seriell über den Scan-Pfad in das Register 1 der Pipelinestufe vor der kombinatorischen Logik, an dessen Eingängen es sofort anliegt. Die Pipeline wird nun für einen Takt aktiviert, um die Ausgangssignale der Logik in das Register 2 zu übernehmen. Man liest dieses Register seriell aus und vergleicht den Wert mit dem erwarteten Wert, den man zuvor errechnet hat. Stimmen die beiden Werte nicht überein, meldet der Test einen Fehler.

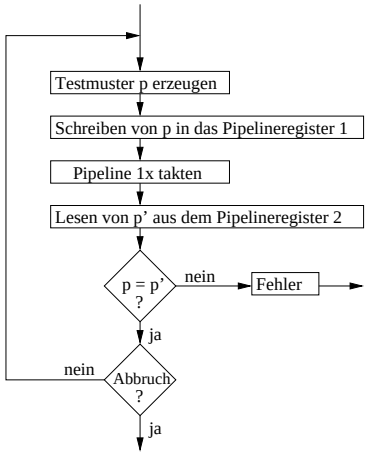


Abb. 5.8. Algorithmus zum Testen einer Pipelinestufe mit kombinatorischer Logik

Enthält eine Testklasse mehrere Pipelinestufen, so funktioniert der Algorithmus entsprechend. Der einzige Unterschied ist, daß man den Pipeline-Prozessor für mehrere Takte aktivieren muß, bis das Muster in das Ausgangsregister übernommen wurde.

5.4.4 Erzeugung der Testmuster

Bestimmte Fehler lassen sich effektiv detektieren, indem man fest vorgegebene Testvektoren verwendet. Stuck-at-Fehler sind mit den beiden Testvektoren *binär* 0000... und 1111... zu finden. Kurzschlüsse zwischen benachbarten Bits findet man durch die Muster *binär* 0101... und 1010.... Da benachbarte Bits nicht unbedingt auch benachbarten Leitungen im Layout entsprechen, verwendet man noch die Muster *binär* 001001..., 010010..., 100100... und 110110..., 101101..., 011011.... Diese werden mit einer Maske auf den Wertebereich der zu testenden Eingänge gekürzt. Die Muster haben eine möglichst kleine Periode, damit sie auch nach einer Kürzung noch zweckmäßig sind.

Um die Fehlerdeckung weiter zu erhöhen, werden Pseudo-Zufallszahlen verwendet. Diese generiert man mit dem Mikrocomputer durch die *rand()*-Funktion und kürzt sie auf die benötigte Bit-Breite.[4]

[4]Durch die Kürzung der Testmuster geht ihre Pseudo-Exhaustivität verloren und der Zufallscharakter ändert sich, da Wiederholungen einzelner Muster möglich sind.

5.4.5 Die Fehlerlokalisation

Findet die Testsoftware einen Fehler, so meldet sie, in welcher Testklasse (siehe Anhang A.1), bei welchem Eingangsmuster der Fehler auftrat. Sie gibt außerdem die erwarteten und detektierten Ausgangsmuster aus. Im folgenden Beispiel handelt es sich um die Testklasse der Pipelinestufe 1-2 mit den Funktionen $\pm dl1 = f_{\pm}(\xi, \omega, \eta, ID)$ in den LUTs Nummer 8 und 9:

```
Test 2: 1(xi,omega,eta,id) -> LUT(8,9) -> 2(negDl,zerDl,posDl)
Test 2.0: error in message parameter 2posDl = 02, correct value 00
```

Durch die Testklasse, in welcher der Fehler auftrat, ist der Ort der Fehlerursache eingeschränkt auf die jeweilige Logik mit ihren Datenpfaden und Registern. In der Hardware beschränkt sich die Fehlerursache damit meist auf nur wenige Bauteile und ihre Verbindungen.

Durch eine Analyse der Ausgangsmuster ist es möglich, den Fehler noch genauer zu lokalisieren. Im Beispiel ist das resultierende Muster *hexadezimal* 02 = *binär* 0000.0010 statt der erwarteten 0000.0000, das weist auf einen Stuck-at-0-Fehler im zweiten Bit des jeweiligen Message-Parameters hin.[5]

In seltenen Fällen kann ein Defekt zu einem sequentiellen Verhalten der Logik führen. Man verwendet in der Regel mindestens 15 Testmuster. Bei einer Wahrscheinlichkeit von 50%, daß der nächste Wert einer Sequenz zum richtigen Ergebnis führt, beträgt die Wahrscheinlichkeit nach 15 Testmuster nur noch $1/2^{15} \approx 0,003\%$. Sequentielles Verhalten wird somit aller Wahrscheinlichkeit nach gefunden.

5.4.6 Test der Scan-Pfade

Eine Voraussetzung für den beschriebenen Testalgorithmus ist die korrekte Funktion der Scan-Pfade. Um sie zu testen schreibt man die oben beschriebenen Testmuster seriell in die Scan-Pfade und liest sie wieder aus. Dieser Test ist in das Single-Step-Test-Programm integriert.

5.4.7 Test der LUTs

Den Speicher der LUTs testet man vor dem Single-Step-Test mit dem oben beschriebenen RAM-Test durch Schreiben und Lesen von Pseudo-Zufallszahlen. Dieser greift über die Schnittstellen zum Mikrocomputer auf das

[5] Es können auch komplexere Fehler vorliegen, z.B. ein Kurzschluß zu einer anderen Leitung auf logisch 0. Eine genauere Analyse mit verschiedenen Testmustern, die in der Regel durchgeführt wird, bringt weitere Aufschlüsse.

RAM zu. Weiterhin ist noch die Schnittstelle zwischen den Datenausgängen der LUTs den Pipelineregistern zu testen (siehe Abbildung 5.9, vergl. Abbildung 5.1 auf Seite 67).

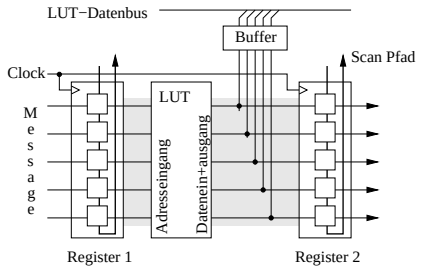


Abb. 5.9. Eine Pipelinestufe mit einer LUT. Grau unterlegt sind die Adreß- und Datenpfade, über welchen der Pipeline-Prozessor die LUT liest.

Test der Datenschnittstelle

Um die Ein- und Ausgänge der LUT gezielt mit effektiven Testmustern prüfen zu können, muß man diese mit darauf abgestimmten Testdaten laden. Ein vollständiges Beschreiben und Umladen aller LUTs dauert ca. 8 s. Müßte man die LUTs für jedes Testmuster vollständig neu laden, nimmt die Dauer des Tests sehr schnell zu. Bei 100 Testmustern beträgt sie dann schon ca. 13 min.

Man spart Zeit, indem man für einen Testdurchlauf nur die jeweiligen von den Testmustern angesprochenen Adressen mit Daten lädt. Auf diese Art und Weise dauert der gesamte Test der Ein- und Ausgänge aller LUTs mit mehreren 100 Testmustern, wie gemessen wurde, nur noch wenige Sekunden.

Der Algorithmus zum Test der Datenschnittstelle funktioniert wie folgt (siehe Abbildung 5.10): Zuerst wird ein Testmuster p erzeugt. Aus diesen erzeugt man durch Maskierung die Muster p_a und p_d. Man schreibt p_d an der Adresse p_a in das RAM. Dazu wird die Adresse p_a über den Scan-Pfad seriell in das erste Pipelineregister geschrieben, dessen Ausgang mit dem Adreßeingang des RAMs verbunden ist (vergl. Abbildung 5.9). Das Datum p_d schreibt man über den LUT-Datenbus in das RAM. Im nächsten Schritt wird der Pipeline-Prozessor für einen Taktzyklus aktiviert. Da im ersten Pipelineregister noch das Muster p_a liegt, selektiert er die LUT an dieser Adresse und liest mit der nächsten aktiven Taktflanke den Inhalt des LUT-RAMs an dieser Adresse in das nächste Pipelineregister. Von dort liest man es über den Scan-Pfad aus und prüft es. Stimmt der gelesene Wert $p_{d'}$ nicht mit dem erwarteten p_d überein, wird eine Fehlermeldung ausgegeben. Diesen Vorgang wiederholt man bis zur gewünschten Anzahl der Testmuster.

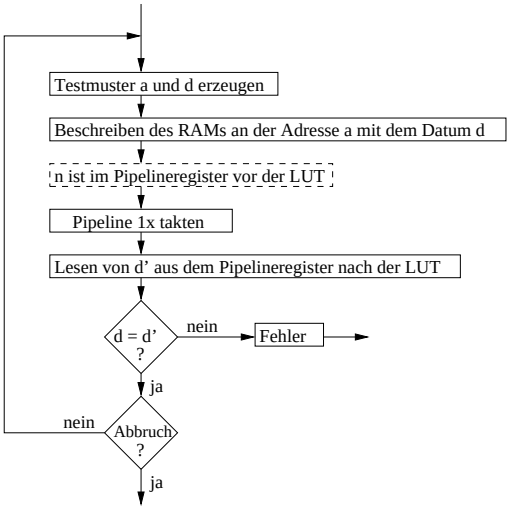


Abb. 5.10. Algorithmus zum Testen der Datenschnittstelle einer LUT

Der Test prüft beim Schreiben den Pfad von der Datenschnittstelle zum Mikrocomputer. Beim Lesen wird der Pfad zum Pipeline-Prozessor überprüft. Stuck-at- und Kurzschlußfehler lassen sich auf diese Weise detektieren.

Mit dieser Strategie werden alle LUTs individuell in einzelnen Testklassen geprüft.

Test des Adreßeingangs

Der Test der Datenschnittstelle findet keine Fehler wie z.B. Stuck-at-Fehler am Adreßeingang des RAMs. Der Test schreibt bei einem solchen Fehler an eine falsche Adresse und liest von dort wieder das erwartete Datum aus. Daher wird der Adreßeingang vorher durch den folgenden Test geprüft (siehe Abbildung 5.11):

Die Speicheradresse 0 wird mit dem Datum 0 beschrieben. Nun lädt man das RAM an den Adressen *binär* 0000.0001, 0000.0010, 000.0100, ... jeweils mit den Werten 1, 2, 3, ..., solange, bis an jedem Pin des Adreßeingangs eine 1 angelegt wurde. Dann liest man wieder die Adresse 0 aus. Erhält man die erwartete 0, so gibt es keine Stuck-at oder Kurzschlußfehler am Adreßeingang.

Existiert z.B. ein Stuck-at-0-Fehler am zweiten Pin, so würde mit der Adresse *binär* 0000.0010 die ursprüngliche 0 überschrieben. Daher liest man den Wert 2 an der Adresse 0 aus. Entsprechendes trifft bei Stuck-at-1-Fehlern zu. Treten bei dem Test mehrere Fehler auf, so können diese auch von Kurzschlüssen zwischen den Adreß-Pins verursacht worden sein.[6]

[6] In der Implementation wird der Test noch mit anderen Testmustern am Adreßeingang

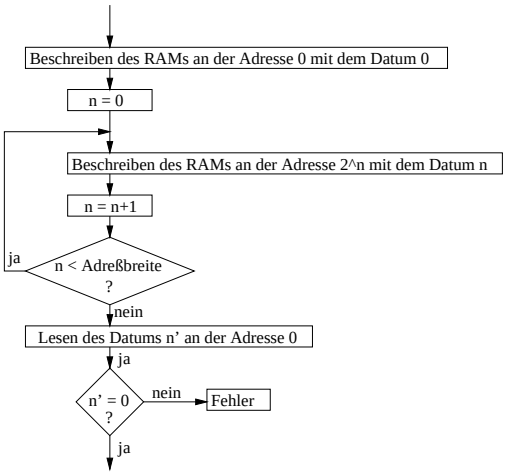


Abb. 5.11. Algorithmus zum Testen des Adreßeingangs einer LUT

5.4.8 Test des Wire-Memorys

Den Speicher des Wire-Memory testet man mit dem RAM-Test von Gläß. Er wird dazu wie das LUT-RAM über die Schnittstelle zum Mikrocomputer beschrieben und gelesen. Die Schnittstelle des Wire-Memorys zu den Empfängern für die Detektordaten überprüft man beim Test der Datenübertragung über die Glasfaserkabel. Der in den Single-Step-Test integrierte Test prüft die Schnittstelle des Wire-Memorys zum Pipeline-Prozessor.

Entsprechend des Tests der Schnittstelle zwischen einem LUT und dem Pipeline-Prozessor beschreibt man das Wire-Memory dabei an der Speicheradresse mit einem Testdatum, an der es vom Pipeline-Prozessor im nächsten Schritt ausgelesen wird (vergl. Abbildung 5.10).

5.4.9 Test der Koinzidenz-Matrix

Zwischen den beiden Teilen der Logik der Koinzidenz-Matrix (siehe Kapitel 3.3.2), in der Pipelinestufe 10, kann man über einen Scan-Pfad auf die Message-Parameter zugreifen. Daher können die beiden Teile der Matrix in verschiedenen Testklassen getrennt getestet werden.

Der Testalgorithmus unterscheidet sich von den übrigen Pipelinestufen nur dadurch, daß zur Berechnung der erwarteten Ausgangsmuster Teile der Simulation eingesetzt werden, die zur Simulation der Matrix verwendeten *operatePipeN()*-Funktionen (*N =10, 13*).

außer 0 durchgeführt. Das ist nicht unbedingt notwendig, da der beschriebene Test schon alle Stuck-at- und Kurzschluß-Fehler abdeckt, trägt aber unwesentlich zur Erhöhung der Testzeit bei.

Das Stall-Signal, welches den vor der Matrix gelegenen Teil der Pipeline anhalten kann, verdient beim Testen besondere Beachtung. Es ist nicht direkt kontrollierbar und observierbar. Kontrollieren kann man es indirekt, indem man einen bestimmten Message-Parameter in der Pipelinestufe 10 entsprechend präpariert. Observieren kann man die Stall-Funktion indirekt durch das Verhalten der vor der Koinzidenz-Matrix gelegenen Pipelinestufen.

Prüft man in Testklassen die vor der Matrix gelegenen Pipelinestufen, muß man verhindern, daß die Stall-Funktion durch zufällige Messages aktiviert wird.

5.4.10 Test der Schnittstellen des Message-Boards zum TFU-Board

Die Schnittstellen zwischen dem Message-Board und dem Pipeline-Prozessor werden in mehreren Testklassen geprüft (siehe Anhang A.1). Dazu wird in eines der 4 bis 6 Empfängerregister ein Testmuster geschrieben und nach wenigen Takten, welche die Message zum Durchlaufen des FIFOs benötigt, in der ersten Pipelinestufe ausgelesen (siehe Abbildung 5.12).

Am Ende der Pipeline kann eine Message aus dem Register nach dem Sender-FIFO ausgelesen werden.[7]

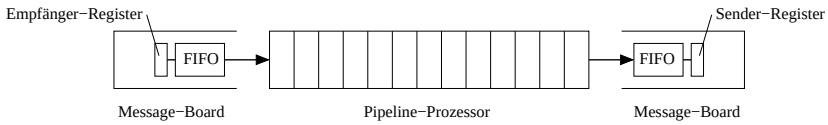


Abb. 5.12. Test der Schnittstellen zwischen dem Message-Board und dem Pipeline-Prozessor.

5.4.11 Effizienz des Single-Step-Tests

Der Single-Step-Test findet statische Fehler. Dazu gehören z.B. Stuck-at- und Kurzschlußfehler, deren Ursachen in defekten oder falschen Bauteilen, Lötstellen, Leiterbahnen oder Steckverbindungen liegen können. Der Test erkennt auch statische Fehler aufgrund falscher Funktion von Bauteilen, wie z.B. durch falsch programmierte oder defekte PLDs.

[7] Der Single-Step-Test der Schnittstelle zwischen dem Pipeline-Prozessor und dem Message-Sender funktioniert in der Realität nicht. Die Ursache kann an einem Fehler in der für den Einzelschrittbetrieb zuständigen Logik der TFU oder an einem Fehler der Software liegen, der nicht gefunden wurden. Da diese Schnittstelle im Dauerbetrieb des Pipeline-Prozessors funktioniert, ist sie jedoch mit dem Burst-Test überprüfbar.

Dynamische Fehler findet er in der Regel nicht, da der Pipelineprozessor nur für einzelne Schritte aktiviert wird, dazwischen ruht er für mehrere TFU-Takte. In dieser inaktiven Zeit können sich gestörte Signalpegel, die im Dauerbetrieb zu dynamischen Fehler führen können, stabilisieren. Zu den Störungen zählen z.B. Reflexionen auf den Signalleitungen aufgrund z.B. defekter oder falsch bestückte Abschlußwiderstände, das Übersprechen von benachbarten Signalleitungen oder Spannungsschwankungen durch das Schalten mehrerer Transistoren zum gleichen Zeitpunkt. Auch defekte ICs können dynamische Fehler verursachen.[8]

Die am häufigsten bei der TFU-Produktion aufgetretenen Fehler waren Löt- und Bestückungsfehler. Da diese Defekte in den meisten Fällen zu statischen Fehlern führten, wurden sie von dem Single-Step-Test entdeckt. Er schränkte durch seine Fehlerlokalisation die Ursache der Defekte auf wenige Bauteile oder Leiterbahnen ein und ermöglichte so eine schnelle Korrektur.

Abschätzung der Testdauer

Daß die Unterteilung der Pipeline in Testklassen bei gleicher Effizienz bzw. Fehlerdeckung einen wesentlich schnelleren Test ermöglicht, soll die folgende Abschätzung zeigen:

Die Testmusterbreite einer typischen Testklasse mit einer LUT beträgt maximal 16 Bit, die Bit-Breite eines LUT-Eingangs. Für einen exhaustiven Test, d.h. unter Verwendung aller möglichen Testmusterkombinationen, benötigt man 2^{16} Muster. Jedes Testmuster muß man über einen Scan-Pfad schreiben und sein Resultat lesen, dazu benötigt man auf 2 durchschnittlich langen Scan-Pfaden $2 \cdot 56 = 112$ Takte. Die Testzeit beträgt damit, unter Vernachlässigung der übrigen von der Software benötigten Zeit, $2^{16} \cdot 112/50\text{MHz} \approx 0.2$ s. Der Test eines der drei Addierer am Anfang der Pipeline dauert $2^{25} \cdot 112/50\text{MHz} \approx 2$ min. Beim Test einer der drei Detektordateneingänge der Koinzidenz-Matrix benötigt man bereits $2^{32} \cdot 112/50\text{MHz} \approx 3$ Stunden. Im Vergleich dazu würde der Test der kompletten ersten Pipelinestufe ca. $2^{59} \cdot 112/50\text{MHz} \approx 40.000$ Jahre dauern. Diese Zeiten reduzieren sich, wenn man nicht alle möglichen Testmuster verwendet, sondern sich auf die 10 fest vorgegebenen Testmuster und eine begrenzte Anzahl von Pseudo-Zufallszahlen beschränkt.

Die Abschätzung zeigt den Vorteil einer Minimierung der Bit-Breite der Testmuster durch eine geeignete Einteilung in Testklassen. Da man Kurzschlüsse zwischen den Signalleitungen benachbarter Testklassen (vergl. Abbildung 5.6, Testklasse 3 und 4) nicht gezielt testen kann, sollte man die

[8] Z.B. haben wenige PLDs bei den TFU-Produktionstests selten auftretende und daher schwer detektierbare dynamische Fehler gezeigt.

Bit-Breiten jedoch nur bis zu einem vernünftigen Maß reduzieren bzw. bei der Einteilung in Testklassen nach Möglichkeit Überschneidungen einbeziehen.[9]

Diese Teststrategie wurde in der Praxis bestätigt, der Single-Step-Test hat seine Fehler meistens schon wenige Sekunden nach seinem Start durch eines der fest vorgegebenen Testmuster detektiert.

Abschätzung der Fehlerdeckung

Die Fehlerdeckung (*engl.* Coverage), ein Maß für die Effizienz, wird normalerweise im Kontext des Fehlermodells, mit dem der Test entwickelt wurde, bestimmt. Man entwickelt einen Test, der so viele Fehler des Fehlermodells wie möglich findet, und beweist, daß der Rest nicht gefunden werden kann [Ait99]. In der Regel geschieht dies mit einer Fehlersimulation. Beim Single-Step-Test hat man eine von Fehlermodellen im diesem Sinne unabhängige Vorgehensweise gewählt. Man generiert bestimmte Testmuster, ohne nachzuweisen, welche Fehler sie konkret im Einzeltest finden, bzw. beweist nicht, daß die übrigen Fehler nicht observierbar sind.

Anhand eines sehr einfachen Fehlermodells soll dennoch versucht werden, die Effizienz abzuschätzen. Wir nehmen an, das Fehlermodell umfasse alle statischen Fehler, die aufgrund defekter Lötstellen entstehen. Weiter nehmen wir an, daß eine Signalleitung im Durchschnitt drei Lötstellen aufweist: am Ausgang eines ICs, am Eingang des nächsten, und am Abschlußwiderstand, der zusammen mit anderen Widerständen im selben Gehäuse eine gemeinsame Masse haben soll (siehe Abbildung 5.13). Daß noch weitere Eingänge, Widerstände oder Bauteile verbunden sein könnten, soll vernachlässigt werden. Ebenso soll vernachlässigt werden, daß als Abschlußwiderstand auch ein Einzelwiderstand angeschlossen sein könnte, welcher zwei Lötstellen besitzen würde.

Ein Defekt in der Lötstelle zum Widerstand führt in der Regel zu Reflexionen auf der Signalleitung, also zu einem dynamischen Fehler, welchen der Test nicht findet. Defekte in den beiden anderen führen zu Stuck-at-Fehlern, die der Test jedoch findet. Da folglich Fehler in 2 der 3 Lötstellen entdeckbar sind, ergibt das eine Fehlerdeckung von $2/3 = 66\%$.

An den Annahmen und Vernachlässigungen sieht man, daß die Größe einen großen Fehler von mindestens $\pm 33\%$ hat und sehr vorsichtig zu interpretieren ist. Eine genaue Aussage setzt ein detailliertes Fehlermodell voraus, wobei es fraglich ist, ob der Aufwand zur Erstellung des Modells im Verhältnis zum Ziel gerechtfertigt ist. Der Aufwand ist sicher größer als die Erstellung

[9] Wie man in Anhang A.1 erkennen kann, überschneiden sich mehrere Testklassen des Single-Step-Tests.

Abb. 5.13. Ein sehr einfaches Fehlermodell (siehe Text)

des kompletten, hier eingesetzten, Tests, da erst alle potentiellen Fehler in der Hardware identifiziert werden müssen, deren Anzahl mindestens in der Größenordnung der 20.000 Lötstellen liegt.

Im Gegensatz dazu bietet das bei der TFU angewandte Testkonzept durch seine Unabhängigkeit von einem solchen Fehlermodell den Vorteil, daß man auch die Ausgangsmuster, die man aufgrund einer Fehlersimulation im Kontext des Fehlermodells nicht erwartet, interpretieren und damit den Fehler diagnostizieren kann.

5.5 Der Burst-Test

Im letzten Abschnitt (Kapitel 5.4.11) wurde beschrieben, daß der Single-Step-Test in der Regel keine dynamischen Fehler findet. Es besteht jedoch die Notwendigkeit, die TFU auch auf dynamische Fehler zu testen, deren Ursachen neben Bauteil- und Leiterbahntoleranzen auch in mangelhaften Lötstellen oder in fehlenden oder defekten Abschlußwiderständen liegen können. Dabei handelt es sich um typische Produktionsfehler.

In Kapitel 2.2.4 wurden die Schwierigkeiten beim Testen dynamischer Fehler beschrieben. Dynamische Fehler aufgrund von Reflexionen am Leitungsende, Übersprechen zwischen Signalleitungen und Spannungsschwankungen durch das gleichzeitige Schalten vieler Transistoren treten in der Regel nur im Dauerbetrieb auf. Daher muß die Schaltung im Dauerbetrieb getestet werden. Dabei können sich auch im Gegensatz zum Single-Step-Test, bei dem der Pipeline-Prozessor nur für ein bis einige wenige Takte aktiviert wird, dynamische Störungen aus früheren Taktzyklen auf die folgenden auswirken. Falls diese Störungen Fehler verursachen, können sie von dem Burst-Test entdeckt werden (vergl. Abbildung 2.13 auf Seite 30).

5.5.1 Teststrategie

Zeitgleich mit der Prototyp-Entwicklung der Hardware hat Gläß die Strategie des Burst-Tests entwickelt (siehe Abbildung 5.14):

Zuerst lädt man die LUTs mit Daten und füllt die Empfänger-FIFOs auf

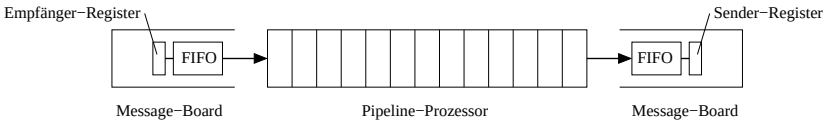


Abb. 5.14. Beim Burst-Test werden die Messages aus einem oder mehreren Empfänger-FIFOs vom Pipeline-Prozessor im Dauerbetrieb verarbeitet und in das Sender-FIFO geschrieben.

dem Message-Board mit Test-Messages aus Pseudo-Zufallszahlen. Dann wird der Pipelinebetrieb gestartet und erst dann angehalten, wenn das Sender-FIFO am Ende der Pipeline mit seiner Kapazität von 256 Messages voll ist.

Im Gegensatz zum Single-Step-Test aktiviert man den Pipeline-Prozessor hier für mindestens so viele Takte, wie die Messages benötigen, um vom Pipeline-Prozessor komplett verarbeitet zu werden. Man startet den Pipeline-Prozessor folglich für einen Dauerbetrieb von mind. 280 Takten und kann daher auch dynamische Fehler finden.

Dieser Test wird wiederholt und der jeweilige Inhalt des Sender-FIFOs mit einem gespeicherten Datensatz verglichen. Findet man Differenzen, so weist dies auf einen Fehler hin.

Zur Lokalisierung eines Fehlers hält man die Pipeline nach einer bestimmten Anzahl von Takten an und vergleicht neben dem FIFO-Inhalt auch die Messages in den Pipeline-Stufen, auf die über Scan-Pfade zugegriffen werden kann. Dies wird jetzt solange wiederholt, und die Anzahl der Takte dabei variiert, bis man die fehlerverursachende Pipelinestufe lokalisieren kann. Sind mehrere Fehler auf dem Board vorhanden, so findet diese Strategie in der Regel den am Anfang der Pipeline liegenden Fehler, falls weiter am Ende liegende Fehler von den Folgefehlern des ersten verdeckt werden. Nach der Korrektur der ersten Fehlerursache muß man den Test und die Suche wiederholen, um weitere Fehler zu finden.

5.5.2 Effizienz des Burst-Tests

Dynamische Fehler, wie sie beim Dauerbetrieb der TFU im FLT auftreten können, kann man mit dem Burst-Test finden, da er dem Dauerbetrieb ähnlich ist. Da außerdem im Gegensatz zum Single-Step-Test mit seiner Unterteilung der Pipeline in einzelne Testklassen die gesamte Pipeline in Betrieb ist, können sich dynamische Störungen überall in der Pipeline bilden und auch überall Fehler verursachen, die man mit diesem Test finden kann.

Dieser Test setzt für die Message- und Detektordaten Pseudo-Zufallszahlen ein. Gegenüber der Unterteilung der Pipeline in einzelne Testklassen besteht der Nachteil, daß die Wahrscheinlichkeit für bestimmte effektive

Testmuster in den einzelnen Pipelinestufen sehr klein sein kann und durch eine große Anzahl von Testdurchläufen mit verschiedenen Testmustern kompensiert werden muß. Eine Unterteilung ist beim Burst-Test jedoch nicht möglich, da das Schreiben und Lesen der Scan-Pfade, das man zum Test der einzelnen Testklassen benötigt, den Pipelinebetrieb für mehrere Takte unterbricht.

Oft führen temporäre Einflüsse aus der Umgebung wie z.B. Spannungsschwankungen oder elektromagnetische Störfelder zu temporären Fehlern. Addieren sich diese temporären Störungen zu vorhandenen dynamischen Störungen, so ist das Auftreten eines Fehlers umso wahrscheinlicher. Aus diesem Grund wird der Test über einen Zeitraum von mehreren Tagen ausgeführt, um auch selten auftretende temporäre Fehler mit einer höheren Wahrscheinlichkeit zu finden. Die Lokalisation eines solchen Fehlers ist sehr schwierig, da normalerweise erst die Ursache des Fehlers in der Umgebung identifizieren werden muß, um seine Rekonstruktion zu ermöglichen. Die Strategie des Burst-Tests zur Lokalisierung eines Fehlers durch Wiederholen des Tests (siehe oben) setzt voraus, daß der Fehler reproduzierbar ist, was bei temporären Fehlern nicht unbedingt trivial sein muß, falls seine Ursache, z.B. eine Störung von außen, nicht bekannt ist.

5.6 Test der Message-Kommunikation

Der Test soll prüfen, ob alle FLT-Boards über ihre Message-Boards richtig miteinander verbunden sind. Er soll feststellen, welche Verbindungen bestehen, und ob die Kommunikation funktioniert. Besonders bei den Message-Boards der Version 2 mit den LVDS-Sender-ICs[10] kommt es vor, daß diese nach dem Einschalten nicht funktionieren. Die Ursache des Verhaltens, das nicht deterministisch ist, liegt an internen Fehlern der verwendeten Sender-ICs. Nach einem Reset (*deutsch* Zurücksetzen) funktionieren sie in der Regel, dies muß man jedoch mit einem Test prüfen.

Hat der Test die Verbindungen überprüft, soll er in einem zweiten Schritt fähig sein, die Qualität oder Fehleranfälligkeit der Verbindungen zu testen.

Prüfen der Verkabelung zwischen den FLT-Boards

Der Algorithmus zum Bestimmen der Verkabelung der Message-Leitungen zwischen mehreren FLT-Boards funktioniert folgendermaßen: (Man führt da-

[10]LVDS: Low Voltage Differential Signaling; Technologie zur differentiellen Hochgeschwindigkeits-Signalübertragung über Leitungen bei kleinem Strom und niederen Signalamplituden.

bei die drei Schritte auf allen Boards synchron hintereinander aus)

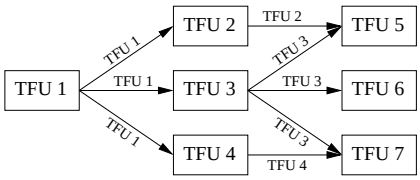


Abb. 5.15. Um die Message-Verbindungen zu identifizieren, senden alle FLT-Boards eine Test-Message mit ihrer Seriennummer.

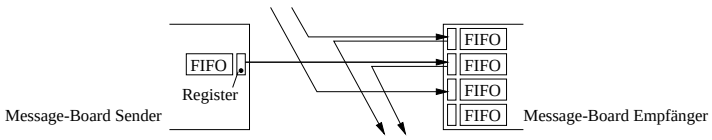


Abb. 5.16. Die Test-Message wird in das Register nach dem Sender-FIFO geschrieben und nach dem Versand aus einem der Empfängerregister ausgelesen.

1. Zuerst setzt man mit einem Reset-Signal alle Message-Boards zurück.

2. In Abbildung 5.16 sieht man, daß sich hinter dem FIFO des Senders ein Register befindet. Auf dieses Register kann man vom Mikrocomputer des angeschlossenen FLT-Boards aus zugreifen. In dieses schreibt man eine Message, welche die Seriennummer[11] des FLT-Boards enthält. Um Übertragungsfehler zu erkennen, beinhaltet die Message zusätzlich ein Testmuster und die invertierte Seriennummer.
   Diese Messages sendet man von allen Message-Boards der verschiedenen FLT-Boards aus ab (siehe Abbildung 5.15).

3. Vor jedem Empfänger-FIFO befindet sich jeweils ein Register, das die letzte empfangene Message enthält. Dies sollte die im letzten Schritt gesandte Message sein. Man liest sie aus dem Register aus und prüft sie auf Übertragungsfehler durch Auswertung des Testmusters und der invertierten Seriennummer.

Sind diese Schritte auf allen Boards korrekt ausgeführt worden, erhält man die Information, welche FLT-Boards miteinander verbunden sind und kann sie mit einer Vorgabe vergleichen.

[11]Das EEPROM des FLT-Boards enthält die Seriennummer zu seiner eindeutigen Identifikation.

Zur Synchronisation der 3 Schritte auf allen FLT-Boards verwendet man einen in der Script-Sprache Expect (siehe Kapitel 3.6.1) programmierten Prozeß. Dieser startet und synchronisiert die zu den Boards gehörenden TRUN-Prozesse.

Test der Qualität der Message-Verbindungen

Der Test der Fehleranfälligkeit von Message-Verbindungen ist in den Burst-Test integriert. Bei der Produktion werden hierzu der Sender und die Empfänger des selben Message-Boards miteinander verbunden und Messages mit Pseudo-Zufallszahlen gesendet und überprüft.

Im FLT muß die bestehende Verkabelung zwischen mehreren Boards auf die Fehleranfälligkeit überprüft werden, dazu muß man eine bestimmte Reihenfolge von Pseudo-Zufallszahlen und Testmustern senden, die sich auf den empfangenden Boards zur Überprüfung rekonstruieren lassen. Die Zufallszahlenfolge konstruiert man, indem man der C-Funktion *rand(x)* jeweils den gleichen Initialisierungswert x vorgibt.

5.7 Zusammenfassung

An die Strategien zum Test der TFU-Hardware werden unterschiedliche Anforderungen gestellt. Bei der Entwicklung der TFU benötigt man Tests zur Überprüfung des spezifizierten Verhaltens. Bei der Produktion sind sie zur Kontrolle der Qualität und zur möglichst genauen Diagnose der Fehler notwendig, um eine schnelle Reparatur zu ermöglichen. Beim Einsatz der TFU im FLT werden Tests zur wiederholten Kontrolle des korrekten Verhaltens gebraucht, da ein Fehlverhalten z.B. zu ausbleibenden Triggerentscheidungen führen und damit die Effizienz des gesamten HERA-B-Experiments mindern könnte.

Der von Gläß und Wurz aufgestellte Testplan für die Entwicklung und Produktion der TFU wurde beschrieben. In dieser Dissertation wurde ein Testplan zum Prüfen der TFU im FLT-Verbund entworfen.

Die Berücksichtigung von Testbarkeitsaspekten beim Design der Hardware ermöglichte es, die Kosten und den Zeitbedarf für das Testen zu reduzieren. Mit dem integrierten Mikrocomputer und der Möglichkeit, über Scan-Pfade auf Knoten in der TFU-Logik zuzugreifen, kann man mit Testsoftware den Datenfluß innerhalb der Hardware observieren und kontrollieren, und so das Verhalten der Schaltung überprüfen, ohne auf teure, externe Testgeräte wie In-Circuit-Roboter angewiesen zu sein. Die TFU-Testsoftware nutzt diese Möglichkeiten.

Die Strategien und Effizienz zweier Tests wurden in der vorliegenden Dissertationsschrift detailliert untersucht:

Bei dem im Kontext dieser Arbeit entwickelten Single-Step-Test wird der Pipeline-Prozessor der TFU mit Hilfe der Scan-Pfade in einzelne Testklassen unterteilt. Diese Methode besitzt verschiedene Vorteile: Die Generierung der Testmuster ist wesentlich einfacher, da man die Schaltung außerhalb der jeweiligen Testklasse in der Regel nicht berücksichtigen muß. Durch kleinere Bit-Breiten der Eingangsmuster läßt sich Testzeit sparen. Zudem können die Fehler schon beim ihrem ersten Auftreten genauer lokalisiert werden.

Da der Pipeline-Prozessor beim Single-Step-Test nur für einzelne Takte aktiviert wird, kann der Test in der Regel keine Delay-Fehler, die aufgrund dynamischer Effekte entstehen können, finden. Der Burst-Test von Gläß hingegen betreibt den Pipeline-Prozessor im Dauerbetrieb und kann daher auch dynamische und temporäre Fehler finden. Man muß allerdings auf die Vorteile einer Unterteilung in einzelne Testklassen verzichten. Beide Tests ergänzen sich aufgrund ihrer Vor- und Nachteile.

5.8 Status und Ausblick

Die Hauptursachen für Fehler waren defekte Lötstellen, man fand pro Platine ca. zwei bis drei. 10% der Mikrocomputer starteten nach der Produktion aufgrund von Lötfehlern nicht. Weitere sehr häufige Fehlerursachen waren falsch bestückte Bauteile und defekte ICs. Defekte Bauteile fand man vor allem unter den PLDs. Wenige PLDs zeigten zeitlich selten auftretende und daher schwer detektierbare dynamische Fehler. Nur ein einziges Mal war eine Leiterbahn unterbrochen. Alterungsbedingte Defekte zeigten Lötstellen und vor allem die Empfänger-ICs für Detektordaten. Da Wärme den Alterungsprozeß beschleunigt, hat Gläß die Kühlung der Empfänger-Bausteine verstärkt. Aus diesem Grund muß man die TFU auch nach der Produktion beim Betrieb in ihrem Einsatzfeld wiederholt testen.

Die Produktion der 90 TFU-Boards wurde abgeschlossen. Alle Fehler konnten lokalisiert und von Gläß und Wurz beseitigt werden. Der Produktionszeitplan wurde eingehalten. Dies wäre ohne Unterstützung durch die beschriebenen Tests zur Qualitätskontrolle und eine schnelle Diagnose der Fehler nicht möglich gewesen.

Die im Rahmen dieser Arbeit entwickelte Single-Step-Testsoftware konnte Defekte in der Hardware des Pipeline-Prozessors, welche statische Fehler verursachten, sehr schnell finden und auf wenige Bauteile genau lokalisieren. Bei den gefundenen Fehlerursachen handelte es sich um defekte Lötstellen und um defekte bzw. falsche Bauteile, die schon oben genannten häufigsten

Fehlerursachen. Waren diese Defekte dagegen bei den Abschlußwiederständen der Leiterbahnen vorhanden, so wurden sie vom Single-Step-Test nicht gefunden, da diese in der Regel dynamische Fehler verursachen.

Um das in der Praxis bewährte Konzept des Single-Step-Tests auch zur Diagnose dynamischer Fehler anwenden zu können, wurden im Rahmen dieser Dissertation entsprechende Konzepte entwickelt. Sie setzen eine Modifikation der zugrunde liegenden Scan-Test-Hardware voraus und werden im nächsten Kapitel vorgestellt.

Die Erfahrungen bei der TFU-Produktion haben die Vorteile einer Integration von Testmöglichkeiten bei dem Design der Schaltung bestätigt. Der Mehraufwand für das Design verringerte die Testkosten bei der Prototyp-Entwicklung, der Produktion und wird sie auch beim Einsatz der Hardware in ihrem Bestimmungsfeld weiterhin verringern. Nur durch effektive Tests, welche auf diesen Testmöglichkeiten aufbauen, konnte der Produktionszeitplan eingehalten werden. Man hat bei der Korrektur der Fehler Zeit gespart, da man dadurch eine höhere Fehlerdeckung und effektivere Diagnose der Fehlerursache erreichte. Außerdem sparte man durch die Integration von Testmöglichkeiten teure, externe Testgeräte. Durch die Unabhängigkeit von diesen Geräten kann man das korrekte Verhalten der TFU jederzeit überprüfen. Diese Tatsache ist gerade beim Betrieb der TFU im FLT bedeutend.

Die Testsoftware wurde für das Testen der TFU im FLT-Verbund an das DESY weitergegeben. Itterbeck hat den Single-Step-Test in die FLT-Diagnose- und Kontrollsoftware des DESYs integriert. Die Erfahrung wird zeigen, ob sich der in diesem Kapitel aufgestellte Testplan und die Testsoftware auch beim Betrieb des FLTs am DESY bewährt.

Kapitel 6

Vorschläge zur Verbesserung der Testbarkeit

Die Produktion der TFU wurde erfolgreich abgeschlossen. Durch die Berücksichtigung von Testbarkeitsaspekten beim Design der Schaltung konnten effiziente Tests entwickelt und angewandt werden. Dadurch konnten Testkosten gespart und der Produktionszeitplan eingehalten werden.

Aus den Erfahrungen, die man beim Test der TFU gesammelt hatte, wurden im Rahmen dieser Dissertation die folgenden Vorschläge zu einer weiteren Verbesserung der Testbarkeit entwickelt, die in künftigen, ähnlichen Entwicklungen berücksichtigt werden können.

6.1 Detektion statischer und dynamischer Fehler mit dem Scan-Test

Der Single-Step-Test hat seine Stärken bei der Diagnose von statischen Fehlern. Dynamische Fehler findet er in der Regel nicht. Der Burst-Dauertest hat dagegen seine Stärken bei der Suche von dynamischen Fehlern, doch muß man auf die Vorteile einer Unterteilung in einzelne Testklassen verzichten. Beide Teststrategien ergänzen sich gegenseitig.

Im Folgenden werden die Unterschiede beider Teststrategien genauer untersucht und ein Vorschlag unterbreitet, der die Vorteile beider Tests vereint.

6.1.1 Test auf statische Fehler mit Hilfe der Scan-Technologie

Die Integration der Boundary-Scan-Technologie während des Designs einer Schaltung ermöglicht es, die Schaltung entlang der Scan-Pfade in einzelne

Testklassen zu unterteilen (siehe Abbildung 6.1). Beim Testen werden über Scan-Pfade Testmuster an die Eingänge der zu testenden Logik gelegt und die Ausgänge nach einer bestimmten Zeit in das Register des dortigen Scan-Pfads übernommen und anschließend überprüft.

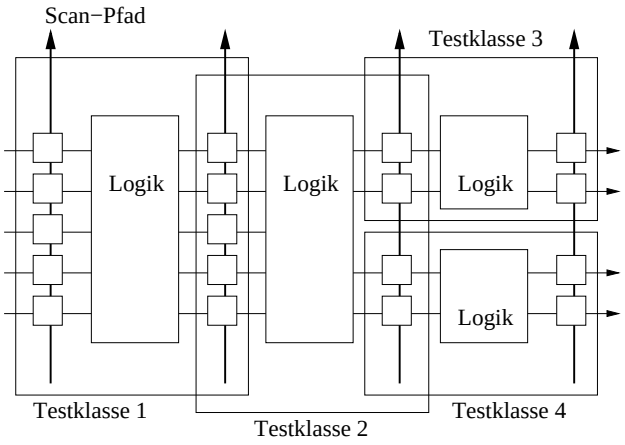


Abb. 6.1. Beispiel für die Unterteilung einer Schaltung in mehrere Testklassen

Die Unterteilung in Testklassen hat den Vorteil, daß bei der Erzeugung der Testmuster und bei der Diagnose eines Fehlers nicht die gesamte Schaltung berücksichtigt werden muß, und man daher effektivere Testmuster erzeugen und den Fehler genauer diagnostizieren kann. Der Single-Step-Test aus Kapitel 5.4 ist ein Anwendungsbeispiel, bei welchem der Pipeline-Prozessor der TFU in einzelne Pipelinestufen und Funktionen unterteilt wird.

In [Ble93] wurde anhand eines Kostenmodells errechnet, daß sich durch Einsatz der Boundary-Scan-Technologie die Hälfte der gesamten jährlichen Entwicklungs- und Produktionskosten einsparen lassen. Zwar fallen durch die zusätzliche Logik beim Design höhere Investitionskosten an, doch spart man bei der Generierung und Durchführung der darauf aufbauenden Tests Zeit und Kosten.

Ein Nachteil dieser Technologie ist, daß die Periode $\triangle t$ zwischen dem Anlegen (Update) und Auslesen (Capture) eines Testmusters beim Testen in der Regel länger dauert als im Echtzeit- bzw. Dauerbetrieb der Schaltung (siehe Abbildung 6.2). Beim Boundary-Scan-Test nach dem IEEE 1149.1 beträgt diese Zeit beispielsweise mindestens 2.5 Taktzyklen [Shi99]. Der Grund für dieses Verhalten liegt in der Logik, die den Testablauf steuert und nach einem Update in der Regel mehrere Takte benötigt, um ein Capture auszulösen. Daher können sich durch dynamische Effekte gestörte Signale, die beim Dauerbetrieb zu Fehlern führen würden, in dieser Zeit stabilisieren.

Ein weiterer Nachteil ist, daß bei einem herkömmlichen Boundary-Scan-

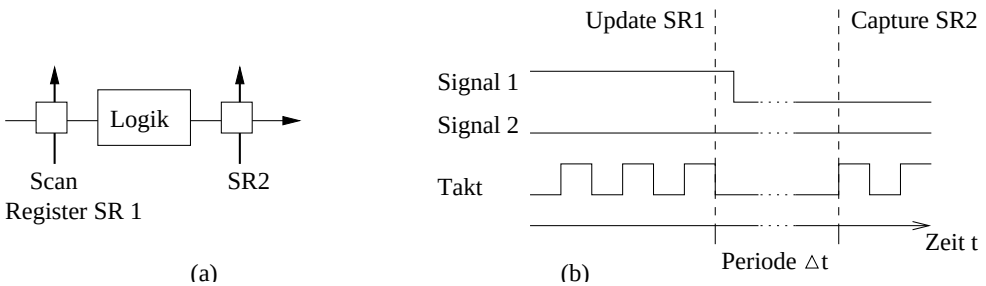


Abb. 6.2. Skizze des Signalverlaufs (b) bei einem Boundary-Scan-Testschritt der Schaltung (a). Je nach Signalpegel, der vor dem Anlegen der Testmusters (Update) vorliegt, und dem Logikwert des Testmusters, erhält man einen Signalwechsel (Signal 1) oder nicht (Signal 2) (gemessen am Ausgang des Scan Registers SR1). Nach der Periode $\triangle t$ wird das Signal in das Scan-Register 2 übernommen (Capture).

Test, z.B. nach IEEE 1149.1, die getestete Logik unmittelbar vor dem Update der Testmuster inaktiv ist. Störungen durch Signale aus früheren Takten konnten sich in dieser inaktiven Zeit stabilisieren. Zu diesen Störungen zählen z.B. Reflexionen am Ende von Signalleitungen aufgrund defekter Abschlußwiderstände, die beim Dauerbetrieb der Schaltung zu Delay-Fehlern führen können. Ebenso fehlen in der Regel Störungen aus der übrigen Schaltung außerhalb der momentan geprüften Testklasse, da diese ebenfalls größtenteils inaktiv ist.

6.1.2 Erweiterung des Boundary-Scan-Tests zur Detektion von Delay-Fehlern

In Abhängigkeit von dem Signalpegel, welcher vor dem Anlegen des Testmusters im ersten Scan-Register SR 1 vorliegt, erhält man am Ausgang des Registers zum Zeitpunkt des Updates einen Signalwechsel (siehe Abbildung 6.2). Hat sich das Signal des Testmusters zum Capture-Zeitpunkt noch nicht stabilisiert, kann der falsche Logikwert in das Register übernommen werden. Den daraus resultierenden Fehler nennt man Delay-Fehler (siehe Kapitel 2.1.3).

Den fehlerverursachenden Defekt findet man nicht, falls durch das Testmuster kein Signalwechsel auftrat und sich das Signal daher schon vorher stabilisieren konnte. Aus diesem Grund strebt man beim sog. 2-Muster- oder Delay-Test an, einen Signalwechsel zu provozieren. Dies erreicht man, indem man die Schaltung im Register SR 1 mit einem ersten Muster initialisiert, bevor man dort das Testmuster anlegt, dessen Resultat man danach im Register SR 2 mißt und auswertet.

Damit sich die Signalstörungen, welche im Dauerbetrieb zu Delay-Fehlern

führen können, nicht stabilisieren können, darf die Periode $\triangle t$ (siehe Abbildung 6.2) zwischen dem Update und Capture des Delay-Tests nicht länger als im Dauerbetrieb sein.

Zu beachten ist außerdem, daß das zuerst im Register SR 1 angelegte Signal beim Test genügend Zeit erhält, sich stabilisieren zu können. Gegebenenfalls kann sonst bei einem Delay des Signals der geforderte Signalwechsel vom ersten zum zweiten Muster nicht zustande kommen und der Delay-Fehler folglich nicht gefunden werden [Che94].

Zur Realisation des Delay-Tests wurden verschiedenen Techniken veröffentlicht, die sich zum Teil durch Modifikation der Logik des Boundary-Scan-Standards nach IEEE 1149.1 realisieren lassen [Gir99][Par00][Pom99][Sav00] [Shi99][Wu99]. In [Nad99] werden beispielsweise ein neues Design der Scan-Zelle und der Boundary-Scan-Kontrollogik vorgestellt, die kompatibel zum Standard sind. Daher lassen sich die darauf aufbauende Test-Hard- und Software unverändert einsetzen.

Bei dem Delay-Test profitiert man von den Vorteilen der Unterteilung in Testklassen und kann Delay-Fehler detektieren. Die Nachteile, keine Fehler durch Störungen aus den Signalwechseln früherer Takte und der aktiven Schaltung außerhalb der Testklasse detektieren zu können, bestehen weiterhin.

6.1.3 Test durch Dauerbetrieb

Beim Dauerbetrieb können alle Störungen aus der gesamten Schaltung und früherer Taktzyklen dynamische bzw. Delay-Fehler verursachen. Eine Möglichkeit zur Detektion dieser Fehler besteht in einem Test durch Dauerbetrieb.

Der Burst-Test aus Kapitel 5.5 entspricht dieser Methode, bis auf die Tatsache, daß er bei der Lokalisierung der Fehler auf die Boundary-Scan-Technologie zurückgreift.

Beim Test durch Dauerbetrieb muß man auf die Vorteile der Unterteilung in einzelne Testklassen verzichten. Diese Methode stellt allerdings gerade bei der Entdeckung eines Fehlers nur wenige Informationen zur Fehlerdiagnose bereit.

6.1.4 Scan-Test im Dauerbetrieb?

Der Bedarf an einer Teststrategie, welche die Vorteile der Unterteilung in Testklassen und des Dauerbetriebs ohne deren Einschränkungen zusammenfaßt, besteht. Wie schon in Kapitel 2.2.4 beschrieben wurde, sind effiziente Tests auf dynamische Fehler ein kritischer Faktor für neuere Halbleiter-Technologien mit höherer Integrationsdichte und steigender Betriebsfrequenz.

Mit den folgenden Anforderungen ließe sich ein solches Testverfahren realisieren (siehe Abbildung 6.3):

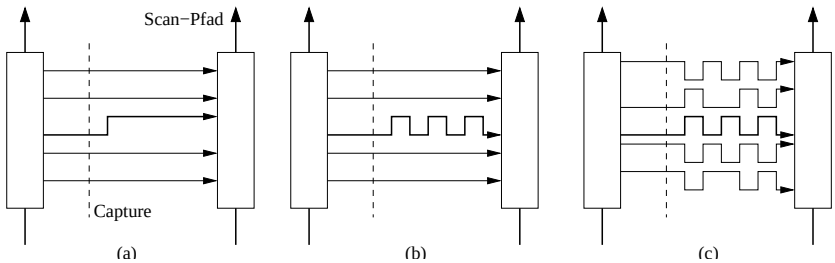


Abb. 6.3. In (a) wird der mittlere Signalpfad zwischen zwei Registern mit Scan-Pfaden durch einen herkömmlichen Scan-Test geprüft. Damit man beim Test auch Fehler findet, die auf dieser Signalleitung von dynamischen Störungen früherer Takte verursacht werden können, generiert man in (b) vor dem Taktzyklus des Tests Signalwechsel. In (c) erzeugt man die Signalwechsel auch auf benachbarten Leitungen, um Fehler aufgrund von Übersprechen oder anderen dynamischen Störungen zu finden.

1. Die Periode $\triangle t$ zwischen dem Update und Capture muß, wie beim Delay-Test, genauso lange wie im Dauerbetrieb sein.

2. Damit auch bis unmittelbar vor dem Test auf einer Signalleitung dynamische Störungen durch Signalwechsel früherer Taktzyklen entstehen können, müssen diese Signalwechsel generiert werden.

3. Alle Scan-Pfade der Schaltung, die nicht zu der momentan im Test befindlichen Testklasse gehören, müssen ebenfalls Signalwechsel produzieren. Auf diese Art und Weise können sie dynamische Störungen erzeugen, die sich wie im Dauerbetrieb auf die aktuelle Testklasse auswirken können.

Zur Realisation der Signalwechsel an den Ausgängen der Scan-Register existieren verschiedene Möglichkeiten:

- Während des Schreibens eines Testmusters über einen Scan-Pfad müssen die Ausgänge der Register aktiv sein. Wie in Abbildung 6.4 zu sehen ist, wechseln die Registerausgänge dann dem Muster entsprechend, das über den seriellen Scan-Pfad geschrieben wird, mit jedem Takt ihren Signalwert. Bei einem Muster *binär* 0101.. wechseln beispielsweise die Signale an den Ausgängen ständig von 0 nach 1 und zurück.
  Diese Möglichkeit ist relativ einfach zu realisieren, die Scan-Pfade der

TFU zeigen beispielsweise dieses Verhalten.[1] Beim Boundary-Test-Standard nach IEEE Std. 1149.1 wird dieses Verhalten jedoch gezielt vermieden. Um stabile Ausgangssignale während des Schreibens und Lesens eines Scan-Pfads zu gewährleisten, liegt zwischen dem Scan-Register und seinen Ausgängen zusätzliche Logik.

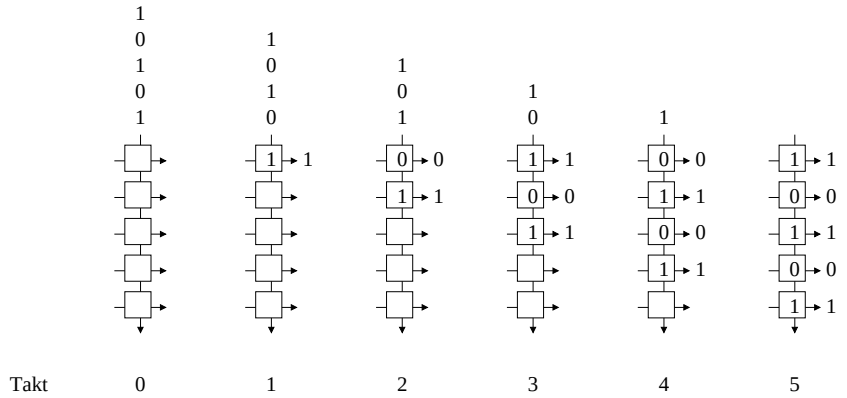


Abb. 6.4. Beim Laden des Registers wird das Muster seriell entlang des Scan-Pfads geschoben. Die Signale an den aktiven Registerausgängen ändern sich dabei dem Muster entsprechend.

- Eine Alternative bietet das Verfahren, in die Ausgänge der Scan-Zellen einen einfachen Testmustergenerator für pseudo-exhaustive Muster oder Pseudo-Zufallszahlen zu integrieren, der beim Test bis unmittelbar zum Update-Capture-Zyklus aktiviert wird.
  Dazu kann man z.B. Linear Feedback Shift Register (LFSR) (siehe Abbildung 6.5) verwenden. In [Pen98] setzt man spezielle LFSR ein, deren Ausgänge beim Test mit gleicher Wahrscheinlichkeit Signalwechsel generieren wie die Registerausgänge im Dauerbetrieb. Die Wahrscheinlichkeit ermittelt man vor dem Design der Schaltung durch eine Simulation.

Sind alle Anforderungen erfüllt, profitiert man sowohl von den Vorteilen der Unterteilung in Testklassen, der Anwendung effektiver Testmuster und effektiver Fehlerdiagnose, als auch von der Möglichkeit, sonst nur beim Dauerbetrieb vorhandene dynamische Fehler zu finden.

[1] Jedoch erfüllt die TFU die erste Anforderung nicht, die Periode $\triangle t$ ist länger als im Dauerbetrieb.

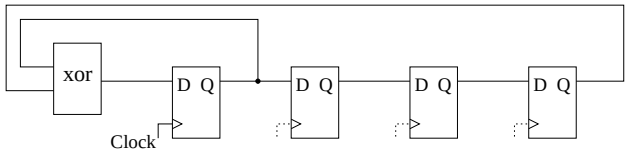


Abb. 6.5. Ein 4-Bit Linear Feedback Shift Register (LFSR). Seine Ausgänge Q_{1-4} generieren die folgende Sequenz: 1111, 0111, 1011, 0101, 1010 [Lal97].

Als Beispiel für die Effizienz dieses Tests sieht man in der Abbildung 6.6 einen Vergleich zu einem Delay-Test mit zwei Testmustern. Dieser Simulation liegt die gleiche fehlabgeschlossene Leitung wie in Abbildung 2.13 auf Seite 30 zugrunde. Im oberen Teil des Diagramms sieht man den Puls, der bei einem Delay-Test mit seinen beiden Testmustern erzeugt wird. Der Delay-Test detektiert zum Capture-Zeitpunkt einen korrekten Logikpegel und entdeckt den Fehler daher nicht. Im unteren Teil erkennt man, daß bei dem in diesem Abschnitt propagierten Test bereits drei Signalpulse vor dem eigentlichen Test-Puls dazu führen, daß beim Capture aufgrund der Störungen durch Reflexionen ein fehlerhafter Signalpegel gemessen wird. Folglich findet dieser Test im Gegensatz zum Delay-Test den Fehler.

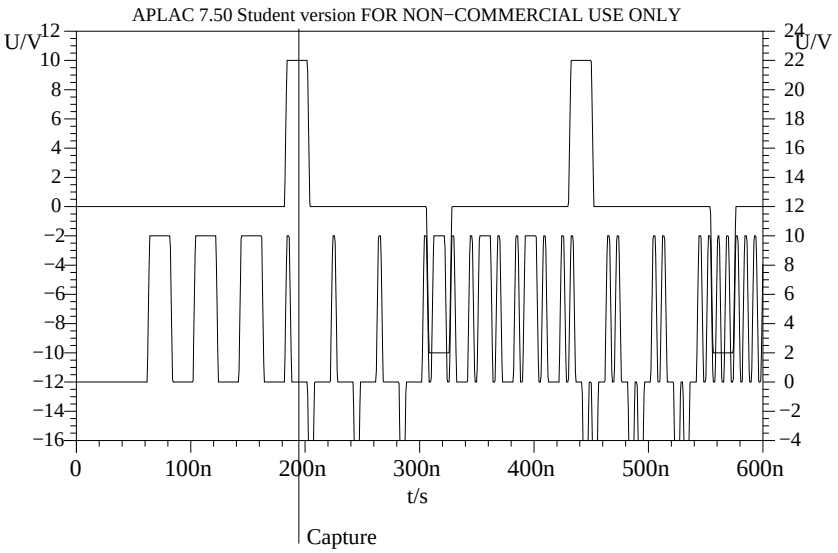


Abb. 6.6. Die Simulation zeigt die Signale am Ende einer fehlabgeschlossenen Leitung (vergl. Anhang B.2). Im oberen Teil sieht man einem einzigen Puls, der am Leitungsende reflektiert wird. Im unteren Teil sieht man die Überlagerung eines oszillierendes Signals mit seinen Reflexionen.

Ergänzend zu diesem Beispiel wird auf den Anhang B.3 verwiesen, in wel-

chem Simulationen weiterer Störeffekte durchgeführt und diskutiert werden.

Bei der Generierung der auf die vorgeschlagene Hardware aufbauenden Test-Software kann man auf die gewohnten Methoden und Fehlermodelle, wie z.B. das Stuck-at-Modell, zurückgreifen, die vom Boundary-Scan-Test zur Diagnose statischer Fehler bekannt sind.

Im Falle der TFU könnte man bei einer Integration der propagierten Technik den Single-Step-Test ohne große Modifikationen zur Diagnose dynamischer Fehler im Pipeline-Prozessor übernehmen.[2] Insbesondere bei einem damit durchzuführenden Dauertest wäre es möglich, im Gegensatz zum Burst-Test temporäre Fehler schon bei ihrem ersten Auftreten möglichst genau lokalisieren zu können (vergl. Kapitel 5.5.2).

6.2 Test durch Variation der Taktrate

Eine weitere Verbesserungsmöglichkeit wäre es, bei einem Test im Dauerbetrieb die Taktfrequenz in kleinen Schritten zu ändern. Dies entspräche einer Variation der Periode $\triangle t$ zwischen den Update- und Capture-Zeitpunkten (vergl. Abbildung 6.2). Dadurch könnte man die Schaltung unter Extrembedingungen testen und kritische Bereiche in der Hardware aufdecken, die beim Dauerbetrieb mit der vorgegebenen Frequenz z.B. bei temporären Störungen leicht zu dynamischen Fehlern führen könnten.

Ein Beispiel zeigt die Simulation in Abbildung 6.7. Im oberen Teil des Diagramms sieht man einen Puls am Ende einer Leitung. Schließt man eine bestimmte Kapazität an das Leitungsende an, so ändert sich der Puls entsprechend des unteren Teils des Diagramms. Der Update-Zeitpunkt entspricht in der Simulation der Zeit $t = 0$. Die Variation der Taktperiode führt zu unterschiedlichen Capture-Zeiten t_{1-3}. Bei t_1 liegt der Signalpegel zwischen 0.8 und 2 Volt und ist entsprechend der TTL-Logik nicht definiert [Häh95]. Der Test erkennt in diesem Fall entweder eine fehlerhafte logische 0 oder die erwartete 1. Zum Zeitpunkt t_2 wird der korrekte Signalpegel erkannt. Bei t_3 detektiert der Test den falschen Logikpegel und damit einen Fehler. t_2 entspricht etwa dem Capture-Zeitpunkt bei normaler Taktrate. Daher tritt unter idealen Betriebsbedingungen bei der Signalform im unteren Teil des Diagramms kein Fehler auf. Schwankungen in der Taktrate (*engl.* Jitter) oder andere Störun-

[2] Zusätzlich müßte auf der TFU-Hardware noch eine direkte Kontrolle und Observation des Feedback-Signals der Stall-Funktion ermöglicht werden (vergl. Abbildung 2.8). Beim momentan eingesetzten Single-Step-Test kontrolliert man es indirekt durch das Schreiben eines bestimmten Wertes in den Scan-Pfad einer bestimmten Pipelinestufe (siehe Kapitel 5.4.9). Dieser zusätzliche Schreibvorgang würde hier jedoch zu Konflikten mit den Anforderungen führen.

gen führen hier dennoch sehr schnell zu logischen Fehlern.

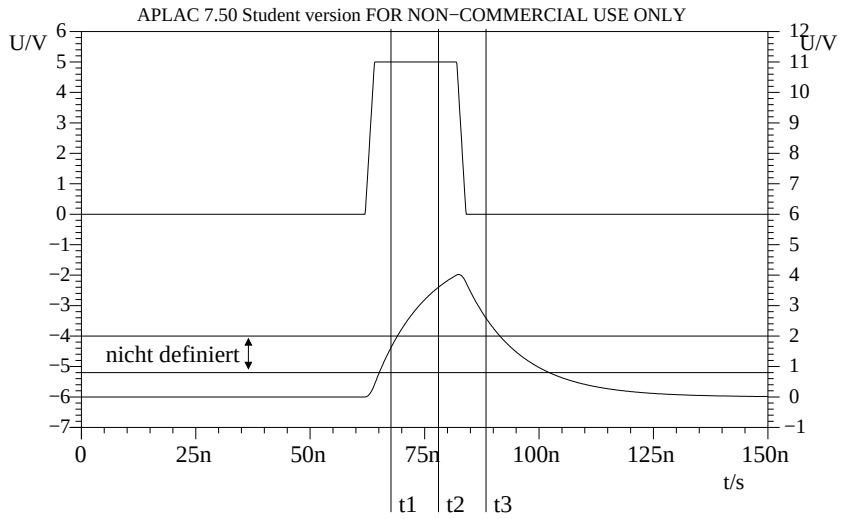


Abb. 6.7. Simulation eines idealen TTL-Pulses (oben) und eines Pulses am Ende einer Leitung mit kapazitiver Last (unten) (vergl. Anhang B.4).

Zur Realisation könnte man einen Frequenzgenerator einsetzen, wie er z.B. auf PC-Mainboards verwendet wird.[3]

Dieser Vorschlag zur Verbesserung der Testbarkeit ließe sich mit jenem aus dem vorigen Abschnitt kombinieren, wodurch man eine Möglichkeit zur effizienten Detektion statischer und dynamischer Fehler mit dem selben Test erhalten könnte.

6.3 Zusammenfassung

Verschiedene Möglichkeiten, auch bei einem Test zur Suche dynamischer Fehler von den Vorteilen einer Unterteilung in Testklassen zu profitieren, wurden in diesem Kapitel diskutiert und anhand von Simulationen demonstriert. Sie könnten in künftigen Entwicklungen ähnlich der TFU berücksichtigt werden. Ein großer Vorteil der hier vorgestellten Strategien ist, daß die zur Diagnose statischer Fehler nach dem konventionellen Scan-Verfahren entwickelte Test-Software ohne große Modifikationen übernommen werden kann.

[3]Die Möglichkeit zum Ändern der Taktfrequenz in kleinen Schritten wird heutzutage auf verschiedenen PC-Mainboards zum Übertakten der CPU eingesetzt.

6.4 Status und Ausblick

In der aktuellen Forschung und Entwicklung besteht, wie schon in Kapitel 2 beschrieben wurde, ein dringender Bedarf an effizienten Diagnosemöglichkeiten für dynamische Fehler, da diese ein kritischer Faktor für die Entwicklung und Produktion künftiger Halbleiter-Technologien sind. Die in diesem Kapitel beschriebenen Vorschläge sollen zur Lösung solcher Probleme beitragen.

Anhand von Simulationen wurde die Effizienz der in diesem Kapitel propagierten Teststrategien demonstriert. Diese Simulationen könnte man in einer ausführlicheren Untersuchung um eine quantitative Analyse der Effizienz unter Berücksichtigung der Rahmenbedingungen unterschiedlicher Halbleiter-Technologien ergänzen.

Die vorgeschlagenen Teststrategien sollten außerdem noch in der Hardware mit realen Störungen überprüft und die Auswirkungen der Störungen quantitativ erfaßt werden. Dies könnte mit einer Versuchsschaltung ähnlich Abbildung 6.3 geschehen.

Anhang A

Dokumentation der Software

A.1 Der Single-Step-Test

Der Single-Step-Test unterteilt den Pipeline-Prozessor in einzelne Testklassen und prüft diese mit zehn fest vorgegebenen Testmustern und Pseudo-Zufallszahlen. Seine Teststrategie wird in Kapitel 5.4 beschrieben und diskutiert.

Der Test wird auf dem VME-Bus-Kontrollrechner (Cetia) unter der Laufzeitumgebung TRUN mit folgendem Kommando gestartet:

```
trun flt-board.cfg tfuTest
```

`flt-board.cfg` ist hierbei die Konfigurationsdatei eines FLT-Boards. Ein Dokumentation zu TRUN und seiner Konfiguration findet man in [Hag97a] oder [Hag97b].

Der Single-Step-Test meldet sich nach seinem erfolgreichen Start mit der Ausgabe eines Menüs:

```
--------------------------------------------------
TFU V3/4 TEST: testing TFU #12 V4

List tests ..................................... l
List test pattern .............................. i

Select test (now:  0) .......................... t
Select test pattern (now:  0) .................. p
Enter number of pattern per test (now:    14) .... n

Run test ....................................... r
Run test with all pattern ...................... u
Run from 0 to selected test with all pattern ..... s
Run all tests with all pattern ................. a
```

```
Start endless test loop ..... e

Quit ..... q
?>
```

In der Kopfzeile des Menüs erscheint die Seriennummer der TFU, auf deren Mikrocomputer die Software gestartet wurde. Aus der Seriennummer leitet die Software die Version der TFU ab (Version 3 oder 4).

Wählt man den Menüpunkt `List test pattern`, so gibt die Software die fest vorgegebenen Testmuster aus:

```
pattern 0: 00000000
pattern 1: FFFFFFFF

pattern 2: 55555555 (01010101)
pattern 3: AAAAAAAA (10101010)

pattern 4: 24924924 (00100100)
pattern 5: 92492492 (10010010)
pattern 6: 49249249 (01001001)

pattern 7: DB6DB6DB (11011011)
pattern 8: 6DB6DB6D (01101101)
pattern 9: B6DB6DB6 (10110111)
```

Die Auswahl `List tests` führt zur Ausgabe aller Testklassen:

```
Test 0: LUT address port
Test 1: shift
Test 2: 1(xi,omega,eta,id) -> LUT(8,9) -> 2(negDl,zerDl,posDl)
Test 3: 1(xi,dxi) -> LUT(0,1) -> 2(n,sf)(dn1=0,dn2=0) -> 3(n,sf)
-> 5(n,sf)
Test 4: 1(xi,eta,id) -> LUT(2,3,4) -> 2(add)(n=0,dn2=0) -> 3(n)
Test 5: 1(xi,dxi,ddxi,nxi) -> LUT(5,6,7) -> 2(add)(n=0,dn1=0) ->
3(n)
Test 6: 1(xi,omega,eta,id) -> LUT(10) -> 2(h) -> 3(omegaHigh,h)
-> 5(omegaHigh,h) -> 7(h)
Test 7: 1(xi,id) -> LUT(12,13) -> 3(xil)
Test 8: 1(ddxi,xi) -> LUT(11) -> 2(l)
Test 9: 1(flag,bx,nTest,nxi) -> 2(flag,nTest,nxi)
-> 3(flag,nTest,bx) -> 5(flag) -> 7(flag)
Test 10: 1(...) -> FIFO,2(flag=1),(muxom=1) -> 16(...)
Test 11: 2(h,l,+-0dl,nxi=0) -> LUT(14,15,16) -> 3(+-0l)
Test 12: 1(xi,dxi,ddxi,eta,id,omega,nxi=0) -> LUT(0-11,14-16) ->
MNADD
-> 3(+-0n,l)
Test 13: 1(xi,dxi,ddxi,eta,id,omega,nxi=1) -> LUT(0-11,14-16) ->
```

```
MNADD
-> 3(+-0n,l)
Test 14: 3(xil,omegaHigh,n=0,flag=1) -> LUT(17,18) -> FIFO
-> 16(ximin,xim,n)
Test 15: 3(n,(h),flag=1) -> FIFO -> 16(n,(h))
Test 16: 3(bx,n=0,l=31,nTest=1) -> WiM -> 7(wd)
Test 17: 3(n=0,l=31,nTest=0) -> WiM -> 7(wd)
Test 18: 3(n,l=31,nTest=0) -> WiM -> 7(wd)
Test 19: 3(n=0,l,nTest=0) -> WiM -> 7(wd)
Test 20: 3(bx,n,l,nTest=1) -> WiM -> 7(wd)
Test 21: 3(zerN,xil) -> XADD -> 5(n,xim)
Test 22: 5(sf)(typeSelect) -> 7(sf)
Test 23: 7(h=0,negWd=0,zerWd,posWd=0,sf=0,flag) -> CoinMatrix
-> 10(x,yo,yu)
Test 24: 7(h=1,negWd=0,zerWd=0,posWd,sf=0,flag) -> CoinMatrix
-> 10(x,yo,yu)
Test 25: 7(h=2,negWd,zerWd=0,posWd=0,sf=0,flag) -> CoinMatrix
-> 10(x,yo,yu)
Test 26: 7(h=3,negWd,zerWd,posWd,sf=0,flag) -> CoinMatrix
-> 10(x,yo,yu)
Test 27: 7(h,negWd,zerWd,posWd,sf,flag) -> CoinMatrix
-> 10(x,yo,yu)
Test 28: 10(x=F00,yo,yu,flag) -> CoinMatrix -> 15(im,km,y,f0)
Test 29: 10(x,yo,yu,flag) -> CoinMatrix -> 13(stall)
-> 15(im,km,y,f0)
Test 30: 15(f0,y,km,im)(muxom=0) -> ADIKM
-> 16(f0,y,omega,im,ikm)
Test 31: 16(id,ikm,(dxi),xi,n=0) -> 17(id,ikl,(dxi),xi)
Test 32: 16(ximin,ikm,id) -> LUT(23) -> 17(dxim)
Test 33: 16(pw,xi,id) -> LUT(24) -> 17(e)
Test 34: 16(xim,im,id) -> LUT(25) -> 17(dxi1)
Test 35: 16(eta,xi,ym) -> LUT(26,27) -> 17(eta,nEta)
Test 36: 16(n,ikm) -> ADDIKL -> 18(ikl)
Test 37: 3(zerN,flag=1)15(im) -> ADDIL -> 17(il)
Test 38: 17(dxi1,e,id) -> LUT(30) -> 18(dxin)
Test 39: 16(id,omega) -> 17(id,omega) -> 18(id,omega)
-> 19(id,omega)
Test 40: 17(dxim,e,id) -> LUT(29) -> 18(dxinm)
Test 41: 17(ikl,il,(h)) -> 18(ikl,il,(h)) -> 19(ikl,il,(h))
-> 20(ikl,il,(h))
Test 42: 16(nTest,all,f0,flag) -> LUT(22) -> 17(nTest,all,mux)
Test 43: 17(nTest,all,eta,xi,mux) -> 18(...) -> 19(...)
Test 44: 19(nTest,all,omega,eta,xi,mux=1)
-> 20(nTest,all,omega,eta,xi) -> 21(...) -> 22(...)
Test 45: 19(ikl,(h),mux=0) -> LUT(31,32) -> 20(xi)
Test 46: 19(dxin,dxinm) -> LUT(33) -> 20(dxim)
Test 47: 19(dxin,dxinm) -> LUT(35) -> 20(dxi)
Test 48: 21(nTest,all) -> (sigLoad,all)
Test 49: 21(xin,nEta) -> LUT(41)(dest2=0) -> 22(dest)
```

```
Test 50: 21(xinm,nEta) -> LUT(42)(dest1=0) -> 22(dest)
Test 51: 20(dxi,ikl,(h)) -> LUT(37) -> 21(xinm)
Test 52: 20(il,dxim,(h)) -> LUT(38) -> 21(xin)
Test 53: 20(dxi,dxim) -> LUT(39) -> 21(ddxi) -> 22(ddxi)
Test 54: 20(dxi,dxim) -> LUT(40) -> 21(nxi) -> 22(nxi)
Test 55: 20(dxi) -> 21(dxi) -> 22(dxi)
```

Folgende Testklassen existieren nur auf einer TFU der Version 3:

```
Test 46: 19(dxiom,id,dxinm) -> LUT(33,34) -> 20(dxim)
Test 47: 19(dxi,id,dxin) -> LUT(35,36) -> 20(dxi)
Test 56: 16(dxi,ddxi) -> LUT(28) -> 17(dxiom)
Test 57: 1(nTest,all,ddxi,dxi,omega,eta,xi) -> SingleStep
-> 22(nTest,all,ddxi,dxi,omega,eta,xi)
Test 58: 1(nTest,all,ddxi,dxi,omega,eta,xi) -> Burst
-> 22(nTest,all,ddxi,dxi,omega,eta,xi)
```

$P_1(m_1, m_2, ..., c_1$=const., ...) -> LUT(l_1, l_2, ...) -> $P_2(n_1, n_2, ...)$ bezeichnet hierbei die jeweilige Testklasse. Zu Beginn eines Testdurchlaufs werden die Message-Parameter m_n mit einem Testmuster beschrieben. Die Message-Parameter c_n = const. erhalten einen konstanten Wert. Die LUTs l_n werden geladen. Nun wird der Pipelineprozessor solange aktiviert, bis die Testmuster als Parameter n_n in die Pipelinestufe P_2 übernommen werden, um anschließend ausgelesen und geprüft zu werden.

Die Testklassen lassen sich in der Schaltskizze der TFU [Glä98b] identifizieren.

Unter den Menüpunkten `Select test`, `Select test pattern` und `Enter number of pattern per test` kann man eine Testklasse *T*, ein bestimmtes Testmuster *P* und die Anzahl *N* der Testmuster bzw. Testdurchläufe pro Testklasse wählen.

Mit `Run test` startet man den Test der Testklasse *T* mit dem Testmuster *P*. Mit `Run test with all pattern` führt man den Test der Klasse *T* mit den Testmustern *0-N* aus. Bei jedem Testdurchlauf werden zuerst die zehn fest vorgegebenen Muster und dann Pseudo-Zufallszahlen verwendet.

Unter dem Punkt `Run from 0 to selected test with all pattern` führt man den Test mit den Testklassen *0-T* mit den Testmustern *0-N* aus. `Run all tests with all pattern` startet den Test aller Testklassen mit den Testmustern *0-N*. Der Test prüft zuerst eine Testklasse mit allen Mustern, bevor er die nächste prüft.

Unter `Start endless test loop` wird der Test unter dem letzten Menüpunkt `Run all tests with all pattern` in einer Endlosschleife gestartet.

Das Beispiel einer Fehlerausgabe wird in Kapitel 5.4.5 beschrieben.

A.2 Der Single-Step-Test des Message-Boards

Alternativ zur im letzten Abschnitt beschriebenen Version des Single-Step-Tests kann man auch eine Version starten, welche zusätzlich die Verbindungen zwischen dem Pipeline-Prozessor und dem Message-Board testet.

Dieser Test setzt voraus, daß das Message-Board an die TFU angeschlossen ist und die Empfänger mit einem aktiven Sender verbunden sind. Vom Sender erhalten sie das zu ihrem Betrieb notwendige Taktsignal (siehe Kapitel 3.4).

Man startet den Test auf dem VME-Bus-Kontrollrechner mit folgendem Kommando:

```
trun flt-board.cfg tfuMbTest
```

Das Menü unterscheidet sich nur geringfügig von dem im letzten Abschnitt beschriebenen Menü, daher soll es hier nicht explizit beschrieben werden.

Abhängig von den aktiven Empfängern ergänzt der Test die folgenden Testklassen:

```
Test: message board shift
Test: receiverN(nxi,xi,dxi,ddxi,eta,omega,all,bx,id,pw,flag) -> 1(..)
Test: all receivers(bx) -> 1(bx)
Test: receiverN(bx,id,pw,flag,e+,e-)
-> 22(dest,nxi,xi,dxi,ddxi,eta,omega,all) -> transmitter(..)[1]
```

Hierbei steht `receiverN` für die aktiven Empfänger `receiver1`, `receiver2`,. ..

[1] Aufgrund der Fußnote 7 auf Seite 81 entfällt in der aktuellen Softwareversion diese Testklasse.

A.3 Die grafischen Benutzerschnittstellen

A.3.1 Die FLT-Console

Die FLT-Console stellt die aktuelle Bestückung eines Crates mit FLT-Boards grafisch dar. Sie ermöglicht den Start von Programmen auf dem VME-Bus-Kontrollrechner (Cetia) und dem FLT-Board-Mikrocomputer (siehe Abbildung A.1).

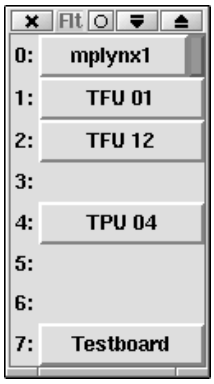


Abb. A.1. Beispiel einer FLT-Console: Die von `boardscan` gefundenen FLT-Boards werden anhand ihrer Seriennummer identifiziert und erscheinen als Button (*deutsch* Knopf) hinter der Nummer ihres Steckplatzes im Crate. Der VME-Bus-Kontrollrechner (hier: mplynx1) befindet sich immer im Steckplatz 0.

Startet man die sog. FLT-Console, so wird auf dem VME-Bus-Kontrollrechner zuerst das Programm `boardscan` ausgeführt. `Boardscan` durchsucht den Adreßbereich des VME-Busses nach FLT-Boards. Die Ergebnisse von `boardscan` werden anschließend grafisch ausgewertet.

Das Aktivieren eines Buttons mit dem Mauszeiger öffnet ein Menü, über das sich z.B. auf dem jeweiligen FLT-Board ein Testprogramm starten läßt.

In regelmäßigen Zeitintervallen wird mit einem `ping`-Befehl der Status des VME-Bus-Kontrollrechners abgefragt, da Zugriffe auf VME-Bus-Adressen zum Absturz dessen Betriebssystems führen können. Der Status wird als grünes bzw. rotes Signal rechts seines Buttons dargestellt.

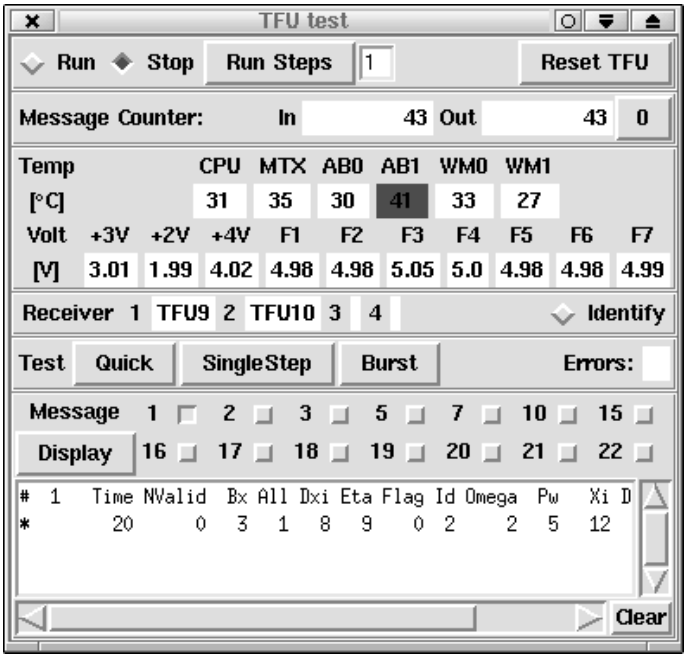
Die FLT-Console wurde in der Script-Sprache Expect programmiert (siehe Kapitel 3.6.1).

A.3.2 Die TFU-Console

Die TFU-Console ist eine grafische Oberfläche zur Kontrolle der TFU (siehe Abbildung A.2). Sie wurde in der Script-Sprache Expect programmiert und

kommuniziert über TRUN mit einem Prozeß auf dem Mikrocomputer (vergl. Kapitel 3.6.1).

Die Software befindet sich in einem frühen Entwicklungsstadium. Da sie gegenüber der textorientierten Software außer der grafischen Oberfläche kaum funktionelle Vorteile bietet, wird ihrer Weiterentwicklung nur eine geringe Priorität zugeschrieben.



Rahmen
1
2
3
4
5
6

Abb. A.2. Die TFU-Console.

Im folgenden werden die einzelnen Rahmen der TFU-Console beschrieben:

1. In diesem Feld kann man den Pipeline-Prozessor anhalten, starten, für eine Anzahl von 1-64 Taktzyklen aktivieren oder die TFU zurücksetzen.

2. Hier wird der momentane Inhalt der Message-Zähler ausgegeben. Sie können auf Null zurückgesetzt werden.

3. Hier werden die auf dem Board gemessenen Temperaturen und Spannungen ausgegeben. Überschreitet ein Wert vorgegebene Grenzwerte, so wird seine Anzeige rot unterlegt.

4. In der Abbildung erkennt man, daß an den Message-Empfänger 1 und 2 die TFUs 9 und 10 angeschlossen sind. Die Erkennung der Message-Verbindungen wird über den Schalter “Identify” gestartet.

5. Hier kann man verschiedene Tests starten: Mit dem Button "Quick" wird ein nur wenige Sekunden dauernder Single-Step-Test mit vorgegebenen Parametern (alle Testklassen, 15 Testmuster) gestartet. Unter "Single Step" erscheint das in der Abbildung A.3 zu sehende Menü zur Auswahl der Single-Step-Parameter und zum Start des Tests.

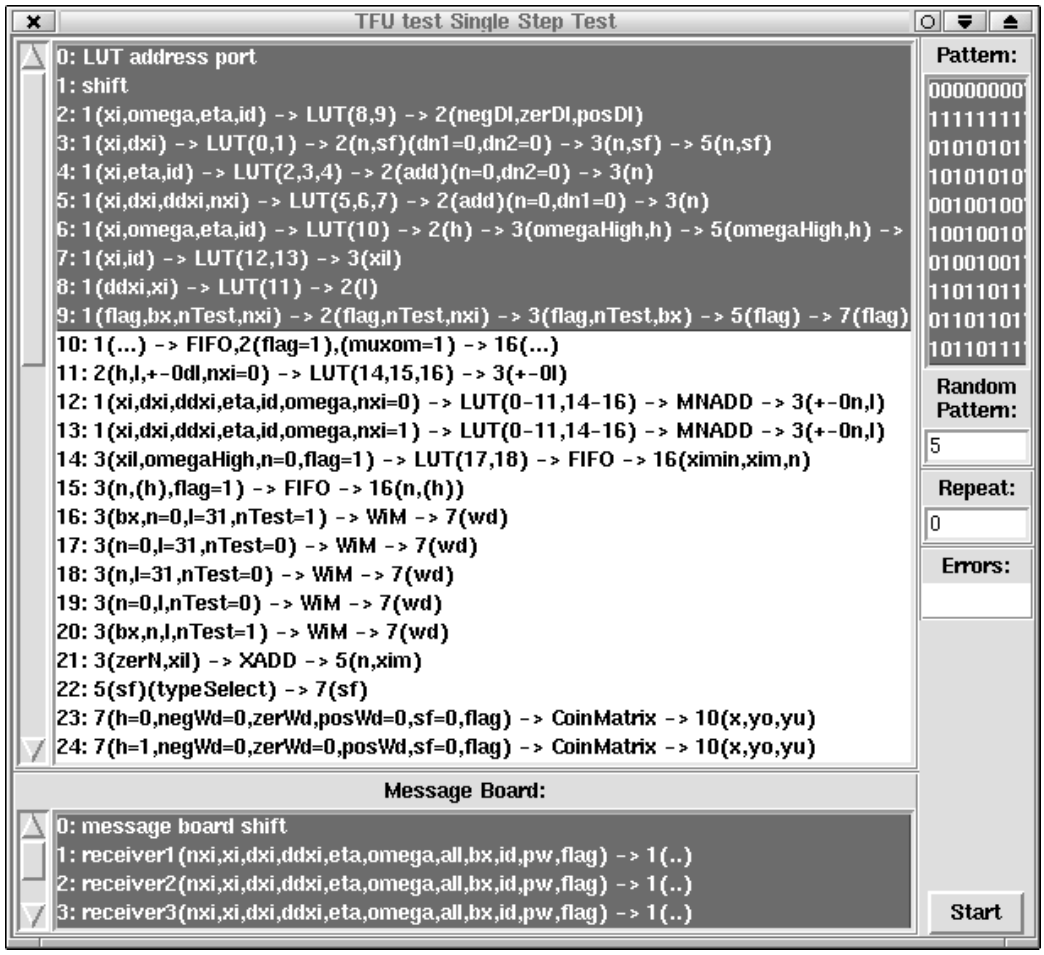


Abb. A.3. Die graphische Oberfläche des Single-Step-Tests.

6. Dieses Feld dient zur Ausgabe von Message-Parametern bei angehaltenem Pipelinebetrieb. Die Parameter werden über die Scan-Pfade aus den Pipelineregistern ausgelesen.

Anhang B

Simulation von Signalstörungen auf Leitungen

Die folgenden Simulationen wurden im Kontext der Dissertation mit APLAC durchgeführt. APLAC ist ein Simulationsprogramm für elektronische Schaltkreise ähnlich SPICE. Verschiedene vordefinierte Komponenten wie Widerstände, Spannungsquellen und Leitungen können zusammengeschaltet und deren Einfluß auf Signale simuliert werden [Apl99].

B.1 Signale einer korrekt- und einer fehlabgeschlossenen Leitung

Die Schaltung aus Abbildung B.1 und die folgende Simulationsvorschrift erzeugen das Diagramm in Abbildung 2.13 auf Seite 30. Die mit den idealen Spannungsquellen erzeugten TTL-Pulse laufen dabei durch ideale Leitungen und werden an deren Ende (Knoten 12 und 22) gemessen. Im erzeugten Diagramm resultiert die doppelte Amplitude des gestörten Signals von 10 V aus der Überlagerung der einlaufenden und reflektierenden Pulse mit 5 V am Leitungsende.

```
* An den Knoten 11 und 21 liegen 2 Spannungsquellen, welche
* TTL-Pulse erzeugen. Die Anstiegs- und Abfallzeiten der
* Plusflanken betragen jeweils 2 ns, die Pulsweiten 18 ns
* und ihre Perioden 40 ns.

Volt V1 11 GND PULSE 0 5 0 2n 2n 18n 40n
Volt V2 21 GND PULSE 0 5 0 2n 2n 18n 40n

* Zwischen den Knoten 11 und 12 bzw. 21 und 22 liegen die
* Leitungen T1 und T2. Sie sind jeweils 18.6 cm lang, ihre
```

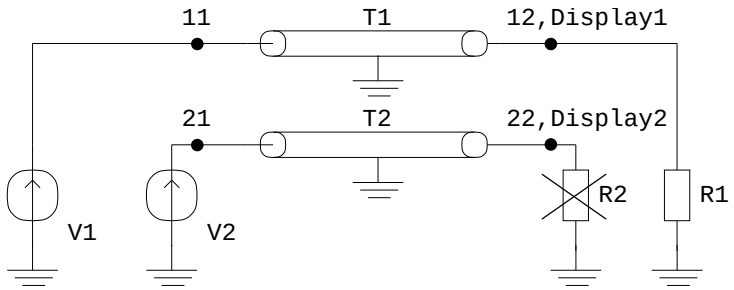


Abb. B.1. Simulation von Signalen auf Leitungen

```
* Wellenwiderstände betragen 470 Ohm.

TLine T1 11 GND 12 GND
+ Z=470 LENGTH=18.6cm
TLine T2 21 GND 22 GND
+ Z=470 LENGTH=18.6cm

* Am Ende der Leitung T1 liegt der Abschlußwiderstand
* R1 = 470 Ohm zur Masse. An der Leitung T2 fehlt er.

Res R1 12 GND 470
* Res R2 22 GND 470

* Es wird eine Zeitanalyse (Transient) durchgeführt. Zwischen
* 0 und 600 ns werden 1000 Schritte berechnet. Der Abstand
* dieser Zeitintervalle ist linear.

Sweep "Ausgangssignale einer korrekt- und einer
+ fehlabgeschlossenen Leitung"
+ LOOP 1001 TIME LIN 0 600n
+ Y  "U" "V" -8 6
+ Y2 "U" "V" -4 24
+ NYTICKS  15
+ NY2TICKS 15
+ NXTICKS  7

* An den Knoten 12 und 22 werden die Spannungen gemessen und
* im Diagramm dargestellt.

DISPLAY
+ Y  "U korrekt" Vtran(12)
+ Y2 "U fehl"    Vtran(22)
EndSweep
```

B.2 Vergleich des Delay-Tests mit dem propagierten Dauerbetriebs-Test

Dieser Simulation liegt die Schaltung aus Abbildung B.1 zugrunde. Die folgende Simulationsvorschrift generiert das Diagramm in Abbildung 6.6 auf Seite 97:

```
* An den Knoten 11 und 21 liegen 2 Spannungsquellen, welche
* TTL-Pulse erzeugen. Die Anstiegs- und Abfallzeiten der
* Plusflanken betragen jeweils 2 ns, die Pulsweiten 18 ns und
* ihre Perioden 40 ns. V1 produziert nur 1 Puls nach 6 Takten
* bzw. 3 Perioden (120ns).

Volt V1 11 GND PULSE 0 5 120n 2n 2n 18n 1000n
Volt V2 21 GND PULSE 0 5 0    2n 2n 18n 40n

* Zwischen den Knoten 11 und 12 bzw. 21 und 22 liegen die Leitungen
* T1 und T2. Sie sind jeweils 18.6 cm lang, ihre Wellenwiderstände
* betragen 470 Ohm.

TLine T1 11 GND 12 GND
+ Z=470 LENGTH=18.6cm
TLine T2 21 GND 22 GND
+ Z=470 LENGTH=18.6cm

* Die Abschlußwiderstände fehlen bei beiden Leitungen.
* Res R1 12 GND 470
* Res R2 22 GND 470

* Es wird eine Zeitanalyse (Transient) durchgeführt. Zwischen
* 0 und 600 ns werden 1000 Schritte berechnet. Der Abstand dieser
* Zeitintervalle ist linear.

Sweep "Delay-Test mit 2 bzw. mehreren Testmustern"
+ LOOP 1001 TIME LIN 0 600n
+ Y  "U" "V" -16 12
+ NYTICKS 15
+ Y2 "U" "V" -4 24
+ NY2TICKS 15
+ NXTICKS 7

* An den Knoten 12 und 22 werden die Spannungen gemessen
* und im Diagramm dargestellt.

DISPLAY
+ Y  "U delay" Vtran(12)
+ Y2 "U fehl"  Vtran(22)
EndSweep
```

B.3 Signale auf parallelen Leiterbahnen

In diesem Abschnitt soll in einer Simulation der Einfluß von Reflexionen am Leiterende, Übersprechen und Ground Bounce auf ein typisches TTL-Signal auf einer 10 cm bzw. 60 cm langen Leiterbahn einer Platine untersucht werden. Man verwendet hierzu das APLAC-Modell *MultiLayerStruct,* die Dimensionen werden dabei ähnlich der TFU gewählt (siehe folgenden Quelltext der Simulation).

Die simulierte Schaltung ist in Abbildung B.2 zu sehen. Um Übersprechen zu simulieren, legte man auf einer parallelen Leitung ein Taktsignal an. Der Ground-Bounce-Effekt wird durch eine oszillierende Spannungsquelle erzeugt.[1] In Abbildung B.3 sieht man die von den Spannungquellen erzeugten Signalformen. Die im Knoten 21 abgegriffenen Signalformen unter dem Einfluß unterschiedlicher Störungen sind im Diagramm der Abbildung B.4 aufgetragen.

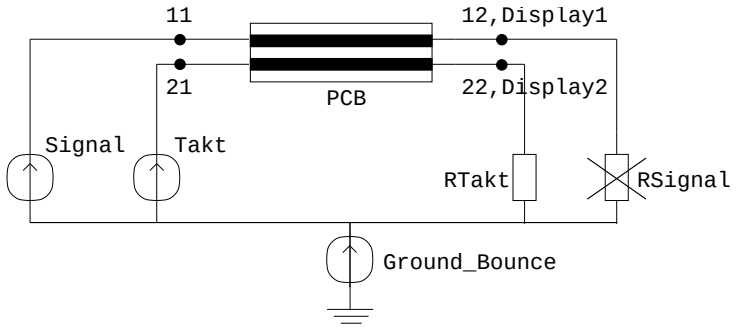


Abb. B.2. Die simulierte Schaltung

```
* An den Knoten 11 und 21 liegen 2 Spannungsquellen,
* welche TTL-Pulse erzeugen. Die Anstiegs- und Abfallzeiten
* der Plusflanken betragen jeweils 2 ns. Die Pulsweite
* beträgt beim "Signal" 18 ns und seine Periode 40 ns,
* beim "Takt" 8ns und 20ns.
* Die Spannungsquelle "GND_BOUNCE" generiert
* Ground-Bounce-Störungen, Spannungsspitzen von
* 0.7 Volt zu den Schaltzeitpunkten des "Signals".

Volt Signal     11 GB  PULSE 0 5   1n 2n 2n 18n 40n
Volt Takt       21 GB  PULSE 0 5   1n 2n 2n 8n  20n
Volt GND_BOUNCE GB GND PULSE 0 0.7 0n 2n 2n 0n  20n
```

[1] Die Signalform und Amplitude des Ground-Bounce-Effekts wurde an die von Wurz in einem experimentellen Aufbau bestimmten angelehnt.

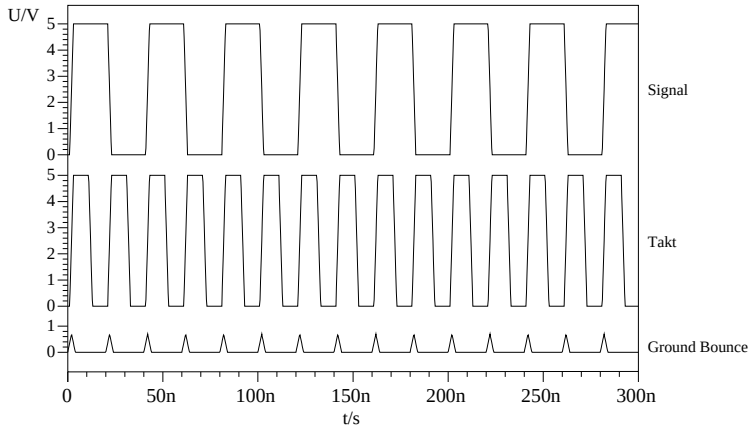


Abb. B.3. Die von den Spannungsquellen erzeugten Signalformen

```
* je nach Simulationsbedarf setzt man die Spannungsquellen
* auf Masse:

*Short Takt       21 GND
*Short GND_BOUNCE GB GND

* Zwischen den Knoten 11 und 12 bzw. 21 und 22 liegen
* die Leitungen T1 und T2. Sie sind jeweils 18.6 cm
* lang, ihre Wellenwiderstände betragen 470 Ohm.

* Auf dem Modell der Platine verlaufen es zwei parallele
* 10 cm lange Leiterbahnen, die jeweils 8 mil breit sind
* und einen Abstand von 8 mil haben (8 mil = 0.2 mm).
* Eine Leiterplattenlage ist 0.18 mm hoch. Das Material
* besitzt eine Dielektrizitätskonstante von 4 (Epoxidharz).

MultiLayerStruct PCB
+ L=10cm LEVEL=0
+ LAYER
+   H=0.18mm ER=4
+ LAYER
+   H=0.18mm ER=4
+   STRIP 11 12 8mil  8mil
+   STRIP 21 22 -8mil 8mil

* Die Abschlußwiderstände der beiden Leiterbahnen führen
* bei 60 Ohm zu den geringsten Reflexionen

Res RSignal 12 GB 60
Res RTakt   22 GB 60

* Es wird eine Zeitanalyse (Transient) durchgeführt.
```

```
* Zwischen 0 und 300 ns werden 1000 Schritte berechnet.
* Der Abstand dieser Zeitintervalle ist linear.

Sweep "Delay-Test mit 2 bzw. mehreren Testmustern"
+ LOOP 1001 TIME LIN 0 300n
+ Y "U" "V" -1 12
*+ Y "U" "V" -12 14
+ NYTICKS 14
+ NXTICKS 7

* An dem Knoten wird 12 die Spannung gemessen und im
* Diagramm dargestellt.

DISPLAY
+ Y "U" Vtran(12)
EndSweep
```

B.3.1 Diskussion der Simulationsergebnisse

In der Abbildung B.4 sieht man, daß die Reflexionen am Leiterbahnende zum größten Störeffekt und damit zu logischen Fehlern führen. Die Ursachen der Reflexionen liegen meist in typischen Produktionsfehlern, fehlenden oder defekten Abschlußwiderständen bzw. defekten Lötstellen an jenen Widerständen.

Die Störungen durch Übersprechen sind bei 10 cm langen Leiterbahnen vergleichbar gering. Bei der Simulation mit 60 cm langen Leiterbahnen war der Effekt größer. In einer realen Schaltung könnte dieser aufgrund zahlreicher vorhandener benachbarter Signalleitungen einen noch größeren Einfluß auf die Signalform verursachen.

Die Störungen durch Ground Bounce sind in der Simulation relativ gering. In einer Schaltung von der Größe der TFU könnte dieser Effekt die hier angenommenen 0.7 Volt übersteigen.

Beide Effekte müssen durch Messungen an einer realen Schaltung, auch unter den Rahmenbedingungen anderer Technologien als der hier angewandten TTL-Logik, z.B. der propagierten Nanometer-Technologie, genauer untersucht werden. Dies könnte an einer Schaltung ähnlich Abbildung 6.3 auf Seite 95 geschehen. Hierbei könnte man auch weitere Fehlerursachen wie z.B. Fehler aufgrund temporärer Störungen analysieren.

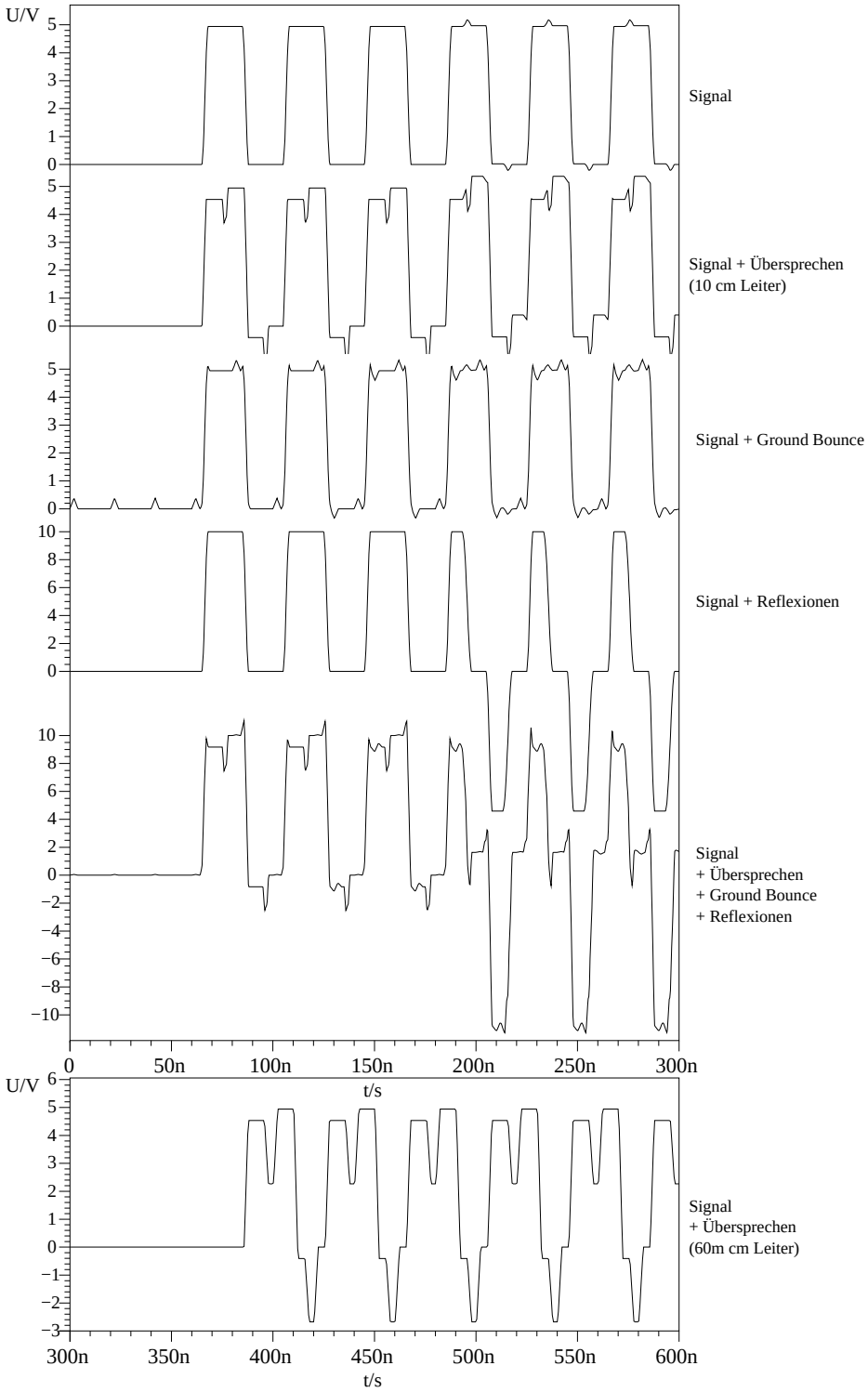


Abb. B.4. Die unter dem Einfluß unterschiedlicher Störungen simulierten Signalformen: Übersprechen, Ground Bounce, Reflexionen am Leiterende.

B.4 Degeneriertes Signal

Die folgende Simulationsvorschrift erzeugt das Diagramm in Abbildung 6.7 auf Seite 99:

```
* An den Knoten 11 und 21 liegen 2 Spannungsquellen, welche
* TTL-Pulse erzeugen. Die Anstiegs- und Abfallzeiten der
* Plusflanken betragen jeweils 2 ns, die Pulsweiten 18 ns
* und ihre Perioden 100 ns.

Volt V1 11 GND PULSE 0 5 0 2n 2n 18n 100n
Volt V2 21 GND PULSE 0 5 0 2n 2n 18n 100n

* Die nächsten beiden Spannungquelle erzeugen die beiden
* Hilfslinien im Diagramm

Volt Vlow  31 GND DC=0.8
Volt Vhigh 32 GND DC=2

* Zwischen den Knoten 11 und 12 bzw. 21 und 22 liegen die
* Leitungen T1 und T2. Sie sind jeweils 18.6 cm lang, ihre
* Wellenwiderstände betragen 470 Ohm.

TLine T1 11 GND 12 GND
+ Z=470 LENGTH=18.6cm
TLine T2 21 GND 22 GND
+ Z=470 LENGTH=18.6cm

* Am Ende der Leitungen liegen die Abschlußwiderstände
* von 470 Ohm zur Masse. An der Leitung T2 liegt zusätzlich
* ein Kondensator mit der Kapazität 50 pF.

Res R1 12 GND 470
Res R2 22 GND 470
Cap C2 22 GND 50p

* Es wird eine Zeitanalyse (Transient) durchgeführt. Zwischen
* 0 und 150 ns werden 1000 Schritte berechnet. Der Abstand
* dieser Zeitintervalle ist linear.

Sweep "Kapazität am Ende einer Leitung"
+ LOOP 1001 TIME LIN 0 150n
+ Y  "U" "V" -7 6
+ Y2 "U" "V" -1 12
+ NYTICKS  14
+ NY2TICKS 14
+ NXTICKS  7

* An den Knoten 12 und 22 werden die Spannungen gemessen und
```

```
* im Diagramm dargestellt.

DISPLAY
+ Y  "U1" Vtran(12)
+ Y2 "U2" Vtran(22)
+ Y2 "Ul" Vtran(31)
+ Y2 "Uh" Vtran(32)
EndSweep
```

Literaturverzeichnis

[Abr90] M. Abramovici, M. B. Breuer, A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, New York, 1990

[Aco99] D. Acostachioaie, *DOC++'s Home Page,* June 1999 (http://www.imaginator.com/doc++)

[Ait99] R. C. Aitken, "Nanometer Technology Effects on Fault Models for IC Testing", *IEEE Computer, Nov. 1999, Vol.. 32, No. 11*, IEEE Computer Society, New York, pp. 46-51

[Apl99] *Online documentation of APLAC software,* Aplac Solution Corporation, Espoo, Finland, 1999 (http://www.aplac.hut.fi/aplac/version/manual.html)

[Aue96] A. Auer, R. Kimmelmann, *Schaltungstest mit Boundary Scan*, Hüthig Verlag, Heidelberg, 1996

[Bel98] I. Belyaev, J. Flammer, T. Fuljahn, E. Gerndt, J. Glaess, A. Groepl, C. Haehnel, D. Kahhnert, R. Maenner, F. Ratnikov, D. Ressing, A. Snijko, T. Wolf, A. Wurz: "The First Level Trigger Simulation for the HERA-B Experiment", *Proc. Computing in High Energy Physics (CHEP)*, Chicago 1998 (CD-ROM)

[Ble93] H. Bleeker, P. v. d. Eijnden, *Boundary-Scan Test, A Practical Approach,* Kluwer Academic Publishers, Dordrecht, 1993

[But98] K. M. Butler, "The Stuck-At Fault: It Ain't Over 'Till It's Over", *Proceedings of International Test Conference 1998,* IEEE Computer Society Press, Los Alamitos, p. 1165

[Che94] C.-A. Chen, S. K. Gupta, *BIST/DFT for Performance Testing of Bare Dies and MCMs,* Conference Proceedings Combined Volumes. Electro/94 International, IEEE, New York, 1994, pp. 803-12

[Che99] K.-T. Cheng, A. Krstic, "Current Directions in Automatic Test-Pattern Generation", *IEEE Computer, Nov. 1999, Vol. 32, No. 11*, IEEE Computer Society, New York, pp. 58-64

[Dat90] *ABEL Design Software, User Manual,* Data I/O Corporation, Washington, 1990

[Eff99] W. Effelsberg, *Skript zur Vorlesung über Rechnernetze, Wintersemester 98/99,* Universität Mannheim, 1999

[Eßn99] H.-G. Eßner, J. Prahm, M. Wielsch, *Linux intern, Technik, Administration und Programmierung,* Data Becker, Düsseldorf, 1999

[Fer98] J. Ferguson, "Flying Probe Test System, Capabilities for Effective Testing", *Proceedings of International Test Conference 1998,* IEEE Computer Society Press, Los Alamitos, p. 1163

[Fla99] J. Flammer, *Simulation des First Level Triggers (FLT) von HERA-B zur Untersuchung der Triggerperformance,* DPG Frühjahrstagung, Heidelberg, März 1999

[Fla00] J. Flammer, *The HERA-B FLT Status Report,* HERA-B Collaboration Meeting, 5. Oktober 2000, DESY, Hamburg

[Ful97] T. Fuljahn, J. Gläß, A. Gröpl, C. Hähnel, T. Wolf, A. Wurz, R. Männer, F. Saadi-Lüdemann, D. Reßing: "Implementation of the HERA-B First Level Trigger", *Proc. Computing in High Energy Physics*, Berlin 1997, pp. 246-248

[Ful98] T. Fuljahn, E. Gerndt, J. Gläß, A. Gröpl, C. Hähnel, D. Kahnert, R. Männer, F. Ratnikov, D. Reßing, F. Saadi-Lüdemann, T. Wolf, A. Wurz: "Concept of the First Level Trigger for HERA-B", *Proc. Xth IEEE RealTime'97 conference*, Beaune, France 1997; *Proc. IEEE Trans. on Nucl. Science*, vol. 45, N4, 1998, pp. 1782-1786

[Ger98] E. Gerndt, H. Itterbeck, F. Ratnikov, D. Ressing, F. Saadi-Lüdemann, *Description of Messages and Process-Logic for the First Level Trigger of HERA-B,* DESY, Hamburg, Apr. 1998

[Gir99] P. Girard, C. Landrault, V. Moreda, S. Pravossoudovitch, A Virazel, "A Scan-BIST Structure to Test Delay Faults in Sequential Circuits", *Journal of Electronic Testing: Theory an Applications, vol. 14, no. 1-2,* Kluwer Academic Publishers, Netherlands, Feb.-April 1999, pp. 95-102

[Glä95] J. Gläß, A. Wurz, T. Wolf, R. Männer, H.-D. Schulz, D. Ressing, *Design of the HERA-B First-Level Trigger,* Proc. IEEE Nucl. Sci. Symp., San Francisco, CA, October 1995, pp. 612-616

[Glä96] J. Gläß, D. Ressing, A. Wurz, *Wire Memory for the FLT,* Lehrstuhl für Informatik V, Universität Mannheim, Jul. 1996

[Glä97] J. Gläß, A. Gröpl, C. Hähnel, D. Reßing, F. Saadi-Lüdemann, T. Wolf, A. Wurz, R. Männer: "Die Implementierung des HERA-B First Level Triggers", *Tagungsband der DPG, Teilchenphysik, T310 Datennahme und Trigger I*, München 1997, S. 27

[Glä98a] J. Gläß, *HERA-B FLT Message Transfer Module,* Lehrstuhl für Informatik V, Universität Mannheim, 1998

[Glä98b] J. Gläß, *TFU Pipeline Flow (V4 12.98),* Lehrstuhl für Informatik V, Universität Mannheim, 1998 (http://www-li5.ti.uni-mannheim.de/mass_par/doc/tfupipev4.pdf)

[Glä99a] J. Gläß, *HERA-B FLT Message Transfer Module LVDS,* Lehrstuhl für Informatik V, Universität Mannheim, 1999

[Glä99b] J. Gläß, A. Gröpl, C. Hähnel, T. Wolf, A. Wurz, R. Männer: "One Terabit per Second Realtime Processing: The HERA-B first Level Trigger", *Proc. Int'l Conf. on Parallel and Distributed Processing Techniques and Applications '99, Vol. III,* Las Vegas, 1999, pp. 1169-1174

[Häh95] C. Hähnel, *Verschiedene Konzepte zur digitalen Hochgeschwindigkeitsübertragung über Leitungen,* Diplomarbeit, Universität Heidelberg, 1995

[Hag97a] C. Hageböke, *Benutzer-Dokumentation für die Systemsoftware des FLT-Testboards*, Universität Mannheim, 1997

[Hag97b] C. Hageböke, *Systemsoftware des First Level Triggers für das HERA-B Experiment am DESY*, Diplomarbeit, Universität Mannheim, 1997

[Har95] E. Hartouni et.al., *HERA-B: An Experiment to Study CP Violation in the B System Using an Internal Target at HERA Proton Ring,* Design Report, DESY-PRC 95/01, DESY, Hamburg, 1995

[Her98] Hera-B collaboration: "DAQ Online Software and the Run Control System in the HERA-B Experiment", *Proc. Comp. in High Energy Physics*, 1998

[Hin92] H. Hinsch, *Skript zur Vorlesung über die Elektronik, Wintersemester 91/92,* Universität Heidelberg, 1992

[Iee90] "IEEE Standard Test Access Port and Boundary-Scan Architekture", *IEEE Standard 1149.1-1990,* IEEE Standards Board, New York

[Jus94] O. Justus, *Berechnung linearer und nichtlinearer Netzwerke mit PSpice-Beispielen*, Fachbuchverlag, Leipzig, 1994

[Kap99] R. Kapur, T. W. Williams, "Tough Challenges as Design and Test Go Nanometer", *IEEE Computer, Nov. 1999, Vol. 32, No. 11*, IEEE Computer Society, New York, pp. 42-45

[Kuh96] G. Kuhr, *Inbetriebnahme und Test von Komponenten des HERA-B First Level Triggers,* Diplomarbeit, Universität Heidelberg, 1996

[Lal97] P. K. Lala, *Digital Circuit Testing and Testability*, Academic Press, San Diego, 1997

[Lat99a] *Introduction to Boundary Scan Test and In-System Programming,* Lattice Semiconductor Corporation, Nov. 1999

[Lat99b] *ISP Overview,* Lattice Semiconductor Corporation, Nov. 1999

[Leg97] I. Legrand, M. Medinnis, D. Ressing, J. Zweizig, *Proposal for a HeraB Message Handler Interface,* DESY, Hamburg, Jun. 1997

[Lib96] D. Libes, *Exploring Expect: A Tcl-Based Toolkit for Automating Interactive Programs*, O'Reilly and Associates, Sebastopol, 1996

[Män95] R. Männer, *Entwicklung und Bau eines First-Level-Triggers für das Vorhaben: "Experiment zum Studium der CP-Verletztung in Proton-Kern-Reaktionen am Speicherring HERA",* Antrag auf Erlangung von Fördermitteln für die Entwicklung des First-Level-triggers vom 27. April 1995, Lehrstuhl für Informatik V, Universität Mannheim

[Mün00] N. Münch, P. Richter, *Ground Bounce - Die Pest des Testens?,* Fa. Göpel Elekronik GmbH, Jena, 2000

[Nad99] B. Nadeau-Dostie, J. F. Cote, H. Hulvershorn, S. Pateras, "An Embedded Technique For At-Speed Interconnect Testing", *International Test Conference 1999, Proceedings,* Int. Test. Conference, Washington, 1999, pp. 431-8

[Nee99] W. M. Needham, "Nanometer Technology Challenges for Test and Test Equipment", *IEEE Computer, Nov. 1999, Vol. 32, No. 11*, IEEE Computer Society, New York, pp. 52-57

[Nör99] M. Nörenberg, *Progress of FLTSIM,* DPG Frühjahrstagung, Heidelberg, März 1999

[Pag91] B. Page, *Diskrete Simulation, Eine Einführung mit Modula-2,* Springer-Verlag, Berlin, 1991

[Par00] S. Park, T. Kim, "A New 1149.1 Boundary Scan Design for the Detection of Delay Defects", *Proceedings Design, Automation and Test and Exhibition 2000,* IEEE Comput. Soc., Los Alamitos, pp. 458-62

[Par98] K. P. Parker, *The Boundary-Scan Handbook, Second Edition, Analog and Digital,* Kluwer Academic Publishers, Boston, 1998

[Rat96] *Round-Trip Engineering with Rational Rose/C++ 4.0*, Rational Software Corporation, Santa Clara, Nov. 1996

[Pat98] J. H. Patel, "Stuck-At Fault: A Fault Model for the Next Millenium", *Proceedings of International Test Conference 1998,* IEEE Computer Society Press, Los Alamitos, p. 1166

[Pen98] R. Pendurkar,A. Chatterjee, Y. Zorian, "Synthesis of BIST hardware for performance testing of MCM interconnections", *1998 IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers*, ACM, New York, 1998, pp. 69-73

[Pov93] B. Povh, K. Rith, C. Scholz, F. Zetsche, *Teilchen und Kerne, Eine Einführung in die physikalischen Konzepte,* Springer, Heidelberg, 1993

[Pom99] I. Pomeranz, S. M. Reddy, "On Achieving Complete Coverage of Delay Faults in Full Scan Circuits using Locally Available Lines", *Proceedings of International Test Conference 1999,* IEEE Computer Society Press, Los Alamitos, pp. 923-31

[Rai98] P. Raines, *Tcl, Tk: kurz & gut,* O'Reilly, Köln, 1998

[Rai99] P. Raines, J. Tranter, *TCL/TK in a Nutshell,* O'Reilly, Sebastopol, 1999

[Rea98] J. Rearick, "Buying Time for the Stuck-At Fault Model", *Proceedings of International Test Conference 1998,* IEEE Computer Society Press, Los Alamitos, p. 1167

[Rei97] M. Reisch, *Elektronische Bauelemente: Funktion, Grundschaltungen, Modellierung mit SPICE,* Springer, Berlin, 1997

[Res96] D. Ressing, F. Saadi-Lüdemann, *The FLT Message and Process-Logic Description,* DESY, Hamburg, Aug. 1996

[Rus98] B. Russel , "ITC 1998 Panel Discussion: Flying Probers - A New Era in Loaded Board Fixtureless Test", *Proceedings of International Test Conference 1998,* IEEE Computer Society Press, Los Alamitos, p. 1162

[Sak67] A. D. Sakharov, "Violation of CP Invariance, C Asymmetry and Baryon Asymmetrie of the Universe" *ZhEFT Prisma 5,* pp. 32-35, 1967

[Sav00] J. Savir, "Distributed BIST architecture to combat delay faults", *Journal of Electronic Testing: Theory & Applications (Jetta)*, vol. 16, no. 4, Kluwer Academic Publishers, Netherlands, Aug. 2000, pp. 369-80

[Sch95] H. Schildt, *C++: The Complete Reference, Second Edition,* Osborne McGraw-Hill, Berkeley, 1995

[Sch97] M. Schader, S. Kuhlins, *Programmieren in C++: Einführung in den Sprachstandard,* Springer, Berlin, 1997

[Shi99] J. Shin, H. Kim, S. Kang, "At-Speed Boundary-Scan Interconnect Testing in a Board with Multiple System Clocks", *Design, Automation and Test in Europe Conference and Exhibition 1999, Proceedings,* IEEE Computer Society, Los Alamitos, pp. 473-7

[Ste95] W. R. Stevens, *Programmierung in der UNIX-Umgebung: Die Refernez für Fortgeschrittene,* Addison-Wesley, Bonn, 1995

[Wag99] K. D. Wagner, "Robust Scan-Based Logic Test in VDSM Technologies", *IEEE Computer, Nov. 1999, Vol. 32, No. 11*, IEEE Computer Society, New York, pp. 66-75

[Wel96] B. Welch, *Programmieren in TCL und TK,* Prentice Hall, München, 1996

[Woj88] H. Wojtkowiak, *Test und Testbarkeit digitaler Schaltungen*, Teubner, Stuttgart, 1988

[Wol98] T. Wolf, *Die Systemsoftware für den First Level Trigger des HERA-B Experiments,* Dissertation, Universität Mannheim, 1998

[Wu99] Y. Wu, P. Soong, "Interconnect Delay Fault Testing with IEEE 1149.1", *Proceedings of International Test Conference 1999,* IEEE Computer Society Press, Los Alamitos, pp. 449-57

Danksagung

Mein Dank gebührt allen, die mich in den letzten Jahren bei der dieser Dissertation zugrunde liegenden Arbeit unterstützt haben, insbesondere den Mitarbeiteitern des Lehrstuhls für Informatik V der Universität Mannheim, Herrn Prof. Dr. Reinhard Männer, Herrn Dr. Joachim Gläß, Herrn Andreas Wurz, Herrn Dr. Thomas Wolf und Herrn Alexander Gröpl für die fruchtbare Zusammenarbeit innerhalb des Projekts.

Ferner danken möchte ich in diesem Kontext auch Herrn Prof. Dr. Peter de With für unser motivierendes Gespräch, das mir zahlreiche zusätzliche Anregungen gab.

Stellvertretend für die Mitarbeiter des HERA-B-Experiments am DESY in Hamburg seien hier insbesondere Herr Dr. Ekkehard Gerndt, Herr Dr. Heiko Itterbeck und Herr Michael Nörenberg genannt.

Danke möchte ich auch Herrn Jeff Lin und Herrn Gerald A. Tobolewski von der Firma Elito-Epox, die mich über die verschieden Tests, welche ihre PC-Mainboards bei der Produktion unterlaufen, informiert haben.

Mein ganz besonderer Dank gilt auch der Fakultät für Volkswirtschaftslehre der Universität Mannheim, insbesondere Herrn Prof. Dr. Horst Stenger, Herrn Prof. Dr. Klaus Winkler und Herrn Prof. Ph.D. Axel Börsch-Supan, welche es mir durch meine Anstellung an der Fakultät ermöglichten, diese Arbeit durchzuführen, und mich auch außerhalb der Promotion unterstützt haben. Ich danke Frau Brunhild Griesbach, Herrn Jürgen Müller, Frau Dr. Angelika Eymann und Herrn Dr. Thorsten Lindenbauer für die gute Zusammenarbeit.

Nicht unerwähnt bleiben dürfen in diesem Kontext die studentischen Mitarbeiter des PC-Pools der Fakultät: Herr Thorsten Dencker, Herr Peter Preuß, Herr Tobias Schöpfner, Herr Henning Colsman-Freyberger, Herr Stefan Holland-Letz, Herr Heiko Kopitzki, Herr Cafer Sengönül, Herr Dietmar Kolb, Herr Siegfried Neumann, Herr Roman Seeberger, Herr Martin Gottron, Herr Andreas Spolwind, Herr Jens Grüning und alle Aufsichtskräfte. Ihren gebührt Dank für ihren ganz besonders motivierten Einsatz bei unserer gemeinsamen Aufgabe, der Administration des PC-Pools. Dadurch

haben sie mir den nötigen Freiraum geschaffen, mich auf meine Dissertation zu konzentrieren.

Ein ganz herzlicher Dank gebührt auch meinen Eltern, Großeltern und meinen Freunden für ihre Unterstützung und Frau Anne Grüneisen, die mich immer wieder zum Schreiben motiviert und bei den Korrekturen geholfen hat.